SYSTEMATIC DIVERSIFICATION METAHEURISTIC FOR THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

OLLI BRÄYSY, GEIR HASLE

SINTEF Applied Mathematics, Department of Optimization P.O. Box 124 Blindern, N-0314 Oslo, Norway

JEAN BERGER, MOHAMED BARKAOUI

Defence Research Establishment Valcartier, Decision Support Technology Section 2459 Pie-XI Blvd. North, Val-Bélair, PQ, Canada, G3J 1X5

This paper introduces an extension of the multi-start local search framework [1, 2, 3], named Systematic Diversification for the Vehicle Routing Problem with Time Windows, where the objective is to design least cost routes for a fleet of vehicles from one depot to a set of geographically scattered points. The routes must be designed in such a way that each point is visited only once by exactly one vehicle within a given time interval; all routes start and end at the depot, and the total demands of all points on one particular route must not exceed the capacity of the vehicle. The problem has been under intensive research during the past few years because of its high practical importance. The suggested method is based on systematically penalizing solution features/attributes to diversify the search within a multi-start local search, where the basic idea is to create several initial solutions, and apply improvement heuristics to each of them. Then, the best solution found during the search is the final output. The experimental results on the well-known 100-customer benchmarks show that the described approach is reliable, effective and robust, being very competitive with all previous methods found in the literature.

1. Introduction

Routing and scheduling problems are important elements of many logistic systems. A lot of research has been done to solve these complex combinatorial problems in an effective way. One of the major research topics has been the Vehicle Routing Problem (VRP) that involves the design of a set of minimum-cost vehicle routes, starting and ending at a central depot, and servicing a set of orders of customers with a fleet of vehicles. Each customer has a given demand that has to be serviced using only one vehicle. A vehicle cannot service more customers than its capacity enables it to. In this paper we focus on a variant of VRP, where each customer has an allowable delivery time period. In the literature the problem is called the Vehicle Routing Problem with Time Windows (VRPTW) and it has been widely studied for the last decade. The time window constraint

is called hard if it must be satisfied and it is called soft if it can be violated. The exact formulation of the problem can be found for example in [4] and [5]. Real-life applications of VRPTW include such examples as deliveries of goods to department stores, school bus routing, bank deliveries, postal deliveries, newspaper and laundry distribution, security patrol and maintenance services etc. The common objective of the VRPTW consists of minimizing the number of routes or vehicles (primary criterion) and the total traveled distance (secondary criterion).

The VRPTW contains several NP-hard optimization problems, such as Traveling Salesman Problem (TSP) and bin packing, so one can say that VRPTW is NP-hard in the strong sense [6]. Even finding out whether a feasible solution exists is a NP-complete problem. Therefore, most of the research has focused on heuristic methods. Recent surveys for the VRPTW can be found in [4, 7, 8, 9]. Current VRPTW heuristics can be categorized as follows:

 (i) Construction heuristics

 (ii) Improvement heuristics (local searches)

(iii) Metaheuristics

Construction heuristics can be divided in sequential and parallel algorithms. Sequential algorithms build a route for each vehicle, one after another, using decision functions for the selection of the customer to be inserted in the route and the insertion position within the route. Parallel algorithms build the routes of all vehicles in parallel, using a precomputed estimate of the number of routes. Different variants of the construction heuristics for the VRPTW can be found in [10, 11, 12, 13, 14].

Most of the recently published VRPTW heuristics use a two-phase approach. First, a construction heuristic is used to generate a feasible initial solution. During the second phase, an iterative improvement heuristic is applied to the initial solution. These route improvement methods iteratively modify the current solution by performing local searches for better neighboring solutions. Generally, a neighborhood comprises the set of solutions that can be reached from the present one by swapping a subset of k arcs between solutions. For most successful applications to VRPTW, see [15, 16, 17, 18, 19, 20]. For more details on these approaches, we refer to [4].

To escape from local optima, the improvement heuristic can be embedded in a metaheuristic. In general, a metaheuristic is guided by some intelligent strategies to get away from trapped at some local optima, though the myopic move from one incumbent solution to its neighborhood solution is basically the same as in conventional local search methods. Examples of metaheuristics are simulated annealing, tabu search and genetic algorithm. For most successful recent applications to VRPTW, see [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 5, 33, 34, 35] and recent survey in [7].

The previously developed heuristics and metaheuristics show significant variability in performance. The solutions obtained with the traditional construction and improvement heuristics are often far away from optimum, and these methods are also highly dependent on the initial solutions. Better results can be obtained with different metaheuristics, but they often require considerable computational effort and fail to convincingly provide a single robust and successful technique. Consequently, there is a need to develop more robust, efficient and stable algorithms. The main contribution of this paper is to develop an extension of the Multi-Start Local Search (MSLS) [1, 2, 3] for combinatorial optimization problems in general, and for VRPTW in particular. The developed extension is named Systematic Diversification (SD), and here it is combined with the recent MSLS heuristic of [31], and with a new fast Threshold Accepting metaheuristic of [32]. In this scheme, the Systematic Diversification is used within the MSLS to guide the construction heuristic creating a set of initial solutions and new improvement heuristics based on well-known CROSS-exchanges [22]. The solution obtained with the Systematic Diversification is thereafter improved further in terms of distance with the Threshold Accepting post-optimization. It is shown that the Systematic Diversification increases the reliability, robustness and the solution quality of the original MSLS heuristic significantly, providing results that are competitive with all previous methods. Moreover, the Threshold Accepting is indicated to quickly provide considerable additional improvements.

The paper is arranged as follows. Section 2 presents the key ideas and principles of the Systematic Diversification along with the MSLS of [31] that is extended with the SD framework. Section 3 describes the Threshold Accepting metaheuristic introduced in [32] that is used as a separate post-optimization technique to further improve the solutions obtained by the Systematic Diversification. The overall computational results and comparisons are listed in section 4. Conclusions are drawn in Section 5.

2. Systematic Diversification Metaheuristic

2.1. The basic concepts

The key idea in Systematic Diversification is to analyze the differences between the created solutions with respect to some common attribute, and penalize search directions that increase the number of previously selected common attributes. Within a multi-start local search [1, 2, 3], the SD is applied both in the creation of initial solutions and in the improvement phase. In the initial solution phase the SD is used to guide creating a diverse set of initial solutions, and on the other hand, in the improvement phase the SD forces searching different regions of the search space.

The background for the SD framework is the typical two-phase approach for solving combinatorial optimization problems where the initial solution created in the first phase is subsequently improved in the second phase. The initial solution is often created either randomly or with some construction heuristic, and in the improvement phase the heuristic searches for better solutions are typically based on small neighborhood, trying only small

changes to incumbent solution. In practice this means that the final output is strongly dependent on the initial solution. MSLS tries to overcome this problem by creating several initial solutions and by improving each of them independently. Then, the best solution found during the whole search is the final output. Different population based metaheuristics such as genetic algorithms (see e.g. [36, 37]) go further by enabling an interaction between the created (initial) solutions, attempting to share good solution attributes between the individuals in the population. Similarly, the Systematic Diversification can be seen as an extension of the multi-start local search that is one of the most commonly used techniques for combinatorial optimization problems. As population based metaheuristics, SD enables an interaction between created individuals, but in a different way. Instead of combining existing solution attributes in a new way to create new individuals, the information on the created solutions is used to diversify the search. This is done by favoring the selection of solution attributes not selected before, provided that the solution quality is not worsened too much. The key advantage of this approach compared to a typical population based method is its simplicity. The performance of a typical population based metaheuristic is strongly dependent on appropriate solution encoding, diverse population, and most of all, good methods for performing interaction between individuals. These issues are problem dependent and extremely hard to solve efficiently in practice. There is no need to solve these issues in case of SD.

Here one must note that the idea of diversification is not new in the context of heuristic search. Different diversification strategies are common for instance in the context of tabu searches (see e.g. [26], [21] and [22]). The most common approaches here are to start from different points and to penalize frequently performed moves. Also SD creates different starting points, but focuses on creating these starting points such that they are as different as possible with respect to the chosen attribute and still of good quality, instead of using random strategies or mixture of different methods. We believe this is a more efficient way of creating a diverse set of initial solutions that is crucial from the viewpoint of getting good results. And instead of penalizing frequently performed moves, SD penalizes moves that increase the number of previously selected attributes. The advantage of this approach is that it is independent of the selected heuristic improvement procedure, it considers the whole set of created solutions and it is more general, i.e., it is not dependent on the actual modifications done. Penalizing just frequently performed moves can also increase the number of previously selected attributes if one considers all created solutions or the whole search history.

As for other metaheuristics, in genetic algorithms the diversification is based often on randomly created initial population, probabilistic selection of individuals and mutation that often applies some random strategies. On the other hand, De Jong [38] suggested a crowding model to introduce diversity among solutions in a population. The basic idea is to replace a solution in the current population that is most similar to the offspring. To our best knowledge there are no systematic efforts to force a diverse initial population or to penalize previously searched directions. Guided Local Search [39] is also based on the

idea of penalizing certain moves, but here the penalization is based on some bad features, not differences with respect to other, previously created solutions.

When generating the set of initial solutions, SD penalizes the selection of previously selected solution attributes each time already during the construction of a new solution. Here this is done by penalizing insertions within our cheapest insertion heuristic that increase the number of common attributes with respect to all previously created solutions. At the same time the alternative insertions cannot exceed a limit set to the allowed worsening of the objective value with respect to the best insertion.

On the other hand, the SD methodology is applied also during the improvement phase by systematically controlling the moves to neighboring solutions. The goal is to force the search to new areas in solution space by penalizing certain local searches to neighboring solutions. We suggest a multi-restart strategy where we start several ($\bar{n}$) times from the same initial solution, memorize the frequencies of attributes selected before, and penalize moves to neighboring solutions that increase the number of previously selected attributes. The algorithm can be presented as follows:

_____

Step 1. Repeat step 2. until λ solutions have been created.

Step 2. Create an initial solution S_i. Penalize selection of previously selected solution attributes during the construction of S_i and store S_i to set P.

Step 3. Repeat step 4. $\bar{n}$ times for all λ solutions S_i in P.

Step 4. Apply the given improvement operators to S_i until no more improvements can be found. If a new best solution is found, set $S_b=S_i$. When evaluating moves, penalize increases in the number of previously selected attributes.

Step 5. Return the best solution found, S_b.

_____

Alternatively, when creating a set of initial solutions, one could either create $\lambda > \mu$ solutions, where μ is the planned number of solutions in the set, and then select to the actual set only solutions that have least number of common attributes, and at the same time are of good quality. One could use also other methods to measure the differences between the given two solutions than the number of common attributes, such as Levenshtein distances [40].

2.2. Application to Vehicle Routing Problem with Time Windows

2.2.1 Overview

The SD was tested in the VRPTW context using the MSLS framework introduced in [9]. In this framework the search can be divided in three phases. In the first phase a set of initial solutions is created using a sequential cheapest insertion heuristic. Phases two and three focus on improving the initial solutions. In the second phase an attempt is made to decrease the number of routes, using a new technique called Injection Tree that has a lot of similarities with Ejection Chains [41, 42]. Finally, in the third phase the method tries to minimize the total distance using intra- and inter-route improvement heuristics that are modifications of CROSS-exchanges of [22]. Within this framework, the SD is used to

guide creation of a diverse set of initial solutions and the distance improvement heuristics to force separate search threads to explore different regions of the search space. Here we selected arcs as the feature to penalize, and maintain a table that stores the frequencies of selecting each arc.

2.2.2. Creation of initial solutions
The initial solution procedure constructs routes one at a time in sequential fashion. The routes are initialized by selecting randomly an as of yet unrouted customer among the customers farthest away from depot or among the customers with earliest deadline. The selection between these two criterions is done randomly based on user-defined probability each time when the algorithm starts to create a new solution. The probability of using the farthest customer from depot criterion is $\bar{p} = 0.8$, and correspondingly the probability for earliest deadline is $1 - \bar{p}$, i.e. 0.2. Next, the unrouted customers that are geographically close (distance within 0.4×maximum distance between any pair of customers in a problem) to at least one of the customers in the current partial route are considered for insertion one by one. Then, the customer that least increases the weighed combination of distance and waiting time of the route is inserted in the best feasible insertion position. In addition, we favour inserting customers located farthest away from the depot by subtracting from the insertion cost the distance of the corresponding customer to the depot multiplied with a user-defined parameter. More formally the cost function for customer c_u

$$C_u = \alpha_1 \cdot D_u + \alpha_2 \cdot W_u + \alpha_3 \cdot d_{0u} \tag{1}$$

where

$$D_u = d_{iu} + d_{uj} - d_{ij} \tag{2}$$

$$W_u = W_u^a - W_u^b \tag{3}$$

$$\alpha_1 + \alpha_2 = 1, \alpha_3 > 0$$

Notations d_{iu}, d_{uj} and d_{ij} refer to the distance between the corresponding pair of customers (c_i, c_u), (c_u, c_j) and (c_i, c_j) and W_u^b and W_u^a correspond to the total waiting time before and after the insertion, respectively. Finally d_{0u} is the distance from the customer c_u to the depot and α_1, α_2 and α_3 are parameter values determined by the user. We used the following values during all computational experiments: α_1: 0.6–1.0 (in increments of 0.1 units), $\alpha_2 = 1 - \alpha_1$ and α_3: 0.7–1.7 (in increments of 0.2 units). All possible combinations of values within the ranges specified for α_1 and α_3 are repeated 10 times (because of the randomly selected seed customers), resulting in 300 initial solutions in total. To speed up checking the feasibility of each insertion, we used the push-forward and push-backward strategies introduced in [43], and keep arrival times and latest possible arrival times at each customer in memory. In addition, we used the speedup technique of [31] to quickly evaluate the cost of each move.

Preliminary computational experiments indicated that even though we use quite powerful neighborhood operators, creating a diverse set of initial solutions is crucial to get good final solutions. To achieve this goal, the Systematic Diversification scheme is used to guide the creation of the initial solutions in the following way. We used SD to memorize the number of times each arc has been selected in the previous search phases, i.e., we maintain a table that stores the frequencies of selecting each arc, and penalize the selection of previously selected arcs when evaluating each insertion. This way we force creating solutions having different arcs with respect to previously created solutions. To control also the quality of the initial solutions, we set a limit $\bar{b}$ to how much more costly the alternative customer and/or insertion place can be compared to the cheapest insertion. In the beginning $\bar{b} = 1$, and its value is increased gradually $b_i = 0.05$ units at a time every time after creating $\bar{f} = 8$ new solutions until the maximum value of $b_m = 1.25$ is reached. Finally, to limit the influence of the diversification scheme, the table storing the arc frequencies is initialized every time after $\overline{m} = 50$ solutions have been created, and $\bar{b}$ is set to $\bar{b} = 1$.

2.2.3. Improvement phase

The SD framework was applied also to the third phase of distance minimization, but not in the route reduction phase. The main reason for this is that the injection tree procedure is mainly dependent on the starting point, i.e., the set of different initial solutions, but not on the selections done within the procedure. Moreover, it would be too time consuming to restart the tree search several times. As mentioned above, the Injection Tree shares similarities with Ejection Chains (EC) [41, 42]. In ejection chains the basic idea is to combine series of simple moves into a compound move. In VRP context these simple moves refer to removal of a customer from its route and reinsertion of the removed customer in another route. The goal is to "make room" for a new customer in a route by first removing another customer from the same route. In each phase within the ejection chain, one customer remains unrouted. The removal and insertion procedures are repeated until one can insert a customer to another route without the need to remove (eject) any customer. For more details on the application of ejection chains in VRP context, we refer to [28], [15], [44, 45] and [46].

The injection tree was built on the ejection chain procedure suggested in [28]. The main innovation in the ejection chain approach described in [28] is reordering of the routes within the ejection chain. If inserting a customer c_i in another route r_t fails due to time window constraints, simple intra-route reinsertions are tried to make r_t feasible after inserting c_i. The basic idea of these reinsertions is to reduce lateness in the infeasible route by trying to serve some customers in alternate locations within the same route. The same strategy is used also in our injection tree, and the number of reinsertions is restricted to $\bar{i} = 5$. We used breadth-first search, and the number of trees stored to memory is restricted to 1000 to keep the memory requirements reasonable. Also the maximum depth of the tree is limited to $\bar{c} = 8$ and we ignore insertions that increase the length of the target route more than a pre-defined limit $\bar{l}$ to reduce the computational effort. The value of $\bar{l}$ is adjusted dynamically during the search such that only the most

valuable insertions are kept in memory in case the number of trees exceeds 1000. The starting value of $\bar{l}$ is 1.25 and the value is adjusted in steps of 0.1 units. Since it is computationally easier to eliminate shorter routes, routes having least customers are considered first for elimination, and insertions are first tried to routes that are geographically closest. Moreover, if the distance between the customer c_i currently considered for insertion and all the customers in a given route r_t exceeds $\bar{d} = 0.3$ times the maximum distance between any pair of customers in the problem, the customer c_i is not even tried to insert to r_t. We used push-forward and push backward strategies introduced in [43] to speed up the feasibility checks of each insertion or reinsertion within the reordering operator. In addition, to further increase the efficiency of feasibility checks, we maintained in memory the arrival times and latest possible arrival times at each customer.

Contrary to ejection chains, instead of removing just one customer from the destination route r_t to make it feasible, unlimited number of customers can be removed, provided that the removed customers can be inserted directly in some neighboring route $r_n \neq r_t$ without the need to remove any customer from r_n. If these re-insertions cannot make r_t feasible, the successful insertions are reversed, and we proceed as in standard breadth-first search ejection chain. That is, we consider all possible removals of one customer from r_t that make it feasible after inserting c_u. Here we start from the customers causing highest increase in route distance, and consider first removing and reinserting them in alternate locations. This way we consider first all possible chains involving two insertions and one ejection, then chains involving three insertions and two ejections and so on. Instead of first ejecting a customer from a destination route to make room for a new customer, in our implementation the new customer is inserted directly to the destination route, causing often the violation of time or capacity constraints. The rationale behind this approach is that by doing the insertion only once instead of repeating the insertion of the same customer after every removal from r_t, one can save a lot of computational efforts.

The distance improvement is based on modifications of the well-known CROSS-exchanges of [22], used for both intra- and inter-route improvements. In CROSS-exchanges the basic idea is to relocate or exchange segments of consecutive customers between two routes such that the orientation is preserved. Two modifications were introduced. The first is to consider also inserting the customers in the currently selected segment in inverted order. In [22] the segment from route R_2 is always inserted in a location, where the segment from route R_1 was removed. The second modification is that in our implementation we consider in addition the most promising insertion position (geographically closest) for search segment removed from R_2. The maximum segment length is set to $\bar{s} = 5$ and we use the best-accept strategy. The SD is used in the evaluation of each move to penalize moves that increase the number of arcs selected in the previous search phases. The distance minimization is started $\bar{n}$ (=8) times from each solution in the set of initial solutions that has the lowest found number of routes, and the

penalty of selecting previously selected arcs is set to infinite. This penalization of previously selected arcs forces the search to new directions in each new search thread.

3. Threshold Accepting

In this section we describe the Threshold Accepting (TA) metaheuristic introduced in [32] for the VRPTW. It is used as a post-optimization technique to further improve the solutions produced by the Systematic Diversification. The TA is based on a modification of well-known Or-Opt-exchanges [47] and a new improvement operator, named GENICROSS that combines the basic ideas of well-known CROSS-exchanges [22] and GENIUS operator [48].

Threshold Accepting, introduced in Dueck et al. [49, 50], is a modification of the well-known Simulated Annealing metaheuristic [51]. In Threshold Accepting also neighboring solutions that worsen the objective function value are accepted if the difference to the incumbent solution is smaller or equal to a deterministic threshold, t. The key components of TA are the function $g(t)$ that determines the lowering of the threshold during the course of the search, and stopping criteria as well as the methods used to create initial and neighboring solutions.

In our implementation $g(t)$ is linear, i.e., we start from a user-defined maximum threshold $t_{max} = 1$ and reduce it in steps of $\Delta t = 0.025$ units at a time until zero is reached. Then, we set the threshold to the maximum value again ($t = t_m$), and repeat the procedure until the stopping criteria are met. Moreover, we maintain in memory the current best solution found during the search, and if any improvements have not been found for a certain number of iterations, $\bar{k} = 45$, we return to the best solution and set the threshold to the maximum. The search is stopped once a user-defined number of iterations, $\bar{n} = 500$, has been tried, and each iteration the routes in the current solution are put in random order to further diversify the search. The initial solutions are created using the Systematic Diversification metaheuristic described in the previous section and for exploring the neighboring solutions we use the IOPT-operator, introduced in [28] and GENICROSS introduced in [32]. IOPT is a modification of well-known Or-opt exchanges, where also inversion of the order of the customers in the currently selected segment is considered. Here it is used only for intra-route optimization. GENICROSS draws its basic ideas from the well-known GENIUS insertion heuristic of [48] and CROSS-exchanges of [22]. The basic idea in CROSS-exchanges is to relocate or exchange segments of consecutive customers between two routes R_1 and R_2 such the orientation of the customers in the selected segments is preserved. In [22] segment S_2 currently served on route R_2 is always tried to insert only to a location in route R_1, where segment S_1 was removed, instead of trying all possible locations for S_2 in R_1. Contrary to CROSS-exchanges, GENICROSS considers all insertion places for S_2 in R_1, and we used the first-accept strategy. In addition, GENICROSS considers also moves where the order of the customers in the selected segments is inverted.

To speed up the algorithm, we evaluate first the cost (change in objective function) of the exchange, and then check the feasibility only if some improvement is found. Moreover, an exchange of segments S_1 and S_2 between routes R_1 and R_2 is divided in two insertions: S_1 from R_1 to R_2 and S_2 from R_2 to R_1, and we consider inserting S_2 to R_1 only if insertion of S_1 to R_2 improved the objective function value. We found that this provides in practice substantial speedups. To evaluate all possible moves, the GENICROSS must be performed twice, inverting the role of the routes in between. The limitation of this approach is that it considers only insertions between pairs of consecutive customers in the current routes. However, it is possible that for example the insertion position of S_1 in route R_2 belongs to segment S_2 removed previously from R_2. In this case the actual insertion of S_1 will be between customers before and after the removed segment, causing that the insertion will not be made between a pair of originally consecutive customers. So, as in GENIUS insertion heuristics of [48], we consider here also insertions between pairs of customers that are not consecutive. Thus, in addition to evaluating the insertion between locations d_1 and d_1+1 in R_2, we also consider insertions between d_1 and d_1+2, $d_1+3,\ldots,d_1+\bar{s}$. To speed up the feasibility checks of each move, we use the well-known push-forward and push-backward strategies introduced in [43]. Moreover, we maintain in memory the arrival times and latest possible arrival times at each customer to quickly evaluate the feasibility of the time window constraints. In case some infeasibilities are found, further checks are disregarded. The algorithm is:

Step 1. Start with feasible solution S_i, created by the SD (see section 2).

Set $t = t_{max}$ and $S_b = S_i$.

Step 2. Repeat steps 3–4 for $\bar{n}$ iterations.

Step 3. Order the routes in S_i randomly.

Step 4. Repeat steps 5–7 for all pairs of routes in S_i

Step 5. Apply GENICROSS to the current pair of routes in S_i.

Step 6. Apply IOPT to both individual routes in the current pair of routes in S_i.

Step 7. Lower the threshold by $\Delta t : t = t - \Delta t$.

If a new best solution is found, store it: $S_b = S_i$.

If no improvement has been found for $\bar{k}$ iterations, restart the search from the best solution obtained so far, and set $S_i = S_b$ and $t = t_{max}$.

If $t = 0$ for 4 iterations, set $t = t_{max}$.

Step 8. Return the best solution found, S_b.

---

4. Computational experiment

4.1. Problem data

In the literature there is a well-known collection of 56 VRPTW instances from [10] that is used by many researchers for evaluation of their VRPTW solving systems. These problems can be classified into three groups, with distinct characteristics of the

distribution of the customer locations: random (R), clustered (C), and a mixture of both (RC). Furthermore, each of these groups can be split into problems with low vehicle capacity (type 1) and high vehicle capacity (type 2). One therefore generally speaks of six problem sets, namely R1, C1, RC1, R2, C2, and RC2. Members of a single set vary both in the distribution of the time windows and in the demands of the single customers. All problems consist of a hundred customer locations and one depot. Both the distances and the travel times between the customers are given by the corresponding Euclidean distances. Therefore, this library incorporates many distinguishing features of vehicle routing with time windows: fleet size, vehicle capacity, spatial and temporal customer distribution, time window density, time window width, and customer service times. The computational experiments were conducted using a 700 MHz personal computer with 128 MB memory.

4.2. Computational results

Table 1 illustrates the performance and the reliability of the suggested Systematic Diversification metaheuristic. The Table is divided in three parts, presenting the results obtained with the local search of [31] and the Systematic Diversification metaheuristic with and without the Threshold Accepting post-optimization, respectively. For all three approaches we describe the best, average and worst results obtained to Solomon's benchmark groups over three independent runs. For each six benchmark group we present both the number of routes and total distance averaged with respect to the corresponding problem group.

Table 1. Comparison of the best, worst and average results of the local search and the systematic diversification metaheuristic with and without the Threshold Accepting post-optimization.

Method	R1	R2	C1	C2	RC1	RC2
LS/BEST	**12.00**	**2.73**	**10.00**	**3.00**	**11.50**	**3.25**
	1235.22	979.88	828.38	589.86	1413.50	1152.37
LS/AVERAGE	**12.08**	**2.73**	**10.00**	**3.00**	**11.58**	**3.25**
	1244.04	997.27	828.41	590.15	1424.44	1170.37
LS/WORST	**12.17**	**2.73**	**10.00**	**3.00**	**11.63**	**3.25**
	1253.04	1013.22	828.48	590.30	1441.28	1186.98
SD/BEST	**12.00**	**2.73**	**10.00**	**3.00**	**11.50**	**3.25**
	1222.03	963.86	828.38	589.86	1393.22	1129.54
SD/AVERAGE	**12.03**	**2.73**	**10.00**	**3.00**	**11.58**	**3.25**
	1230.13	969.19	828.38	589.86	1399.50	1135.95
SD/WORST	**12.08**	**2.73**	**10.00**	**3.00**	**11.63**	**3.25**
	1236.12	975.67	828.38	589.86	1407.51	1144.76
SD+TA/BEST	**12.00**	**2.73**	**10.00**	**3.00**	**11.50**	**3.25**
	1212.83	956.43	828.38	589.86	1387.22	1126.40
SD+TA/AVERAGE	**12.03**	**2.73**	**10.00**	**3.00**	**11.58**	**3.25**
	1223.02	962.60	828.38	589.86	1392.10	1133.84
SD+TA/WORST	**12.08**	**2.73**	**10.00**	**3.00**	**11.63**	**3.25**
	1230.55	968.19	828.38	589.86	1400.32	1142.98

According to Table 1, the Systematic Diversification improves both the solution quality and the reliability of the local search of [31] significantly. Even the best solutions that are already very good are improved on average by 1%, and correspondingly the difference in average solutions is about 1.5% in terms of distance over all six problem groups. In number of routes the differences remain very small. The reason for this is that the local search performs very well regarding the number of routes also independently. The biggest improvements can be found in the group RC2 that is often considered to be the most complex one regarding distance minimization. The difference between the local search and the Systematic Diversification is even 2.0–3.7%. By comparing the middle and lower part of Table 1, one can see that the Threshold Accepting is able to improve even the best solutions of SD that are already very good by 0.4%, indicating the power of the procedure as a post-optimization technique.

As can be seen from Table 1, the local search of [31] is quite reliable. The average results are close to the best ones in all problem groups, and in general, the difference between the best and the worst results is about 1.7%. The biggest difference, 3.4%, can be found in the group R2. According to Table 1, the Systematic Diversification is able to reduce the variance of the local search considerably, thus increasing the reliability even further. The differences between the best and the average, and the best and the worst results of SD are about 0.4% and 0.8%, respectively. On the other hand, the Threshold Accepting post-optimization does not affect the reliability, implying its dependence on the initial solutions.

In Table 2 we illustrate the speed and robustness of our approach by comparing our best results over three runs using constant parameter setting with the results of well-known and best-performing methods found in the literature. We included in Table 2 only approaches where sufficient amount of information is provided by the authors. At least the computer, number of computational runs as well as the time consumption, number of routes and total traveled distance must be reported to make the comparison possible. The first column to the left lists the authors and columns R1, R2, C1, C2, RC1 and RC2 present the average number of routes and average total distance with respect to the six problem groups of [10]. The CNV/CTD column indicates the cumulative number of vehicles (CNV) and cumulative total distance (CTD) over all 56 test problems. The computer, number of runs, and the CPU time used to obtain the reported results are described in the rightmost column of Table 2. For each method two CPU times are given: the one reported by the authors, and in parenthesis a modified CPU, where we use the factors of Dongarra [52] to make the running times obtained with different computers comparable. Moreover, if the results in Table 2 are the best ones over multiple independent runs, the computation times are multiplied by this number to illustrate the total computational effort. The reader must note that the computation times presented in the parenthesis are only indicative, and should be used only to get a rough picture of the effort required to get the reported results. In the lower part of the Table, we present the best-known results obtained with numerous different methods, and the results obtained with our Systematic Diversification with and without the post-optimization.

Table 2. The performance comparison among the VRPTW algorithms.

Author	R1	R2	C1	C2	RC1	RC2	CNV/CTD	CPU
[21]	**12.58**	**3.09**	**10.00**	**3.00**	**12.38**	**3.62**	**427**	Silicon Graphics,
	1197.42	954.36	828.45	590.32	1369.48	1139.79	57120	1 run, 92.2 (**138**) min.
[22]	**12.33**	**3.00**	**10.00**	**3.00**	**11.90**	**3.38**	**417**	Sun Sparc 10,
	1220.35	1013.35	828.45	590.91	1381.31	1198.63	58614	1 run, 248 (**248**) min.
[25]	**11.92**	**2.73**	**10.00**	**3.00**	**11.63**	**3.25**	**406**	Pentium 200 MHz,
	1228.06	969.95	828.38	589.86	1392.57	1144.43	57876	10 runs, 13 (**312**) min.
[53]	**12.42**	**2.82**	**10.00**	**3.00**	**11.88**	**3.25**	**415**	4×Pentium 200 MHz,
	1198	947	829	590	1356	1144	56946	1 run, 5 (**48**) min.
[24]	**12.17**	**2.82**	**10.00**	**3.00**	**11.88**	**3.25**	**412**	HP 9000/720,
	1249.57	1016.58	830.06	591.03	1412.87	1204.87	59318	3 runs, 20 (**102**) min.
[23]	**12.38**	**3.00**	**10.00**	**3.00**	**11.92**	**3.33**	**418**	Sun Ultrasparc 1,
	1210.83	960.31	828.38	591.85	1388.13	1149.28	57583	1 run, 30 (**210**) min.
[46]	**12.08**	**3.00**	**10.00**	**3.00**	**11.63**	**3.38**	**412**	Sun Ultra 10,
	1210.21	941.08	828.38	589.86	1382.78	1105.22	56953	10 runs,183(**19430**) min.
[27]	**12.00**	**2.73**	**10.00**	**3.00**	**11.50**	**3.25**	**406**	4×Pentium 400 MHz,
	1217.57	961.29	828.63	590.33	1395.13	1139.37	57641	5 runs, 13.5 (**1458**) min.
[28]	**11.92**	**2.73**	**10.00**	**3.00**	**11.50**	**3.25**	**405**	Pentium 200 MHz,
	1222.12	975.12	828.38	589.86	1389.58	1128.38	57710	1 run, 82.5 (**198**) min.
[33]	**12.17**	**2.73**	**10.00**	**3.00**	**11.75**	**3.25**	**410**	Pentium 400 MHz,
	1230.22	1009.53	828.48	589.93	1397.63	1230.20	59065	3 runs, 30 (**486**) min.
[29]	**12.17**	**2.73**	**10.00**	**3.00**	**11.63**	**3.25**	**409**	Sun Ultra 10, 440 MHz,
	1203.84	980.31	828.38	589.86	1379.03	1158.91	57707	5 runs, 30 (**1590**) min.
[3]	**12.08**	**2.91**	**10.00**	**3.00**	**11.75**	**3.25**	**411**	Pentium 545 MHz,
	1215.14	953.43	828.38	589.86	1385.47	1142.48	57467	3 runs, 30 (**594**) min.
[34]	**11.92**	**2.73**	**10.00**	**3.00**	**11.63**	**3.25**	**406**	Pentium 1 GHz,
	1220.02	961.64	828.38	589.86	1378.72	1132.17	57 480	1 run, 33 (**559**) min.
[5]	**12.00**	**2.73**	**10.00**	**3.00**	**11.50**	**3.25**	**406**	AMD 700 MHz,
	1220.14	977.57	828.38	589.86	1397.44	1140.06	57870	3 runs, 9.1 (**374**) min.
[31]	**12.00**	**2.73**	**10.00**	**3.00**	**11.50**	**3.25**	**406**	AMD, 700 MHz,
	1235.22	979.88	828.38	589.86	1413.50	1152.37	58303	3 runs, 2.1 (**86**) min.
[35]	**12.08**	**2.73**	**10.00**	**3.00**	**11.50**	**3.25**	**407**	5×Pentium 850 MHz,
	1209.19	963.62	828.38	589.86	1389.22	1143.70	57 412	1 run, 12 (**810**) min.
BEST	**11.92**	**2.73**	**10.00**	**3.00**	**11.50**	**3.25**	**405**	____
KNOWN	1209.92	952.09	828.38	589.86	1384.16	1119.42	57195	
SD	**12.00**	**2.73**	**10.00**	**3.00**	**11.50**	**3.25**	**406**	AMD, 700 MHz,
	1222.03	963.86	828.38	589.86	1393.22	1129.54	57627	3 runs, 3.1 (**129**) min
SD+TA	**12.00**	**2.73**	**10.00**	**3.00**	**11.50**	**3.25**	**406**	AMD 700 MHz,
	1212.83	956.43	828.38	589.86	1387.22	1126.40	57361	3 runs, 3.7 (**152**) min.

According to Table 2 it seems that as the local search of [31], our systematic diversification metaheuristic is able to find the lowest known number of routes to all benchmark instances except one. Only [28] reports the best-known CNV, when only limited computational experiments are performed. For clustered problem groups C1 and C2 many authors report the optimal solutions. As for the remaining four groups, the combination of Systematic Diversification and Threshold Accepting dominates all previous approaches in three cases. Only for group R1, [28] and [25] report slightly better average number of routes. Here one must note that the due to conflicting nature of minimizing the number of routes and total distance, the comparison over distance values

is fair only over same number of routes that is considered as the primary objective. When comparing our results with the best-known results obtained with all previous approaches, it can be seen that the results are very close to the best known. The differences in total distance are only about 0.3% on the average, and we obtained the lowest known number of vehicles to all problems except R104. As for the CPU time, it seems that our approaches are faster than most of the previous metaheuristics. Some of the earlier papers of [21], [53] and [24] report lower computation times, but fall clearly behind in terms of solution quality. The same applies also to other earlier heuristic methods.

5. Conclusions

The vehicle routing problem with time windows has been under intensive research in the past few years due to its high practical importance. In this paper, we proposed a new heuristic method for solving combinatorial optimization problems in general, and vehicle routing problem with time windows in particular. The new method was named Systematic Diversification, and it is an extension of the multi-start local search that is one of the most commonly used approaches for combinatorial optimization problems. The basic idea of Systematic Diversification is to penalize search directions that increase the number of previously selected solution attributes within a multi-start local search to diversify the search. The Systematic Diversification was coupled with the local search algorithm of [31] to guide the creation of diverse set of initial solutions and improvement heuristics for distance minimization. In addition, the Threshold Accepting metaheuristic of [32] was applied as an independent post-optimization technique. The experimental results indicate that the proposed metaheuristic improves the performance of the original multi-start local search significantly. The suggested approach is shown to be very efficient and robust, outperforming previous approaches, and producing results within 0.3% from the best known.

6. References

[1] S. Lin, *Computer solutions of the traveling salesman problem*, Bell System Tech. J. **44** (1965) 2245–2269.

[2] S. Reiter and G. Sherman, *Discrete optimizing*, J. Soc. Ind. App. Math. **13** (1965) 864–889.

[3] H. Li, A. Lim and J. Huang, *Local search with annealing-like restarts to solve the VRPTW*, to appear in Eur. J. Oper. Res. (2002).

[4] O. Bräysy and M. Gendreau, *Vehicle routing problem with time windows, part I: route construction and local search algorithms*, to appear in Transp. Sci. (2002).

[5] O. Bräysy and W. Dullaert, *A fast evolutionary metaheuristic for the vehicle routing problem with time windows*, Int. J. AI Tools **12** (2003) 153−172.

[6] N. Kohl, *Exact methods for time constrained routing and related scheduling problems*, Ph.D. Thesis, Technical University of Denmark (1995).

[7] O. Bräysy and M. Gendreau, *Vehicle routing problem with time windows, part II: metaheuristics*, to appear in Transp. Sci. (2002).

[8] J.-F. Cordeau, G. Desaulniers, J. Desrosiers, M.M. Solomon and F. Soumis, *The VRP with time windows*, in The Vehicle Routing Problem, SIAM Monographs Disc. Math. App., eds. P. Toth and D. Vigo, SIAM, Philadelphia (2001) 157−194.

[9] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis, *Time constrained routing and scheduling*, in Handbooks in OR and MS 8: Network Routing, eds. M. Ball, T. Magnanti, C. Monma and G. Nemhauser, Elsevier, Amsterdam (1995) 35−139.

[10] M.M. Solomon, *Algorithms for the vehicle routing and scheduling problems with time window constraints*, Oper. Res. **35** (1987) 254−265.

[11] J.-Y. Potvin and J.-M. Rousseau, *A parallel route building algorithm for the vehicle routing and scheduling problem with time windows*, Eur. J. Oper. Res. **66** (1993) 331−340.

[12] J. Bramel and D. Simchi-Levi, *Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows*, Oper. Res. **44** (1996) 501−509.

[13] G. Ioannou, M. Kritikos and G. Prastacos, *A greedy look-ahead heuristic for the vehicle routing problem with time windows*, J. Oper. Res. Soc. **52** (2001) 523−537.

[14] W. Dullaert and O. Bräysy. *Routing with relatively few customers per route*, to appear in Top (2002).

[15] Y. Caseau and F. Laburthe, *Heuristics for large constrained vehicle routing problems*, J. Heuristics **5** (1999) 281−303.

[16] R. Cordone and R. Wolfler-Calvo, *A heuristic for the vehicle routing problem with time windows*, J. Heuristics **7** (2001) 107−129.

[17] J.-Y. Potvin and J.-M. Rousseau, *An exchange heuristic for routeing problems with time windows*, J. Oper. Res. Soc. **46** (1995) 1433−1446.

[18] R.A. Russell, *Hybrid heuristics for the vehicle routing problem with time windows*, Transp. Sci. **29** (1995) 156−166.

[19] P. Shaw, *Using constraint programming and local search methods to solve vehicle routing problems*, in Principles and Practice of Constraint Programming, Lect. Notes Comp. Sci., eds. M. Maher and J.-F. Puget, Springer, New York (1998) 417−431.

[20] P.M. Thompson and H.N. Psaraftis, *Cyclic transfer algorithms for multivehicle routing and scheduling problems*, Oper. Res. **41** (1993) 935−946.

[21] Y. Rochat and E. Taillard, *Probabilistic diversification and intensification in local search for vehicle routing*, J. Heuristics **1** (1995) 147−167.

[22] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin, *A tabu search heuristic for the vehicle routing problem with soft time windows*, Transp. Sci. **31** (1997) 170−186.

[23] L.M. Gambardella, E. Taillard and G. Agazzi, *MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows*, in New Ideas in Optimization, eds. D. Corne, M. Dorigo and F. Glover, McGraw-Hill, London (1999) 63–76.

[24] F.-H. Liu and S.-Y. Shen, *A route-neighborhood-based metaheuristic for vehicle routing problem with time windows*, Eur. J. Oper. Res. **118** (1999) 485–504.

[25] J. Homberger and H. Gehring, *Two evolutionary metaheuristics for the vehicle routing problem with time windows*, INFOR **37** (1999) 297–318.

[26] J.-F. Cordeau, G. Laporte and A. Mercier, *A unified tabu search heuristic for vehicle routing problems with time windows*, J. Oper. Res. Soc. **52** (2001) 928–936.

[27] H. Gehring and J. Homberger, *Parallelization of a two-phase metaheuristic for routing problems with time windows*, Asia-Pac. J. Oper. Res. **18** (2001) 35–47.

[28] O. Bräysy, *A reactive variable neighborhood search for the vehicle routing problem with time windows*, to appear in INFORMS J. Comput. (2001).

[29] R. Bent and P. Van Hentenryck, *A two-stage hybrid local search for the vehicle routing problem with time windows*, to appear in Transp. Sci. (2001)

[30] S. Jung and B.-R. Moon, *A hybrid genetic algorithm for the vehicle routing problem with time windows*, Proc. GECCO, New York (July 2002) 1309–1316.

[31] O. Bräysy, G. Hasle and W. Dullaert, *A multi-start local search algorithm for the vehicle routing problem with time windows*, to appear in Eur J. Oper. Res. (2003).

[32] O. Bräysy, J. Berger, M. Barkaoui and W. Dullaert, *A threshold accepting metaheuristic for the vehicle routing problem with time windows*, to appear in Central Eur. J. Oper. Res. (2003).

[33] J. Berger, M. Barkaoui and O. Bräysy, *A route-directed hybrid genetic approach for the vehicle routing problem with time windows*, INFOR **41** (2003) 179–194.

[34] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno and M. Yagiura, *Effective local search algorithms for routing and scheduling problems with general time window constraints*, to appear in Transp. Sci. (2003).

[35] A. Le Bouthillier and T.G. Crainic, *A cooperative parallel meta-heuristic for the vehicle routing problem with time windows*, Working Paper, Centre for Research on Transportation, University of Montreal, Canada (2003).

[36] D. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison Wesley Publishing Company, New York (1989).

[37] J.H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor (1975).

[38] K.A. De Jong, *An analysis of the behavior of a class of genetic adaptive systems*, Ph.D. Thesis, University of Michigan (1975).

[39] C. Voudouris and E. Tsang, *Guided local search*, Eur. J. Oper. Res. **113** (1998) 80–119.

[40] V. Levenshtein, *Binary codes capable of correcting deletions, insertions, and reversals*, Soviet Physics – Doklady **10** (1966) 707–710.

[41] F. Glover, *Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem*, Working Paper, College of Business & Administration, University of Colorado (1991).

[42] F. Glover, *New ejection chain and alternating path methods for traveling salesman problems*, in Computer Science and Operations Research: New Developments in Their Interfaces, eds. O. Balci, R. Sharda and S. Zenios, Pergamon Press, Oxford (1992) 449–509.

[43] M.M. Solomon, E.K. Baker and J.R. Schaffer, *Vehicle routing and scheduling problems with time window constraints: efficient implementations of solution improvement procedures*, in Vehicle Routing: Methods and Studies, eds. B. Golden and A. Assad, Elsevier, Amsterdam (1988) 85–106.

[44] C. Rego, *A subpath ejection method for the vehicle routing problem*, Manage. Sci. **44** (1998) 1447–1459.

[45] C. Rego, *Node ejection chains for the vehicle routing problem: Sequential and Parallel Algorithms*, Parallel Computing **27** (2001) 201–222.

[46] L.-M. Rousseau, M. Gendreau and G. Pesant, *Using constraint-based operators to solve the vehicle routing problem with time windows*, J. Heuristics **8** (2002) 43–58.

[47] I. Or, *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*, Ph.D. thesis, Northwestern University (1976).

[48] M. Gendreau, A. Hertz and G. Laporte, *A new insertion and postoptimization procedures for the traveling salesman problem*, Oper. Res. **40** (1992) 1086–1093.

[49] G. Dueck and T. Scheuer, *Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing*, J. Comput. Physics **90** (1990) 161–175.

[50] G. Dueck, T. Scheuer and H.-M. Wallmeier, *Toleranzschwelle und sintflut: neue ideen zur optimierung*, Spektrum der Wissenschaft **3** (1993) 42–51.

[51] S. Kirkpatrick, C.D. Gelatt and P.M. Vecchi, *Optimization by simulated annealing*, Science **220** (1983) 671–680.

[52] J. Dongarra, *Performance of various computers using standard linear equations software*, Report CS-89-85, Department of Computer Science, University of Tennessee (1998).

[53] H. Gehring and J. Homberger, *A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows*, in Proc. EUROGEN99, eds. K. Miettinen, M. Mäkelä and J. Toivanen, University of Jyväskylä, Jyväskylä (1999) 57–64.