Combining Meta-Heuristics to Effectively Solve the Vehicle Routing Problems with Time Windows

Vincent Tam

vtam@eee.hku.hk
Department of Electrical and Electronic Engineering, The University of Hong Kong
Pokfulam Road, Hong Kong.

K.T. Ma

makengte@comp.nus.edu.sg
Department of Computer Science
National University of Singapore
Lower Kent Ridge Road, Singapore 119260.

Abstract. The vehicle routing problems with time windows are challenging delivery problems in which instances involving 100 customers or more can be difficult to solve. There were many interesting heuristics proposed to handle these problems effectively. In this paper, we examined two well-known meta-heuristics and carefully combined the short-term and long-term memory-like mechanisms of both methods to achieve better results. Our prototype was shown to compare favorably against the original search methods and other related search hybrids on the Solomon's test cases. More importantly, our proposal of integration opens up many exciting directions for further investigation.

Keywords : Vehicle Routing Problems, Meta-heuristics, Guided Local Search, Tabu Search, Search Hybrids.

1. INTRODUCTION

Many delivery problems in real-world applications such as the newspaper delivery and courier services can be formulated as the (capacitated) vehicle routing problems (VRPs) [10] in which we are interested to route a number of vehicles with limited capacity in order to satisfy all the customers' requests with the minimal operational cost as usually measured by the number of vehicles used multiplied by the total distance travelled. Often, a customer may specify a time-window with the earliest and latest time for the delivery to occur, which gives rise to the VRP with time windows (VRP-TWs). In other words, a vehicle must arrive at a customer within the duration as specified by that customer in a VRP-TWs. The arrival of a vehicle before the earliest time as specified by a customer in a VRP-TWs will result in idle time. On the other hand, a vehicle is not allowed to reach a customer after the specified latest time. Moreover, a service time is commonly associated with servicing each customer. A real-life example of the VRP-TWs is the newspaper delivery in which each customer often requests the newspaper to be delivery within a given period (time window) of a day. Unfortunately, the VRP-TWs are shown to be *NP*-complete, implying an exponential growth in the time complexity for a general algorithm to solve any of these delivery problems in the worst case. In practice, there are many instances of VRP-TWs involving 100 customers [1, 10] or more [9] which are readily difficult to solve optimally [1, 3, 6]. In the past two decades, VRP-TWs, due to its challenging nature and practical values, have continuously attracted many interesting proposals of useful heuristics and search algorithms [1, 3] to effectively solve the problems in the areas of Artificial Intelligence [3], Constraint Programming [1] and Operations Research [3].

Among the heuristics [1, 2] proposed to solve the vehicle routing problems, there are some interesting proposals to initialize the search while others are targeted at advancing the search from a given initial state in an effective manner. As far as search initialization is concerned, there are two useful heuristics, namely the push-forward insertion heuristic (PFIH) [6] and the virtual vehicle heuristic (VVH) [2], proposed to generate more feasible initial states that may lead to better results. The PFIH is a simple-yet-efficient method to compute every route by comparing the cost of inserting a new customer into the existing route against that of starting a new route in each iteration until all customers are served. However, when the capacity of a vehicle is exceeded or the delivery time must be dragged behind the latest time specified by the new customer, a new route has to be started. Clearly, the PFIH can only quickly return a feasible solution without any guarantee for its global optimality. On the other hand, the VVH works by using *virtual* vehicles with unlimited capacity to hold the deliveries that are not currently serviced by any real vehicle so as to allow a more optimized delivery plan to be computed. In other words, the virtual vehicles are used as temporary buffers to which no problem constraints (such as time and capacity) can be applied. Furthermore, to ensure all the deliveries will be ultimately be performed by real vehicles, the cost incurred by a virtual vehicle for a customer visit is much higher than that incurred by a real vehicle. In this paper, we will focus on comparing the influence of the above initialization heuristics on the search meta-heuristics which we are interested to study. Section 3 will have a more detailed discussion on the initialization heuristics.

After the initial routes for the VRPs are generated, we can apply many possible heuristic methods to improve on the current solution until a better delivery plan with lower operational cost is obtained. The Tabu search (TS) [1] is a well-known meta-heuristic possibly used for such improvement, which has also been successfully applied to solve many other combinatorial optimization problems [3]. In solving the VRPs, given any initial route(s), there can be many possible *moves* [1] such as the 2-opt operation, which replaces any two links in a route with two different links to reduce the operational cost, to generate other possible route(s). The Tabu search works by firstly performing a neighborhood search on all possible moves, executing only non-Tabu moves which reduce the total operational cost, and ‘memorizing’ those recently performed moves with a Tabu list, of usually fixed length, to avoid cycling. In this way, TS promotes a diversified search from the current solution by continuously updating a short-term memory-like Tabu list until the predetermined stopping criterion is reached. In some cases, depending on the application-specific aspiration level criterion, the Tabu status of a move can be changed when it leads to a solution better than the current best. Alternatively, the guided local search (GLS) [2, 3] is another interesting meta-heuristic which has also achieved impressive results [2] in solving VRP-TWs effectively. Both TS and GLS are based on a local search operator to perform neighborhood search. However, each time after the local search is performed, the GLS uses a long-term memory-like penalty scheme to penalize all the undesirable *features* found in the current solution so as to avoid being trapped at the same local minima in the future exploration. The resulting penalty information, after multiplied by a regularization parameter λ, is incorporated into an augmented objective function to *guide* the local search to iteratively look for a better solution from the current search position until a predefined stopping condition such as the maximum number of iterations is reached. In addition to solving VRPs, GLS has been successfully applied to solve many difficult scheduling problems such as the traveling salesman problem [12].

Besides, there was some previous work [1, 2] proposed to combine the possible advantages of different meta-heuristics for solving constrained optimization problems or in particular VRPs more efficiently or effectively. Wah *et al.* [4] considered the integration of simulated annealing technique into the Discrete Lagrangian search framework, which was closely related to the GLS, as constrained simulated annealing (CSA) to solve a set of 10 constrained optimization problems (named G1 – G10) [4] with constraints and objective functions of various types. Later, the CSA was further improved with a genetic algorithm as CSAGA to achieve a better efficiency in solving the same set of constrained optimization problems. Specifically, Genfreau *et al.* [13] integrated simulated annealing and Tabu search to effectively solve VRPs. Interestingly, Becker *et al.* [1], after comparing the individual performance of GLS and TS on the Solomon's test cases, tried to combine both TS and GLS as the guided Tabu search (GTS) method which outperformed the original meta-heuristics with an average of 1.7% better in the solution quality on the "long haul" problems, that are problems involving deliveries of long distances. However, Becker *et al.*'s work suffered from three major drawbacks. First, their original aim, possibly only to obtain better experimental results, for combining both meta-heuristics was never explicitly stated. Second, the algorithm of the GTS was not clearly defined. There were only 3 statements to describe GTS very briefly. Third, no analytical model or clear explanation about the performance of GTS was given. On the other hand, we carefully propose in this paper a different integration of GLS and TS as GLS-TS with a clear objective to combine the *best* possible advantages of applying both short-term and long-term memory mechanism used in GLS and TS to solve VRP-TWs more effectively. More importantly, we provide a simple and easy-to-understand analytical model to under the behavior of the resulting GLS-TS when compared to the original meta-heuristics. Furthermore, similar to Wah *et al.*'s work [4], we propose the integration of simulated annealing technique into the GLS-TS search framework as GLS-TS-SA. However, it is worth noting that simulated annealing is specially used as a *monitor* in GLS-TS-SA to carefully determine, depending on the relative merit of each meta-heuristic in the previous search, which meta-heuristic to apply in each iteration. To demonstrate the effectiveness of our proposals in solving VRP-TWs, we implemented the prototypes of both GLS-TS and its variant GLS-TS-SA. Both prototypes have been shown to compare very favorably in the solution quality against the original GLS and TS methods, and the search hybrid GTS on the well-known Solomon's test cases. In addition, our proposed GLS-TS improved on one of the best published results, which were obtained from a number of advanced search techniques such as the Ant Colony search algorithm [9] and thus fairly difficult to break any of these records, in solving the Solomon's benchmarks on VRP-TWs. Up to our knowledge, this work represents the first attempt to systematically study the integration of TS into the GLS search framework.

This paper is organized as follows. Section 2 describes the general vehicle routing problems and VRP-TWs in detail. The various search initialization heuristics such as the PFIH, interesting meta-heuristics such as the Tabu Search and combined meta-heuristics such as the Guided Tabu Search (GTS) to effectively solve VRP-TWs will be given in Section 3. Section 4 details our proposals of adapting the generic MGA to solve the VRP-TWs more effectively. In Section 5, we evaluate the performance of our proposal against the original heuristic search methods and the related GTS on the widely used Solomon test cases. Lastly, we conclude our work in Section 5.

2. THE VEHICLE ROUTING PROBLEMS WITH TIME WINDOWS

A general vehicle routing problem (VRP) [1, 10], which is shown to be *NP*-complete [3], can be formally defined as follows. We are given a fixed N or infinite number of vehicles with limited capacity cv (measured in weight or volume) and M customers' requests, in which each request rq_j demands a delivery service for Q_j quantity of goods/service, to different locations. The distance, usually measured in term of minutes or hours required for travel, between any two possible delivery points is also provided, usually as a distance matrix. Then, our task is to optimize certain user-defined criteria subject to the following basic constraints:

1. for any customer request rq_j, it should only be served by one single vehicle only;
2. for each vehicle v, the sum Qv of quantity of goods to be delivered by vehicle v must be less than or equal to cv;

Besides the above basic constraints, in many real-life applications such as the supermarket delivery, each customer may request the items to be delivered in a given duration (time window) of a day. Thus, a vehicle routing problem with time windows (VRP-TW) has an additional time-window constraint to be imposed for each delivery as follows.

3. when a time window with the earliest time E_j and the latest time L_j is specified for each delivery in a VRP-TW, the arrival time T_{vj} of vehicle v to serve customer request rq_j must lie within the specified duration, that is $E_j \leq T_{vj} \leq L_j$.

One of the most common objectives for minimization in VRPs or VRP-TWs is *TV x TD*, where *TV* is the number of vehicles used, and *TD* is the total distance travelled by all vehicles. Clearly, there can be many variants of VRPs or VRP-TWs with different objectives such as the total travel time of all the vehicles or the total waiting time of all the customers for optimization in many real-life applications. In the next section, we are going to examine two common heuristics, namely the push-forward insertion heuristic and the virtual vehicle heuristic, useful to obtain an initial and feasible solution in solving the VRPs or VRP-TWs. Moreover, we will review two interesting meta-heuristics to solve the difficult VRP-TWs optimally.

3. USEFUL HEURISTICS AND META-HEURISTICS FOR VEHICLE ROUTING

Since the search space for all the possible (feasible or slightly infeasible) routes in VRPs or VRP-TWs can be fairly large even for instances involving 100 customers [1, 10] or more [9], and the time-window constraints in the VRP-TWs can be difficult to satisfy, the careful choice of a suitable heuristic to return only feasible, and possibly more optimal, solutions can be important for further optimization. The push-forward insertion heuristic [6] and virtual vehicle heuristic [2] are two useful heuristics for search initialization in solving difficult VRPs. In addition, we will examine in this section two well-known meta-heuristics, namely the guided local search (GLS) and Tabu search (TS), which are based on completely different memory-like control mechanism to restrict the local search operator in continuously optimizing the current solution, after inputted from the heuristic initialization method, until an optimal solution is obtained to solve the VRP-TWs successfully. TS uses a short-term memory-like Tabu list to avoid cycles in search. On the other hand, GLS uses a long-term memory-like penalty scheme to "memorize" all the undesirable features occurred in the previously visited local minima. In Section 4, we will examine various proposals to integrate TS into the GLS framework to solve VRP-TWs more effectively.

3.1 Useful Search Initialization Heuristics

The push-forward insertion heuristic (PFIH) [10], introduced by Solomon in 1985, is an efficient method to compute a feasible solution for any VRP by assuming an infinite number of vehicles. The PFIH starts a new route by selecting an initial customer, usually farthest from the delivery depot, and then iteratively insert any unassigned customer into the current route until the capacity of the current vehicle is exceeded or the waiting time for any newly added customer will exceed its associated duration constraint. At this moment, a new route will be created. And this process repeats until all customers are served. Basically, the algorithm for PFIH can be summarized as follows.

1. Begin an empty route r_0 starting from the depot; $i := 0;$
2. Among all unassigned customers, select the customer farthest from the depot (in case there is a tie, break the tie randomly) and insert into the current route r_i;
3. **If** all customers are routed, **then goto** step 6. **else** :

   If the capacity *cv* of the vehicle *v* involved in the current route r_i is exceeded, **then goto** step 5. **else** :

   foreach unassigned customer, find the best position for insertion in r_i. Compute the cost of starting a new route against that for the best position found.
4. Pick the customer with the greatest cost difference and insert it into r_i. Update the capacity *cv* of the vehicle *v* involved. **goto** step 3.
5. Start a new route r_{i+1} starting from the depot; $i := i+1;$ **goto** step 2.
6. Return the current solution.

Clearly when handling VRP-TWs, we have to adapt the last **else**-part of step 3 as follows.

foreach unassigned customer, find the best position for insertion in r_i **without violating any specified time-window.** Compute the cost of starting a new route against that for the best position found.

Moreover, we have to add the following conditional statement to the beginning of step 4 as follows.

If there exists no feasible position for insertion into the current route r_i, **then goto** step 5. **else** :

In general, PFIH can be integrated into the search framework of many heuristic search algorithms [3, 6] to efficiently handle the VRP-TWs. It should be noted that the initial solution returned by PFIH represents only a feasible and usually not optimal solution for further optimization by the different meta-heuristics.

On the other hand, the virtual vehicle heuristic (VVH) [2], with the availability of application-specific knowledge such as the number of vehicles used in the best published result for a particular VRP-TW, can often return both feasible and fairly optimal initial solutions for further processing. Using this approach, a virtual vehicle is used to hold the unassigned customers. The virtual vehicle is different from a real vehicle in 3 respects. First, the virtual vehicle cannot visit 2 or more customers en-route, but have to visit each customer in turn after it makes a return trip to the depot. Second, it is not constrained by any domains. Third, the cost of a customer visit for virtual vehicle is higher than a real vehicle. This is to ensure that all customers will ultimately be assigned to a real vehicle. The VVH method for search initialization can be specified formally as follows.

Given

- an objective function O (often measured in term of the total distance travelled by all vehicles)
- the cost of delivery by a virtual vehicle = $\alpha_1 (D_{di} + D_{id}) + \alpha_2$, where
    - o D_{ij} is the distance between customers c_i and c_j with d as the depot,
    - o α_1 and α_2 are parameters set to increase the cost of a visit by the virtual vehicle,

the VVH algorithm proceeds as follow.

1. The virtual vehicle services all the customers' requests.
2. For each customer-vehicle combination in the current solution, use 4 heuristic operators, namely the *2-opt*, *relocate*, *exchange* and *cross*, to try to improve the solution quality by firstly considering only legal moves that do not violate any constraint, and then executing the legal move which decreases the objective function most.
3. **If** no more customer-vehicle combination can be improved to reduce the solution quality, **goto** step 5.
4. **goto** step 2.
5. **return** the current solution.

The 4 heuristic operators used in step 2 try to improve the current solution by moving customers' location in the same or different route. These operators can be classified as intra-route or inter-route. Figure 1 shows the 4 heuristic operators.

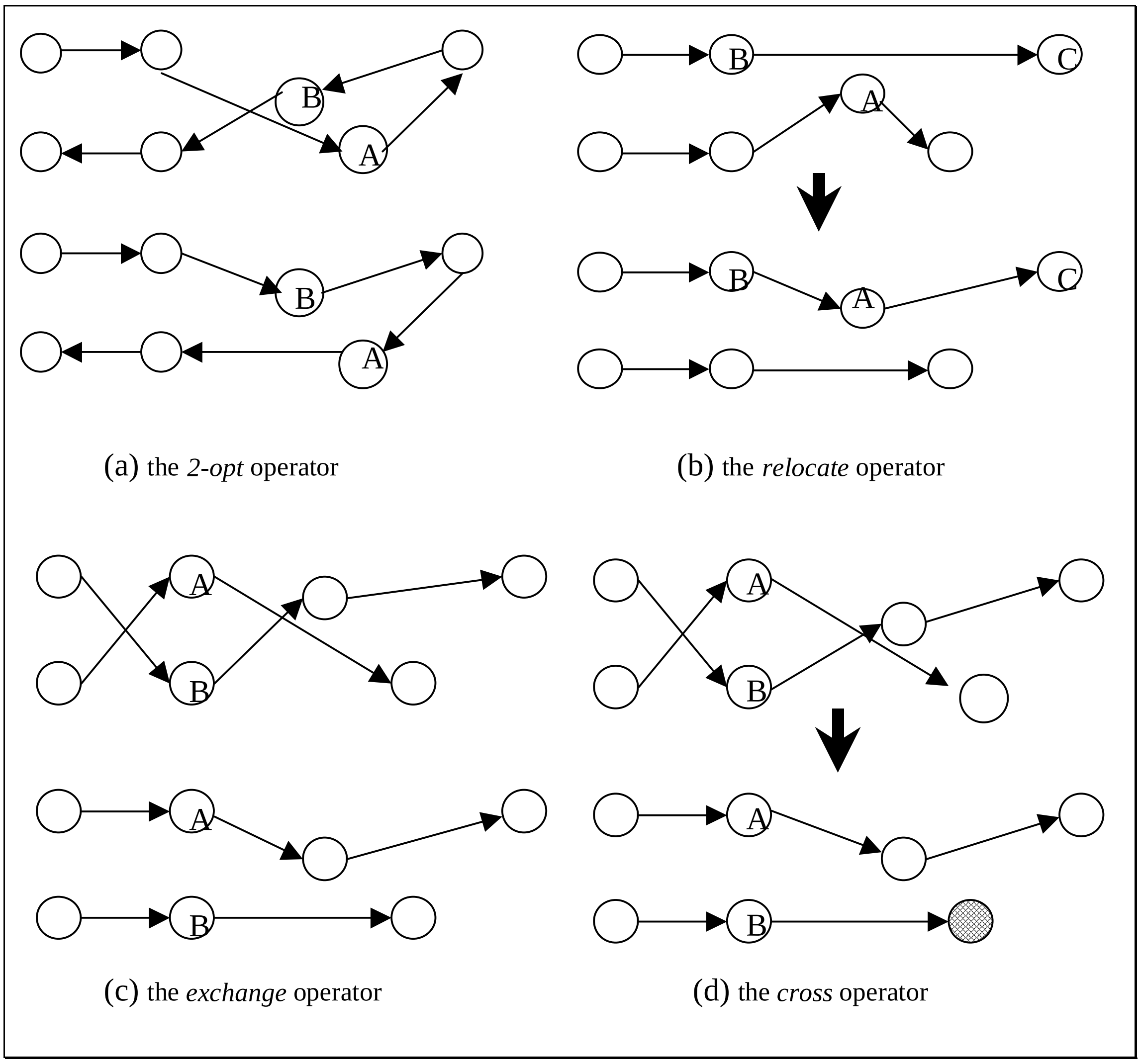

Figure 1: The 4 Heuristic Operators for Improving Routes in Solving VRPs

The *2-opt* [14] is an intra-route operator which reverses a section of a route by deleting 2 arcs, and replacing them with any 2 arcs to reform the route. On the other hand, the *relocate, exchange* and *cross* [15] are inter-route operators. The *relocate* operator moves a visit from its position in one route to another position in either the same or a different route. *Exchange* swaps 2 visits from either the same or different route while *cross* swaps the end portions of 2 routes.

Besides, from the empirical results obtained in [1], good settings of parameters to calculate the cost of customer visits performed by virtual vehicles are $\alpha_1 = 1.025$ and $\alpha_2 = 0.0005\Delta$, where Δ is the distance between the two most remote sites in the problem. Also, to achieve impressive results in solving the VRP with VVH, we should set the maximum number of vehicles allowed. This observation is consistent with some previous report [2] in which the maximum number of vehicles allowed for VVH should be v_p, v_p+1 or v_p+2, where v_p is the number of vehicles used for the best published results.

Nevertheless, the stringent demand for this extra parameter v_p to improve the quality of the initial solution produced will make VVH a more application-specific heuristic, thus less favorable for any general real-life application.

3.2 Guided Local Search

The guided local search (GLS) uses a meta-heuristic based on penalties to continuously modify the penalty terms associated with an augmented objective function to escape from local minima for efficiently solving a wide range of constrained optimization problems [3]. Based the original objective function *O(S)* for optimization, GLS defines an augmented objective function as: $O'(S) = O(S) + \lambda \sum_{i \in F} f_i(S).p_i.c_i$ where λ is a penalty factor representing the relative importance of all the penalties, $f_i(S)$ is an indicator function returning *1* when the feature *i* in the set *F* of features under consideration appears in the current solution *S*, and *0* otherwise; p_i is the number of times which the feature *i* is penalized, and c_i represents the cost of feature *i*. With all these input parameters properly set for a specific application, the GLS algorithm [2, 3] works as follows.

```
P := $\vec{0}$ ;                    // initialise the penalty vector to be zeros
S := InitialSolution();    // S is the current solution
S* := LocalSearch(S);    // Make sure the best solution S* found starts from a local minimum
while not StoppingCondition() do
  f := ChoosePenaltyFeatures(S, p);-----------------(1)
  foreach x in f do $p_x := p_{x+1}$;------------------------(2)
  S := LocalSearch(S);
  If O(S) < O(S*) then S*:= S;
return S*
```

Figure 2: The Pseudo-codes of Guided Local Search

The function *InitialSolution()* returns the initial solution to solve the VRPs by PFIH or VVH as previously described. *LocalSearch(S)* performs a local search on the current solution *S* till no more improvements can be made. The local search is executed based on the 4 heuristics operators described in the previous section. Similar to the VVH, the “best accept” approach is used in which the move which reduces the value returned by the objective function *O* by the largest amount will be executed. *StoppingCondition()* checks whether the predetermined stopping criterion to terminate the search is satisfied in each iteration. In many real-life applications, the stopping criterion can be defined in terms of the maximum number of iterations or improving moves. *ChoosePenaltyFeatures(S, p)* takes the current solution *S* and the penalty vector *p*, and then returns the set of features f_i to be penalised. For general applications, GLS chooses all features in S for which the utility $c_i/(p_i+1)$ is largest amongst the features in S. The statements marked with (1) and (2) are important for the subsequent discussion of search hybrids based on GLS.

It is interesting to note that the penalty vector *p* serves as a long-term memory to memorize all the undesirable features appeared in some previously visited local minima. In solving VRPs, GLS penalizes the longest arc in the current solution, weighted by the number of times the features that have been already penalized. Furthermore, there were some empirical observations [2, 3] stating that the value of λ might greatly affect the performance of GLS and the quality of the solution returned. As for solving VRP-TWs, it was discovered in [2] that GLS worked well when λ was ranged from *0.1* to *0.3*, giving the best result at *0.2*.

3.3 Tabu Search

Tabu search is an interesting meta-heuristic which aims to model the human memory process through the use of a Tabu list, usually of a fixed length, to prohibit most recent *moves* possibly leading to some previously visited local optima. In solving VRP-TWs [1, 10], moves can be defined in term of the addition or deletion of arcs associated with individual nodes (as customers) into or out of all computed routes. Besides the Tabu list, an *aspiration* criterion is used to change the Tabu status of a move so that the move in the Tabu list can still be accepted when it leads to a solution better than the current best solution. In general, the performance of a Tabu search algorithm depends largely on effectiveness of the local search operators, the length of the Tabu list and the settings used for the aspiration criterion. Nevertheless, Tabu search has been extensively used to solve many difficult combinatorial problems with good results [1].

Figure 3 gives the Tabu search algorithm as described in [1, 2].

```
S := InitialSolution();    // S is the current solution
S := LocalSearch(S);       // Make sure we start with a local minimum
S*:= S;                    // S* represents the best solution found
```

```
while not StoppingCondition() do
  moves := RankMoves(S);  ----------------(*)
  moved := false;
  while not moved do
    m := head(moves);
    if IsNotTabu(m) then
      Perform(m);
      moved := true;
     InsertTabuList(m);
    If O(S) < O(S*) then S*:= S;
return S*
```

Figure 3: The Pseudo-codes of Tabu Search

The search starts with an initial solution *S* provided by *InitialSolution()*. Then, *LocalSearch(S)* uses the 4 heuristic intra-route and inter-route operators to perform local search so as to iteratively improve the current solution *S* until it reaches a local minimum which is then assigned to the current best solution *S**. Before a predefined stopping condition is met, *RankMove(S)* firstly returns a ranked list of all possible moves, ordered by decreasing cost difference, from the current solution *S*. While there is no move performed, the first non-Tabu move *m*, as determined by *IsNotTabu(m)*, from the list *moves* will be executed by *Perform(m)*. Then, the flag *moved* will be reset to exit the inner loop before *InsertTabuList(m)* adds the most recently performed move *m* into the Tabu list. Lastly, the current best solution found may be updated, for which *O(S)* is the objective function returning the quality of the current solution *S*. The whole process is repeated until the stopping criterion is reached. For details, refer to [1]. It should be noted that, the above discussion mainly focuses on the use of Tabu list as a simple short-term memory to diversify the search for avoiding local optima. However, it may also be possible to include some long-term memory structure for explicit intensification and diversification phrases as in some sophisticated Tabu search algorithms. In fact, in the guided Tabu search algorithm (GTS) [1] or our proposal of integration to be discussed in Section 4, the penalty-based learning scheme [2, 3] can also be regarded as one possible type of long-term memory mechanism for avoiding the already visited local optima. The statement marked with (*) is used for the subsequent discussion of GTS.

4. COMBINING META-HEURISTICS TO EFFECTIVELY SOLVE VRP-TWS

As discussed in the previous section, Tabu search employs a Tabu list as the short-term memory to avoid cycles whereas the guided local search (GLS) uses a penalty vector as the long-term memory to memorize all undesirable features occurred in the previously visited local minima. Therefore, it can be advantageous to combine both meta-heuristics to complement each other during the search process. In the following paragraphs, we are going to discuss the various proposals of combining both TS and GLS to solve VRP-TWs effectively. First, we review the guide Tabu search (GTS) as an ad hoc integration of TS and GLS proposed by Backer *et al.* [1]. Second, we consider our careful proposal of combining TS and GLS as the GLS-TS algorithm with an easy-to-understand and interesting analytical model to differentiate its expected search behavior from that of GTS. Lastly, we will describe an integration of simulated annealing technique into the GLS-TS framework as a variant of GLS-TS to demonstrate the numerous opportunities opened up by our proposal of integration.

4.1 Guided Tabu Search

Originally, Backer *et al.* [1] aimed at comparing the performance of GLS and TS in solving the Solomon's test cases of VRP-TWs. Later in the experimentation stage, after observing the simplicity and impressive results produced by GLS, Backer *et al.* proposed a fairly ad hoc integration of the GLS penalty scheme into the Tabu search algorithm as the guide Tabu search (GTS) to possibly improve on the performance of the original search methods. It should be noted that GTS was only briefly mentioned in [1] without showing any explicit pseudo-code. To facilitate our subsequent discussion, we explicitly define the GTS algorithm as follows.

```
S := InitialSolution();    // S is the current solution
S := LocalSearch(S);    // Make sure we start with a local minimum
S*:= S;                        // S* represents the best solution found
while not StoppingCondition() do
  f := ChoosePenaltyFeatures(S, p); // to select and then penalize the concerned features
  foreach x in f do $p_x := p_{x+}1$;        // so as to avoid re-visiting the same local minima
  moves := RankMoves(S);
  moved := false;
  while not moved do
```

```
    m := head(moves);
    if IsNotTabu(m) then
      Perform(m);
      moved := true;
     InsertTabuList(m);
    If O(S) < O(S*) then S*:= S;
return S*
```

Figure 4: The Pseudo-codes of Guided Tabu Search

Figure 4 gives the pseudo-codes of the GTS algorithm. Syntactically, the GTS algorithm is obviously obtained by inserting lines (1) and (2) of the GLS algorithm as shown in Figure 2 before the line (*) of the TS algorithm as described in Figure 3. The semantic meaning of this addition is revealed in Backer *et al.*'s original work [1] which clearly stated that after each move, a GLS-type procedure was called to update the weight in the cost matrix (used in the penalty scheme). However, we consider this frequent invocation of the GLS-type procedure may collect "immature and hazardous" penalty information after every single *diversifying move* of TS. The collected penalty information can be "hazardous" in two senses. First, it can mislead the current search direction by biasing towards some unimportant features found in the previous solutions. Second, the vast amount of possibly useless penalty information generated may simply cause information or memory overflow during program execution. Besides, it may slow down the performance of GTS. In the next subsection, we will provide an interesting and simple model to carefully consider the possible influence of this frequent penalty mechanism on the search behavior of GTS. After all, from Backer et al.'s empirical experience [1], GTS showed some benefits by outperforming the TS and GLS with an average of *1.7*% on the long-haul problems, including C2, R2 and RC2, of the Solomon's test cases. However, for the remaining problem classes, GTS did not show any convincing result when compared to GLS.

4.2 Our Proposed Guided Local Search - Tabu Search (GLS-TS)

As previously discussed, the guided Tabu search (GTS) did not seem to be an attractive proposal of integrating GLS and TS since the short-term memory nature of the TS may *not be able to effectively use* the long-term memory-like penalty information *frequently* provided by the GLS scheme to diversify the search from its current status. In other words, the GLS scheme may penalize some features which will be totally irrelevant to the current state of TS. Therefore, we carefully propose an alternative proposal of integration: instead of performing a single non-Tabu move for every update of the cost matrix used in the penalty scheme, we propose to update the penalties only when the TS reaches a *stable state*. In other words, the penalties are updated only when all the non-Tabu moves cannot further improve the current solution any more. For clarity of presentation, the GLS-TS algorithm is formally defined as follows.

```
P := $\vec{0}$;                // initialise the penalty vector to be zeros
S := InitialSolution();    // S is the current solution
S* := LocalSearch(S);    // Make sure the best solution S* found starts from a local minimum
while not StoppingCondition() do
  f := ChoosePenaltyFeatures(S, p);
  foreach x in f do $p_x := p_{x+1}$;
  improved := true;    ---------------------------------------------(**)
  while improved do // Start of TS Local Search
       prevO := O(S);
       moves := RankMoves(S);
       moved := false;
       while not  moved do
               m := head(moves);
               if IsNotTabu(m) then
                  Perform(m);
                  moved := true;
                  InsertTabuList(m);
       if O(S) < O(S*) then
               S*:=S;
       If prevO <= O(S) then
               improved := false; // End of TS Local Search -------------(***)
return S*
```

Figure 5: The Pseudo-codes of Guided Local Search – Tabu Search (GLS-TS)

Figure 5 shows the GLS-TS algorithm. The definitions for the function *InitialSolution(), LocalSearch(), StoppingCondition(), RankMoves(S), head(moves), IsNotTabu(m), InsertTabuList(m)* and *ChoosePenaltyFeatures(S,p)* follow those previously described for the GLS and TS algorithms. The boolean *improved* is used to detect whether the TS has reached a stable state where no improvement on the current solution can be made by any non-Tabu move. Although our proposed GLS-TS may be misleadingly seen as a slight modification from the GTS syntactically, the semantic implication is in fact very significant. By appropriately integrating the penalty information of GLS only when the Tabu search is stable, our proposed GLS-TS algorithm is trying to maximize the best advantages of information gained from GLS to improve the original Tabu search. This competitive advantage of GLS-TS when compared to GTS is illustrated clearly in the following simplified Markov chain diagrams to analyse their different search behaviour in successfully finding a solution for general and solvable VRPs.

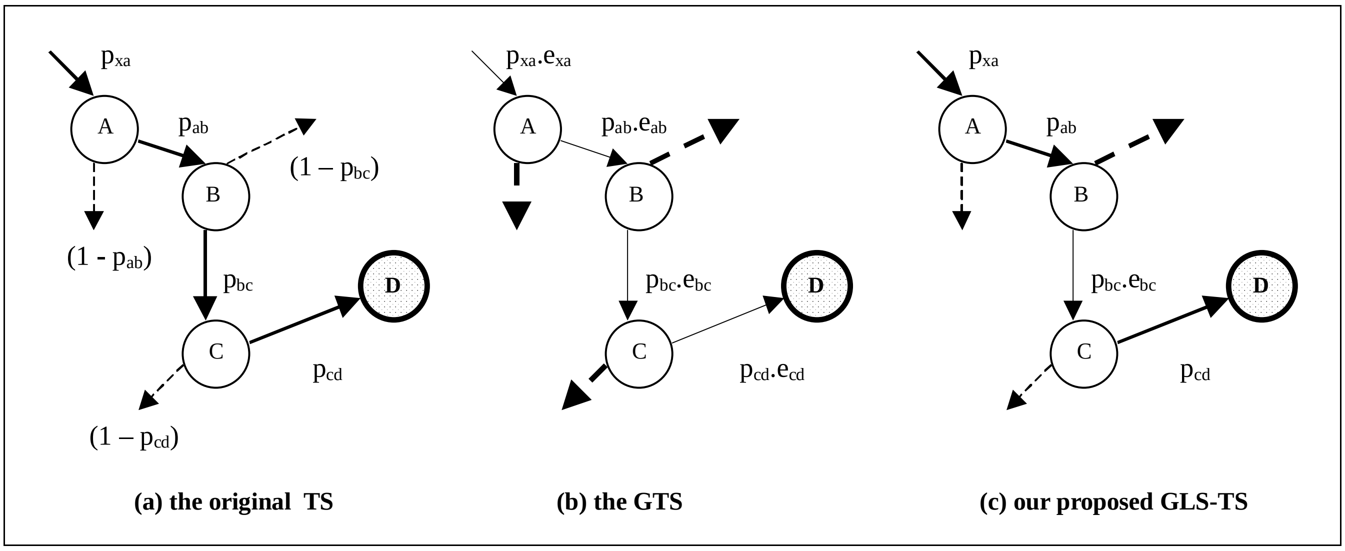

Figure 6: The simplified Markov models for the original TS, GTS and GLS-TS algorithms

Figure 6 shows the simplified Markov models for the original TS, GTS and GLS-TS algorithms to successfully find a solution for a general and solvable VRP. Each circle in the diagram denotes a (search) state. Each arrow, usually attached with the corresponding probability, represents a state transition from one state to another. The arrows with dotted lines represents the transition to any non-*X* state, where *X* is the targeted state for the current state. The thickness of the (solid- or dotted-) line of any arrow is proportional to the likeness/probability for that transition to occur. For example, the solid-line arrow from state A to state B in Figure 6(a) is attached with the probability p_{ab} for such transition to occur while the dotted-line arrow from state A to any non-B state is attached with the probability (1 - p_{ab}). Presumably, the probability p_{ab} is higher than (1 - p_{ab}) since the solid-line is clearly thicker than the corresponding dotted-line. Suppose we have identified state A, B, C and D as the typical last four states to reach a solution for a particular solvable VRP out of a careful empirical observation on all the successful runs, with state B as the last 'stable' state representing a local minimum before the final 'solution'/terminal state D. Figure 6(a) shows the state transitions with their attached probabilities for these last four states of the original TS as the basis for subsequent comparisons against the GTS and our proposed GLS-TS. For instance, the probability of reaching from state A successfully to state D equals to $p_{ab} . p_{bc.} p_{cd}$ + (1 - p_{ab}). 1/ (N - 1) + $p_{ab} .(1- p_{bc})$. 1/ (N - 1) by assuming the independence of events for a simple analysis, where N is the total number of possible states, and the second term (1 - p_{ab}). 1/ (N - 1) gives the probability of reaching from state A to non-B states which includes state D as one out of the (N - 1) remaining states. For simplicity of analysis, we also assume the probability of reaching from any previous state to state A is greater than 0.5 so that the probability of successfully reaching state D largely depends on the decisions made in the last four states. Since the three probabilities p_{ab} , p_{bc} and p_{cd} normally range from 0 to 1, and N is at least of the order of *1,000* or more, the first term $p_{ab} . p_{bc.} p_{cd}$ is usually the most important quantity to consider in determining the probability of reaching from state A to state D. To illustrate, let p_{ab} , p_{bc} and p_{cd} be *0.4, 0.5* and *0.6* respectively, and N be *1,000,* the probability of reaching from state A to state D equals to *0.4 x 0.5 x 0.6 + (1 – 0.4) x 1/999 + 0.4 x (1 – 0.5) x 1/999 = 0.12 + 0.0006 + 0.0002 = 0.1208* in which the first term is clearly the most determining factor. However in the GTS, by including the GLS penalty mechanism in every step of the TS, we can assume the probability of a system transition (A $\rightarrow$ B) being affected by the GLS penalties as g_{ab}, and conversely the probability of not being affected as $\mathrm{e}_{\mathrm{ab}} = 1 - g_{\mathrm{ab}}$. Similarly, all of these *e's* and *g's* will be in the range of 0 to 1. Moreover, the *g's* should be progressively increasing from the initial to the last state since there can be more features to be penalized or higher penalty resulted after each iteration which may affect the state transition of the original TS more drastically. Overall speaking, applying the GLS penalty scheme in each TS step will substantially lower the first important term of the probability of successfully reaching from state A to state D, at least by the order of 10^{-3}, to $p_{ab} . p_{bc.} p_{cd.} e_{ab} . e_{bc.} e_{cd}$ which cannot be offset by the relatively small opposite increase in the second and third terms of the specific probability due to the very small increase in the probability of reaching the non-B and non-C states from state A and B. This clearly shows the possible pitfalls of the GTS proposal which may simply increase the chance of diverting the current search to some unintentional states, that are any states other than the solution state D in the non-B and non-C states. On the other hand, by intelligently maintaining the original characteristics of the TS and only affect the TS by the GLS penalty mechanism in *stable states* like state B in our proposed GLS-TS as shown Figure 6(c), the original strength as reflected in the first important term of the probability of reaching from state A to D is largely unaffected by multiplying only one e_{bc} as p_{ab} . $p_{bc}.e_{bc}.$ p_{cd}.. Meanwhile, the probability of reaching non-C states at the stable state B is appropriately increased to open up a *risky but possibly favorite* opportunity to try to directly reach the solution state D. Also, it is worthy noting that since the system already arrives at the stable state B, there is a fair chance of re-entering the same stable state B even after it fails to reach the solution state D. In other words, we carefully inject some intended *noise* into the original search model of TS (as shown in Figure 6(a)) to increase the chance of directly reaching the solution state(s) only at stable states while not drastically scarifying the original strength, that is a higher probability of following the normal transitions from A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D, of TS in our proposed GLS-TS. After all, Section 5 will give the experimental results of the original TS, the GTS and our proposed GLS-TS on the well-known Solomon's benchmarks [10] to justify our discussion here.

4.3 Yet Another Variant of GLS-TS

Both GTS and our proposed GLS-TS are basically *fixed* strategies to invoke GLS-type penalty scheme within the TS framework, it will be interesting to have other variants of GLS-TS which can flexibly invoke different search mechanisms depending on the relative merit of the involved mechanisms during the dynamic search process. Basically, we need a kind of measures and a *monitor* as the feedback-and-control mechanism of this flexible invocation scheme. Similar to Wah *et al.*'s work [4], we propose to integrate the simulated annealing technique as a monitor to supervise the performance of different search mechanisms and decide which mechanism to invoke in each stage of the search. This forms the basis of our proposed GLS-TS-SA algorithm as a variant of GLS-TS integrated with the simulated annealing technique to decide whether to invoke the common local search method used in GLS or the unique Tabu search method during the search process. For testing, we simply used the number of accumulative *improving moves* made by each search method as a score to measure their relative merit. Obviously, there can be more sophisticated measures defined to carefully evaluate their performance. More importantly, our proposal of integration discussed here opens up many potentially interesting directions for future investigation.

```
$P := \vec{0}$;                  // initialise the penalty vector to be zeros
$S$ := InitialSolution();    // $S$ is the current solution
$S^*$ := LocalSearch(S);    // Make sure the best solution $S^*$ found starts from a local minimum
```

```
i:=0                    // initialise i
impM:=0                 // initialise the counter impM
while not StoppingCondition() do
  f := ChoosePenaltyFeatures(S, p);
  foreach x in f do $p_x := p_{x+1}$;
  temp:= MAX_TEMP * $e^{-(impM)/i}$
  $P_{calc} := 1 / (1 + e^{(-d/temp)})$
  $P_{gen} := Random(0,1)$
  if $P_{gen} <= P_{calc}$ then
    S:=Tabu-LocalSearch(S)
    i:=i+1
    if O(S) < O(S*) then
        impM:=impM+1
  else
    S:=LocalSearch(S)
  if O(S) < O(S*) then
        S*:=S;
return S*
```

Figure 7: The Pseudo-codes of GLS–Tabu Search integrated with Simulated Annealing (GLS-TS-SA)

Figure 7 gives the pseudo-codes of the GLS-TA integrated with simulated annealing. *InitialSolution()*, *LocalSearch()*, *StoppingCondition()* and *ChoosePenaltyFeatures(S,p)* are as described in the GLS-TS algorithm in Figure 5. *Random(0,1)* generates a random real number between *0* and *1*. The function *Tabu-LocalSearch()* basically performs a Tabu search until no more improvements for the objective function is achieved. The variable *i* counts number of times the variant GLS-TS-SA performs Tabu search while the counter *impM* measures the number of accumulative improving moves affected by Tabu search. The variable *d* represents the *decrease* in the total distance traveled, which is either *0* or -*d* for any increase. The parameter *MAX_TEMP* was empirically determined at 0.3. In addition, the formulation for *temp* and P_{calc} basically follows the standard logistic function described in [11]. Besides our proposed GLS-TS-SA, we can clearly have many other possible search hybrids integrated with simulated annealing such as an adaptive strategy which can flexibly invoke the GLS penalty scheme within the TS framework depending on its performance. Some of these proposals form our on-going work.

5. Experimental Results

We used the well-known Solomon's test cases to evaluate the performance of our proposals against the original Tabu search [13] and Backer's *et al.*'s variant - the Guided Tabu search [1]. There are 56 instances of delivery problems in the Solomon's test set, each with 100 customers, which can be categorized into 6 classes: C1, C2, R1, R2, RC1 and RC2. The first letter(s) of the class name denotes the type of customer distribution. For instance, 'C' represents the clustered customers, 'R' denotes the randomly distributed customers, and 'RC' involves a mix of customers of both types. After that, the number in the class name encodes the types of vehicle capacity and service duration. '1' refers to the test cases with small vehicle capacity and short service duration while '2' stands for those with large vehicle capacity and long service duration. For all the 6 classes, the demand for each customer within the same class is the same. Refer to **Appendix A** for the full description of each class.

The original guided local search (GLS) and Tabu search algorithms and their hybrids such as the guide Tabu search (GTS), our proposed guide local search – Tabu search (GLS-TS) and its variant GLS-TS integrated with simulated annealing (GLS-TS-SA) are all implemented in C/C++ and compiled with g++ (the GNU C++ version egcs-2.91.66 with optimised compilation option), running on a machine with Intel Pentium III (450Mhz) processor and 256 megabytes of random-access memory (RAM). The operating system is Linux RedHat Version 6.1 (Cartman) Kernel 2.2.12-20 on i686. All the implementations are in fact modified from the original PFIH+MGA [6]. Most of the data structures are implemented as global array for efficient handling. Moreover, the *4* heuristic operators are modified so that instead of performing an actual move, the effects of performing the move are calculated. This is achieved by considering the arcs added and deleted for a particular move. For the constraint checking, the new move is only checked against those routes that it will affect. Hence, the execution time of the program is greatly reduced by a factor of 10 or more.

The lemma values used are 0.2 for GLS, and 0.15 for GTS, GLS-TS and GLS-TS-SA. For each approach, the solver is executed 10 times with the average and the best results recorded. For all the approaches, the resource limit or stopping criterion is set at 1000 iterations. For the following results, we use "PFIH+" to represent those search methods using the push-forward insertion heuristic for initialisation while “VVH+” is used to represent those methods using the virtual vehicle heuristic as discussed in Section 3.

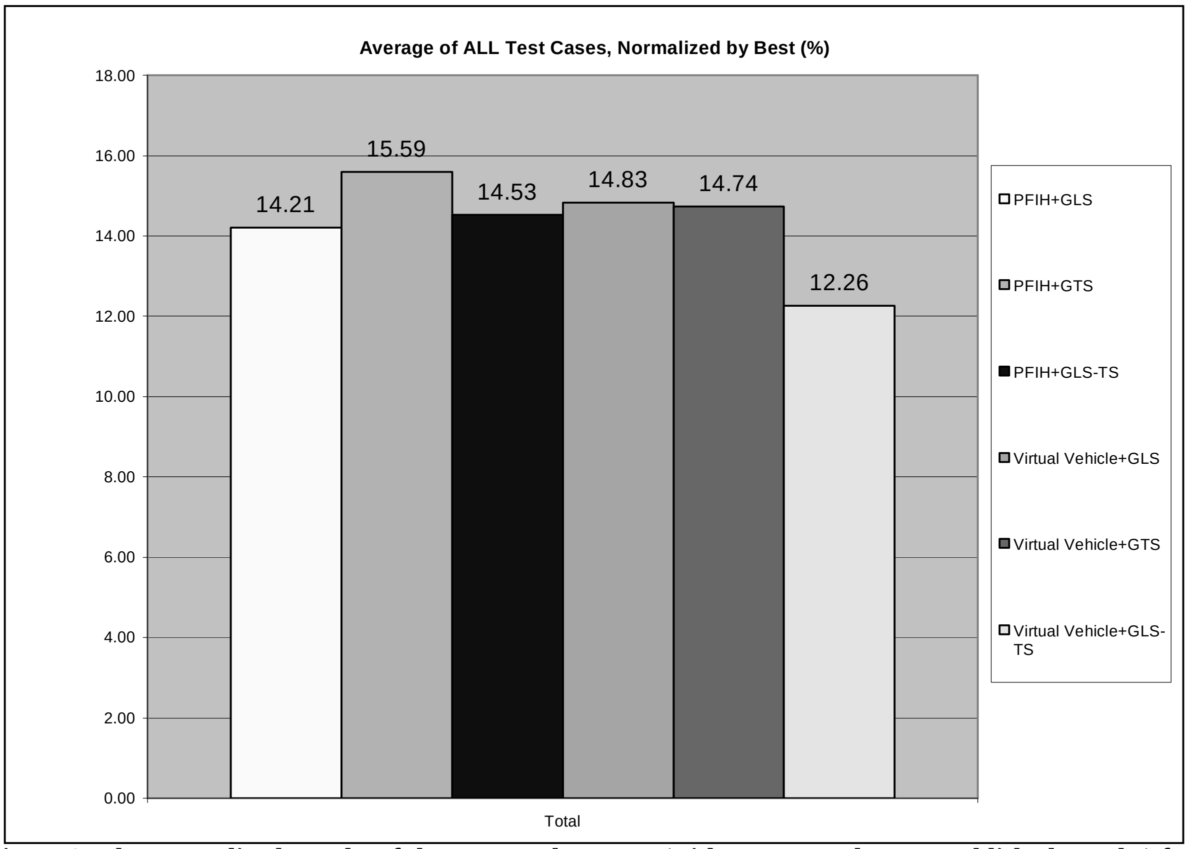

Figure 8: The Normalized Results of the Averaged *TDxTV* (with respect to the Best Published Results) for All the Test Cases of the GLS, GTS and GLS-TS Methods Initialised with PFIH or VVH

Figure 8 shows the performance of GLS-TS against GLS and GTS after normalized by the best published results from 6 sources [16]. All the results are measured in term of the averages of the objective values *TV x TD* , as discussed in Section 2, for all the 56 test cases in the Solomon’s benchmark [10]. From these experimental results, GLS-TS outperforms against both GTS regardless of the initial search. For instance, GLS-TS remains competitive when comparing the PFIH+GLS with PFIH+GLS-TS. Furthermore, VVH+GLS-TS is the most effective search methods among the 6 search methods we compare here. Most importantly, VVH+GLS-TS has improved on 1 test case, R205, giving: *TV=997.385*, *TD=3* and *TV x TD=2992.16* which is *0.13%* better than the best result: *TV=998.72*, *TD=3* and *TV x TD=2996.16* as reported in [2]. It should be noted that it is very difficult to break any of these best-published records since they are in fact the best results among the best solvers ever compared. This supports our analysis discussed in

Section 4.2 that updating the penalty vector only at local minima rather than after each Tabu move may significantly improve the performance of the original search methods. In general, these results demonstrate that our proposal, GLS-TS, has competitive advantages over its parents GLS and TS in solving VRP-TWs effectively.

Besides, VVH+* searches outperform the corresponding PFIH+* searches in 2 out of 3 test cases on the average. This improvement can be explained in term of the additional domain knowledge gained from the parameter v_p which is the number of vehicles used in the best-known results. Nevertheless, VVH+* searches fail to find solutions in some specific test cases probably due to the bound $v_p + 1$ or $v_p + 2$ imposed on the initial search, which may hinder the applicability of this initialisation heuristic in many real-life applications. After all, our proposed GLS-TS still proves to be effective with both PFIH and VVH.

Approaches	**TVxTD**	**Averaged differences of all test case from BEST (%)**
BEST	453997.1	0
PFIH+GLS	503206.1	14.21
PFIH+GTS	507321.2	15.59
PFIH+GLS-TS	504924.3	14.53
PFIH+GLS-TS-SA	556400.3	16.10
VVH+GLS *	486590.1	14.83
VVH+GTS*	455471.0	14.74
VVH+GLS-TS*	480169.8	12.26
VVH+GLS-TS-SA	497175.1	12.96

Table 1: A Comparison of the Performance of All the Solvers initialised with the PFIH or VVH Against the Best Results on the Solomon's Test Cases

Table 1 shows the performance of different solvers initialised with PFIH or VVH in solving the Solomon's test set of VRP-TWs. The performance of the solvers is measured in terms of two quantities: the sum of *TV x TD* and the averaged deviations from the best published results in percentage for all the test cases. In general, the VVH+* methods give better results for the sum of *TV x TD* when compared to the corresponding PFIH+* methods probably due to the advantage of the added domain knowledge. Undoubtedly, in term of the averaged deviations from the best results, VVH+GLS-TS with the lowest *12.26%* is the winner among the GLS- or TS-based solvers. Again, this demonstrated the competitive advantage of invoking the GLS-type penalty scheme only at stable states of TS as illustrated in the analytical model in Section 4.2. On the other hand, VVH-GTS scored the lowest sum of *TV x TD* among the compared solvers probably due to the probabilistic nature of the non-targeted states, that are the non-*X* states as discussed in Section 4.2, which *accidentally* lead the search to the solution states in some specific instances. The actual reason(s) for the overall improvement on the sum of *TV x TD* in this particular case require a more detailed investigation. Nevertheless, this result clearly demonstrates the effectiveness of combining both GLS and TS meta-heuristics in solving real-life VRP-TWs.

Furthermore, the sum of *TV x TD* for the VVH+GLS-TS-SA is the largest among the VVH+* solvers while the PFIH-GLS-TS-SA shows the largest averaged deviations from the best results among all the comparing solvers, clearly demonstrating the difficulty in controlling the local search or Tabu search with the integrated simulated annealing technique. We will try to explain this difficulty in term of the probabilistic model we have discussed in Section 4.2. Roughly, integrating the simulated annealing technique to decide which search method to use in each iteration is similar to adding a pre-requisite state before each original state in the state transitions of the TS algorithm as shown in Figure 6(a). Each pre-requisite state has two out-going arrows as two possible transitions – one leads to the original TS state while another leads to the additional local search state as possibly occurred in the GLS algorithm. The overall effect is the main component ($p_{ab} . p_{bc}. p_{cd}$) of the probability of successfully reaching the solution state is significantly lowered by a factor of h^n where h, ranging from *0 to 1,* is the probability of still being the most rewarding search method (with respect to the normal local search method), and n is the number of TS states involved. Moreover, the newly added pre-requisite and local-search states into the original state transition diagram of the TS simply means the resulting GLS-TS-SA algorithm more difficult to manage with more possibility to consider. After all, this observation prompts us to carefully refine the simple performance measure used in our proposed GLS-TS-SA solver for testing.

Besides, we also try to identify the best optimiser for each of the 6 classes of Solomon's test cases. We observe for the C1 and C2 classes, where there is no clear winner in term of the sum of *TV x TD*, PFIH+GLS is the simplest and most stable solver. Also, it is worth noting that GLS-TS-SA is marginally better in the C1 class with only *0.14%* deviated from the best published results. In addition, VVH+GLS-TS performs the best for both R2 and RC2-type problems. For a more detailed comparison on the performance of the different solvers in each individual problem class of the Solomon's test set, refer to [16].

6. CONCLUDING REMARKS

In this paper, we proposed our 2 interesting search hybrids by integrating the well-known guided local search (GLS) and Tabu search (TS) algorithms to effectively solve the VRPs. As opposed to the GTS in [1], we consider a different integration of Tabu Search into GLS framework as GLS-TS in which the GLS-type penalty scheme is invoked only at stable states. More interestingly, from GLS-TS, we implement the SA approach to work as the guiding principle to intelligently choose which is the *right* search method to apply in each iteration based on some cooling schedules. The cooling schedule is governed by a standard logistic function as defined in [11] together with some performance metric such as the reduction in the total distance travelled. There was some previous work which combined simulated annealing with Tabu search [11]. However, in our work, the SA mechanism is embedded into the combined GLS and Tabu search framework to decide which search method to apply in each search step.

In the empirical evaluation of our proposals on the Solomon's benchmarks, we obtained exciting results from our proposed GLS-TS algorithm. Generally speaking, GLS-TS is a very efficient algorithm compared to its ancestors, the GLS and GTS. Surprisingly, the VV+GLS-TS produces a better-than-best-published result for the R205 problem of the Solomon's test set. In addition, from our empirical results, we made some careful observations that improving on *TV* through any heuristic search operator can be more rewarding than the improvement on *TD* through the GLS-type penalty scheme on the long arcs since the objective function is *TV x TD*. This will definitely provide one exciting direction of our on-going research work. More importantly, we provide in this paper a simple and interesting probabilistic model for analysing the search behaviour of our proposed hybrids. This readily forms the basis for our future investigation.

Our new approach in the hybridisation of GLS and TS algorithms with simulated annealing truly opens up many new directions for future exploration. Here, we suggest to use the simulated annealing as a more sophisticated guiding principle, or namely meta-meta-heuristic, to monitor the underlying meta-heuristics such as GLS which in turn controls the search on the initial solution returned by the initialisation heuristics such as the PFIH or VVH. As for future work, we will look into other types of initialisation method to be integrated into our search hybrids to improve their performance. Another interesting direction is to investigate other possible logistic functions and performance metrics to particularly improve the performance of our proposed GLS-TS-SA solver. Lastly, it should be interesting to investigate other TS-related variants integrated with simulated annealing such as an adaptive strategy which can flexibly invoke the GLS penalty scheme within the TS framework depending on its relative merit.

ACKNOWLEDGEMENTS

The authors wish to thank Professor Edward Tsang for his fruitful discussion.

REFERENCES

[1] Bruno De Backer, Vincent Furnon, Philip Kilby, Patrick Prosser and Paul Shaw. Solving Vehicle Routing Problems using Constraint Programming and Met heuristics. Journal of Heuristics. Vol 1-16(1997), Kluwer Academic Publisher, 1997.

[2] Philip Kilby, Patrick Prosser and Paul Shaw. Guided Local Search for the Vehicle Routing Problems. *Proceedings of 2nd International Conference on Metaheuristics* (MIC97). Pp.1-10, Sophia-Antiplois, France, July 21-24,1997.

[3] E. Tsang, Chang J. Wang, Andrew Davenport, C. Voudouris and Tung L.L.. *A family of Stochastic Methods for Constraint Satisfaction and Optimisation.* The First International Conference on the Practical Application of Constraint Technologies and Logic Programming, London, April 1999

[4] Ben. Wah and Y.X. Chen. *Constrained Genetic Algorithms and their Applications in Nonlinear Constrained Optimization.* Proceedings of IEEE ICTAI. Vancouver, Cananda, November 2000.

[5] I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Ann. Oper. Res..* 41:421-451,1993.

[6] J.C. Jee, *Solving Vehicle Routing Problems with Time Windows Using Micro-Genetic Algorithms,* Proceedings of the 6th National Ungergraduate Research Opportunities Programme Congress 2000, Faculty of Engineering, NUS, September 7-8, 2000.

[7] P. Prosser and P. Shaw. *Study of greedy search with multiple improvement heuristics for vehicle routing problems.* Technical Report RR/96/201, Department of Computer Science, Univeristy of Strathclyde, Glasgow, January 1997.

[8] Rochat, Y. and Taillard, E. D.. *Probabilistic diversification and intensification in local search for vehicle routing.* Technical report CRT-95-13, Centre de recherche sur les transports, Université de Montréal, Montréal. 1995.

[9] D., Seah . *Ant Colonoy Optimization On Vehicle Routing Problems.* Thesis for the Master of Computer Science, National University of Singapore, 1999

[10] M.M. Solomon. *Algorithms for the vehicle routing and scheduling problem with time windows. Oper. Res.* 35:254-265,1987.

[11] William M. Spears. *Simulated Annealing for Hard Satisfiability Problems*. NCARAI Tech. Report AIC 93-015. AI Center, Naval Research Laboratory, Washington D.C., 1993.

[12] Chris Voudouris and Edward Tsang. *Technical Report CSM-247,* Department of Computer Science, Univeristy of Essex, August 1995

[13] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. Management Science, 40(10):1276-1290, 1994.

[14] S. Lin. Computer solutions for the traveling salesman problem. Bell Systems Technology Journal, 44:2245-2269, 1965.

[15] M. W.P. Savelsbergh. Computer aided routing. Centrum voor Wiskunde en Informatica, Amsterdam, 1988.

[16] K.T. Ma. Using Hybrid Search Methods to Solve A.I. Search and Optimization Problems. Hons Thesis, The National University of Singapore, March 2001.

Appendix A: Descriptions of the 6 Classes of Problems in Solomon's Test Set

Type	No. Of Cases	Description
C1	9	Vehicle capacity is small Spatial distribution of customers is clustered Width of servicing time window varies Service duration is large
C2	8	Vehicle capacity is large Spatial distribution of customers is clustered Width of servicing time window varies Service duration is large
R1	12	Vehicle capacity is small Spatial distribution of customers is uniformly distributed Width of servicing time window varies Service duration is small
R2	11	Vehicle capacity is large Spatial distribution of customers is uniformly distributed Width of servicing time window varies Service duration is small
RC1	8	Vehicle capacity is small Spatial distribution of customers is clustered Width of servicing time window varies Service duration is small
RC2	8	Vehicle capacity is large Spatial distribution of customers is clustered Width of servicing time window varies Service duration is small