A Parallel Algorithm for the Vehicle Routing Problem with Time Window Constraints *

Jürgen Schulze and Torsten Fahle

Department of Computer Science, University of Paderborn,
Fürstenallee 11, 33102 Paderborn, Germany
E-mail: {j.schulze,tef}@uni-paderborn.de

In this paper we describe a new parallel tabu search heuristic for the vehicle routing problem with time window constraints (VRPTW). The neighborhood structure we propose is based on simple customer shifts and allows to consider infeasible interim-solutions. Similar to the column generation approach used in exact algorithms, all routes generated by the tabu search heuristic are collected in a pool. To obtain a new initial solution for the tabu search heuristic a fast set covering heuristic is periodically applied to the routes in the pool. The parallel heuristic has been implemented on a Multiple-Instruction Multiple-Data computer architecture with eight nodes. Computational results for Solomon's benchmark problems demonstrate that our parallel heuristic can produce high quality solutions.

Keywords: Vehicle routing problem, time windows, tabu search, parallel algorithms
AMS Subject classification: 90B06, 68Q22

1. Introduction

Many problems in distribution management can be formulated as vehicle routing problems with time window constraints (VRPTW). The objective of the VRPTW is to deliver a set of customers with known demands on minimum-cost vehicle routes originating and terminating at a central depot. Each customer is assigned to exactly one vehicle route and the total demand of any route must not exceed the vehicle capacity. In addition, a time window is associated with each customer defining an interval wherein the customer has to be supplied. Especially for companies using just-in-time manufacturing time windows are strict and, in general, very narrow (less than one hour). Here, strict means that a solution becomes infeasible if a customer is supplied after the upper bound of its time window. In the case of soft time windows a later service does not affect the feasibility of the solution but is penalized by adding a value to the objective function. A vehicle arriving before the lower limit of the time window causes

* Supported by the German Federal Department of Science and Technology (PARALOR Project) and EC HC&M Project SCOOP.

additional waiting time on the route. Each route must start and end within the time window associated with the depot.

In this paper all time windows are assumed to be strict. The objective is to minimize the vehicle fleet and the sum of travel time and waiting time needed to supply all customers. Because of the NP-hardness of the vehicle routing problem (Lenstra and Kan [22]), only small to medium sized problem instances (up to 100 customers) can be solved to optimality (see, e.g., Desrochers et al. [6], Gélinas et al. [12], Kohl and Madsen [19], Fisher et al. [9]). Therefore, during the last decades much effort has been devoted to the development of powerful heuristic algorithms. Surveys of exact and heuristic algorithms for the solution of the vehicle routing problem with and without time window constraints can be found in Gendreau et al. [14], Laporte [20], Desrosiers et al. [7], and Fisher [8]. For a recent bibliography with 500 references on four classical routing problems we refer to Laporte and Osman [21].

Most of the recently published VRPTW heuristics are two-phase approaches. First, a construction heuristic is used to generate a feasible and as good as possible initial solution. Construction heuristics can be divided in sequential and parallel route building methods. The former successively build the routes of the initial solution (see, e.g., Solomon's insert heuristic [34]), whereas the latter build several routes simultaneously (see, e.g., the parallel insertion algorithm of Potvin and Rousseau [28]). During the second phase an iterative improvement heuristic is applied to the initial solution. Russell [32] developed a hybrid heuristic that merges the two phases, i.e., an improvement procedure is invoked periodically during route construction.

To escape from local optima, the improvement procedure is embedded in a metaheuristic like simulated annealing (Chiang and Russell [5]) or tabu search (Potvin et al. [27], Rochat and Taillard [30], Taillard et al. [36]). Recently, a new metaheuristic called simulated trading was introduced by Bachem et al. [1]. The concept of simulated trading is to generate new assignments of customers to routes by applying the mechanisms of trading. The heuristic was implemented on a cluster of workstations using the message passing library PVM [11]. In addition, several researchers have used genetic algorithms and neural networks for solving the VRPTW. For a survey on metaheuristics applied to the vehicle routing problem the reader is referred to Gendreau et al. [14], and Thangiah et al. [37]. A recent bibliography with more than 1400 references can be found in Osman and Laporte [25].

This paper introduces a new parallel tabu search heuristic implemented on a distributed memory multiprocessor system with 8 processors. The reason for applying a parallel algorithm to the solution of the VRPTW is to accelerate the tabu search process. This is important especially in practical settings, where additional demands have to be scheduled on-line, or for the case that the problem instance consists of several hundred customers. Parallelism can be exploited in three different ways (Toulouse et al. [38]). In the first approach, the neighborhood

of the current solution is divided among the processors. Each processor explores its part of the neighborhood and sends the modifications necessary to obtain the best neighboring solution to a special processor, called master processor. The master selects from all received modifications the one that leads to the best solution of the whole neighborhood. This modification is broadcasted to all slaves and the whole process is repeated. Garcia et al. [10] showed how this approach can be applied to the VRPTW. The disadvantage of the master/slave approach is that it requires a high degree of synchronization. In the second case, communication overhead is reduced by decomposing the problem into independent subproblems, each being solved by a single processor (see, e.g., Taillard [35]). Synchronization only occurs when the subproblems are merged by the master processor to obtain a new initial solution. In the third approach several independent search threads are performed in parallel. The threads are started with different initial solutions or use a different set of search parameters. Periodically, information is shared between the threads to improve the performance of the parallel heuristic (see, e.g., Badeau et al. [2], Rego and Roucairol [31]).

The outline of this paper is as follows: In §2 a new neighborhood structure for the VRPTW is introduced. §3 describes our tabu search algorithm that traces several independent search threads in parallel. A novel feature of this algorithm is that periodically new solutions are generated by solving a set covering problem. §4 presents computational results for Solomon's benchmark problems and, finally, §5 provides some concluding remarks.

2. The neighborhood structure

Most iterative improvement algorithms for the VRPTW use an edge exchange heuristic to obtain a neighboring solution. Two classical exchange heuristics originally described for the traveling salesman problem are k-opt [23] and Or-opt [26]. In contrast to the VRPTW the traveling salesman problem consists of only a single route. The k-opt heuristic replaces a set of k links in this route by another set of k links. The complexity of the heuristic crucially depends on the size of k. For larger k the heuristic becomes more powerful, but also the computational effort increases. Lin and Kernighan [24] developed a heuristic that dynamically determines a value for k, but in most algorithms a fixed value is chosen (typically, k is set to 2 or 3). If time windows are present in the traveling salesman problem, the k-opt exchange heuristic is likely to produce infeasible solutions, since it does not preserve the orientation of the route. The Or-opt exchange, as originally described by Or [26], removes a chain of at most three consecutive customers from the route and tries to insert this chain at all feasible locations. Here, the orientation of a route is preserved. Both heuristics can be adapted for the VRPTW (multiple routes). The 2-opt* heuristic proposed by Potvin and Rousseau [29] considers two routes and removes an edge (x_i, x_{i+1})

from the first route and an edge (y_j, y_{j+1}) from the second. Then, both routes are merged by inserting edges (x_i, y_{j+1}) and (y_j, x_{i+1}). In this way, the orientation of both routes is preserved. The Or-opt heuristic is slightly modified by allowing the chain to be inserted in the same route (intra-route exchange) and in other routes (inter-route exchange). A more detailed description of edge exchange heuristics and their efficient implementation can be found in [18]. Recently, Taillard et al. [36] proposed a CROSS exchange that generalizes 2-opt* and Or-opt exchanges.

In our heuristic a neighboring solution is generated by performing a sequence of simple customer shifts. Each shift moves a customer from one route to another. The first customer shift gives a certain gain $G > 0$, but may lead to an infeasible solution. The following customer shifts are used to reestablish feasibility, but may cause a reduction of the original gain G. Thus, a neighboring solution is obtained by successively performing *all* shifts of the sequence starting with the first one. In contrast to the exchange heuristics described above, three or more routes may be involved in the exchange process. In the following, we give a more formal definition of the VRPTW and the objective function. The notation will be used in the algorithmic description of our heuristic.

2.1. Formal definitions

Let $N = \{1, \ldots, n\}$ denote the set of customers, d_x the demand of customer $x \in N$, s_x the service time at customer x (time needed to unload all goods), and $t_{x,y}$ the distance in time units (travel time) between customers x and y. If no time window constraints are present, the cost or length of a route $R_i = (x_0, x_1, \ldots, x_{m(i)}, x_{m(i)+1})$, $x_0 = x_{m(i)+1} = 0$ (0 denotes the depot), is given by

$$C_{\text{VRP}}(R_i) = \sum_{p=0}^{m(i)} t_{x_p, x_{p+1}} + \sum_{p=1}^{m(i)} s_{x_p}.$$

Route R_i is feasible if the total demand of all customers supplied on R_i does not exceed the vehicle capacity D (capacitated VRP) and if the total duration of the route does not exceed a prespecified bound T (time constrained VRP), that is $\sum_{p=1}^{m(i)} d_{x_p} \leq D$ and $C_{\text{VRP}}(R_i) \leq T$. D and T are equal for all vehicles and all routes, respectively.

When considering the vehicle routing problem with time window constraints, an interval $[e_x, l_x]$ is associated with each customer $x \in N$. The interval $[e_0, l_0]$ at the depot is called the scheduling horizon. Let b_x denote the beginning of service at customer x. Now, for a route $R_i = (x_0, x_1, \ldots, x_{m(i)}, x_{m(i)+1})$ to be feasible it must additionally hold $e_{x_p} \leq b_{x_p} \leq l_{x_p}$, $1 \leq p \leq m(i)$, and $b_{x_{m(i)}} + s_{x_{m(i)}} + t_{x_{m(i)},0} \leq l_0$. Provided that a vehicle travels to the next customer as soon as it has finished service at the current customer, b_{x_p} can be recursively computed as $b_{x_p} = \max\{e_{x_p}, b_{x_{p-1}} + s_{x_{p-1}} + t_{x_{p-1},x_p}\}$ with $b_0 = e_0$ and $s_0 = 0$. Thus, a waiting

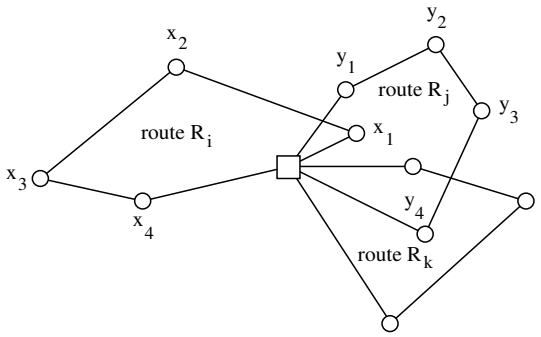

Figure 1.

time $w_{x_p} = \max\{0, b_{x_p} - b_{x_{p-1}} - s_{x_{p-1}} - t_{x_{p-1},x_p}\}$ may be induced at customer x_p. The cost of route R_i is now given by

$$C_{\text{VRPTW}}(R_i) = \sum_{p=0}^{m(i)} t_{x_p,x_{p+1}} + \sum_{p=1}^{m(i)} s_{x_p} + \sum_{p=1}^{m(i)} w_{x_p}.$$

For a solution S with routes $R_1, \ldots, R_s$ the cost of S is given by

$$F_{\text{VRPTW}}(S) = \sum_{i=1}^{s} (C_{\text{VRPTW}}(R_i) + M),$$

where M is a large constant. M is added, because minimization of the fleet size is considered to be the primary objective of the VRPTW. S is said to be feasible if all routes belonging to S are feasible and each customer is served by exactly one route.

As described by Solomon [34], we assume that initially all vehicles leave the depot at the earliest possible time e_0. Having obtained a solution of the VRPTW, we adjust the depot departure time of each vehicle to eliminate any unnecessary waiting time.

2.2. Construction of a shift sequence

Many powerful VRP algorithms allow to consider infeasible solutions during the iterative improvement process, since this helps to escape from local optima (see, e.g., Gendreau et al. [13]). Unfortunately, for the VRPTW with strict time window constraints it is much more difficult to obtain a feasible solution from an infeasible one. Therefore, in our heuristic, infeasibility is limited to interim-solutions found within the shift sequence. After applying the whole shift sequence – and, thus, after one round of the algorithm – a feasible solution is obtained again.

The computation of a shift sequence of length two can be demonstrated by using the example in Figure 1. Let us assume that it is cheaper to supply

customer x_1 on route R_j. When shifting customer x_1 to route R_j, time window restrictions on customer y_4 will be violated. The algorithm now checks for each customer y_r, $1 \leq r \leq 4$, whether the removal of y_r reestablishes the feasibility of route R_j. If this is the case, route R_i and route R_k will be searched for the best location to insert y_r. As the length of the shift sequence is limited to two, the insertion of y_r must maintain the feasibility of the corresponding route. From all shift sequences generated this way the one with highest gain G is saved.

The above description directly leads to the following recursive function for the construction of a shift sequence. Parameters of the function are: Customer x for which a new location to insert is searched, gain G, and length L of the already determined part of the shift sequence. Important global variables are G_{best} which holds the gain obtained by the best shift sequence found in the neighborhood of the current solution and $L_{\max}$ which limits the maximal length of a shift sequence.

Algorithm 1 SHIFT(x, G, L).

Step 1 Let R_i denote the route on which customer x is currently being supplied. Remove customer x from R_i and set $G := G + g$, where g denotes the gain resulting from the removal.

Step 2 For each route $R_j = (y_0, y_1, \ldots, y_{m(j)}, y_{m(j)+1})$, $y_0 = y_{m(j)+1} = 0$, $i \neq j$, and for each pair of consecutive customers (y_q, y_{q+1}) in R_j, compute the extension d of R_j when x is inserted between y_q and y_{q+1}. If $G - d$ is greater than the gain G_{best} obtained by the best shift sequence found so far, then

Step 2.1 Insert x between y_q and y_{q+1} and set $G := G - d$. Append the shift of x at the end of the shift sequence and set $L := L + 1$.

Step 2.2 If route R_j remains feasible, save the current shift sequence as the best shift sequence found so far and update G_{best}.

Step 2.3 If route R_j is infeasible and $L + 1 \leq L_{\max}$, check for each customer y_r in route R_j whether the removal of y_r reestablishes feasibilitiy. If this is the case, call SHIFT(y_r, G, L).

Step 2.4 Remove x from route R_j and set $G := G + d$. Remove the shift of customer x from the end of the shift sequence and set $L := L - 1$.

Step 3 Re-insert x in route R_i and set $G := G - g$.

Initially, SHIFT is called with $G = 0, L = 0$ and $G_{\text{best}} = -\infty$. Step 2 describes the main loop of the function. The construction of the shift sequence is stopped as soon as its gain drops below the gain of the best sequence found so far. This strategy guarantees a good performance of the heuristic and follows the idea to allow infeasibility as long as we have a high gain in return. In Step 2.3, SHIFT is recursively called for each customer y_r whose removal reestablishes feasibility. After completion of the last recursive call (or if feasibility cannot be reestablished by the removal of any y_r), the function continues with Step 2.4.

2.3. Implementation details

In this section we introduce some techniques to speed up the construction of a shift sequence. A simple and well-known technique is to reduce the size of the neighborhood by discarding any customer shifts that are unlikely to yield any improvements. For this, we only try to insert a customer between its 60 nearest neighbors. Due to the bounding via G_{best}, this improves the performance of the algorithm only slightly. A better speed-up is achieved by using the techniques described below.

Each time a customer x is inserted in a route, the service begin of all customers scheduled behind x must be updated to check the feasibility. Furthermore, the original route-extension may be reduced by waiting times on these customers. A similar situation occurs when x is removed from a route. Now the original gain can be reduced by new generated waiting times. In the following we show how a shift can be evaluated in constant time without physically moving the customer from one route to another. Again, let $R_i = (x_0, x_1, \ldots, x_{m(i)}, x_{m(i)+1})$, $x_0 = x_{m(i)+1} = 0$, be a route of the VRPTW. For a customer x_p, $1 \leq p \leq m(i)$, let $\text{bwd}(x_p)$ denote the maximal number of time units for which the service at x_p can be pushed backwards in the schedule without generating a waiting time at x_p or at a customer x_q, $p < q \leq m(i)$. The value $\text{bwd}(x_p)$ can be computed by the following recursive function:

$$\begin{aligned} \text{bwd}(x_p) &= \min\{b_{x_p} - e_{x_p}, \text{bwd}(x_{p+1})\}, \quad p < m(i), \\ \text{bwd}(x_{m(i)}) &= b_{x_{m(i)}} - e_{x_{m(i)}}. \end{aligned}$$

A value of zero indicates that there is a waiting time at x_p or at a customer x_q supplied after x_p in R_i. Provided that the bwd-value is known for all customers on R_i, the gain g resulting from the removal of any customer x_p, $p < m(i)$, is given by $g = \min\{b_{x_{p+1}} - b^{\text{new}}_{x_{p+1}}, \text{bwd}(x_{p+1})\}$. Here, $b^{\text{new}}_{x_{p+1}}$ denotes the service begin at customer x_{p+1} after the removal of x_p. More formally, $b^{\text{new}}_{x_{p+1}} = \max\{e_{x_{p+1}}, b_{x_{p-1}} + s_{x_{p-1}} + t_{x_{p-1},x_{p+1}}\}$.

Thus, the gain obtained in Step 1 of function SHIFT can be computed in constant time without actually removing customer x from route R_i. In Step 2 the extension d of route R_j resulting from the insertion of x between y_q and y_{q+1} is given by $d = \max\{0, b^{\text{new}}_{y_{q+1}} - b_{y_{q+1}} - W_{R_j,y_{q+1}}\}$, where $W_{R_j,y_{q+1}}$ denotes the sum of waiting times at customers $y_{q+1}, \ldots, y_{m(j)}$. Because pairs of consecutive customers are considered in ascending order $(y_0, y_1), (y_1, y_2), \ldots, (y_{m(j)}, y_{m(j)+1})$, $W_{R_j,y_{q+1}}$ can be computed by setting $W_{R_j,y_{q+1}} = W_{R_j,y_q} - w_{y_q}$. W_{R_j,y_0} is initialized with the total waiting time on route R_j, i.e. $W_{R_j,y_0} = \sum_{r=1}^{m(j)} w_{y_r}$. Each time a customer is removed or inserted the total waiting time on a route can be updated without any additional effort. Therefore, the route-extension d can be computed in constant time.

Also, the feasibility check in Step 2.2 can be carried out in constant time. For this, let $\text{lb}(y_q)$ denote the latest possible service begin at customer y_q so that

the time window constraints of all customers y_r, $q \leq r \leq m(j)+1$, (this includes the depot) are met. The value $\mathrm{lb}(y_q)$ can be computed by the following recursive function:

$$\begin{aligned} \mathrm{lb}(y_q) &= \min\{l_{y_q}, \mathrm{lb}(y_{q+1}) - s_{y_q} - t_{y_q,y_{q+1}}\}, \quad q < m(j), \\ \mathrm{lb}(y_{m(j)}) &= \min\{l_{y_{m(j)}}, l_0 - s_{y_{m(j)}} - t_{y_{m(j)},0}\}. \end{aligned}$$

Then, route R_j remains feasible after a potential insertion of x between y_q and y_{q+1}, if $b^{\mathrm{new}}_{y_{q+1}} \leq \mathrm{lb}(y_{q+1})$. Especially, in the last recursion level ($L = L_{\max}$) no physical customer shift will be necessary, if lb-values are available.

To summarize, the number of physical customer shifts can be reduced significantly when bwd- and lb-values have been computed in advance. Only when entering the next recursion level the corresponding customer must be physically moved from R_i to R_j. Additionally, all bwd- and lb-values have to be updated in both routes. Asymptotically, this update does not impose any overhead. To avoid a re-computation of the values after returning from recursion, bwd- and lb-values are stored for each level.

3. The parallel tabu search algorithm

Our parallel tabu search algorithm performs several search threads in parallel. Each thread is started with a different initial solution and tries to improve it by applying a local optimization process encapsulated in a tabu search metaheuristic. At termination of these local optimization steps the worst solution is replaced by a new one. With this new set of solutions the whole process is restarted until a certain stopping criterion is fulfilled. The basic framework of our algorithm can be summarized as follows:

Algorithm 2 Basic framework of the parallel algorithm.

Step 1 Generate k initial solutions $S_1, \ldots, S_k$.
Step 2 While the stopping criterion $Stop_1$ is not met do

- **Step 2.1** While the stopping criterion $Stop_2$ is not met do
  For each i, $1 \leq i \leq k$, set $S_i :=$ OptimizeVRP(S_i).
- **Step 2.2** Generate a new solution $\tilde{S}$ from $S_1, \ldots, S_k$. and replace the worst solution by $\tilde{S}$.

Step 3 Output the best solution found.

In Step 2.1 a local optimization process is applied to the solutions $S_1, \ldots, S_k$. In each iteration of this process OptimizeVRP(S_i) returns the best (non tabu) solution found in the neighborhood of S_i. §3.1 gives a detailed description of the main components of this function and §3.2 explains how Step 2.1 can be embedded in tabu search. Based on the optimized solutions $S_1, \ldots, S_k$ a new

solution $\tilde{S}$ is generated during Step 2.2. This enables a *sharing of knowledge* among the independent search threads. As described in §3.3 our algorithm uses a set covering heuristic to construct $\tilde{S}$. Finally, §3.4 describes the steps of our parallel tabu search algorithm.

3.1. Components of the local optimization process

Function OPTIMIZEVRP consists of three components. First, a route elimination heuristic (called ROUTEELIMINATION) is performed, because the minimization of the vehicle fleet is given highest priority. This heuristic considers all routes containing at most three customers, and tries to move these customers into other routes while maintaining the feasibility of the solution. If the heuristic detects more than one feasible location to insert a customer, the best one will be chosen (cf. Garcia et al. [10]). Then, function SHIFT is executed to determine a new assignment of customers to routes. Finally, the routing of customers supplied by the same vehicle is improved by performing an intra-route edge exchange heuristic. In our algorithm we use a modified version of the Or-opt heuristic. As originally described by Or [26], the heuristic considers a chain of at most three consecutive customers, removes this chain, and searches all other possible locations within the same route for the best feasible insertion place. The reallocation of chains is iterated until no further improvement can be achieved. The heuristic has been adapted to account for time window constraints as described by Savelsbergh [33]. The complete function can now be formulated as follows:

Algorithm 3 OPTIMIZEVRP(S).

Step 1 (ROUTEELIMINATION) For each route in S containing at most three customers try to move these customers to other routes in S. If there are several feasible locations to insert a customer, choose the one with minimal cost.

Step 2 (SHIFT) Set $G_{\text{best}} := -\infty$. Compute for each customer x in S the shift sequence with highest gain G by calling SHIFT$(x, 0.0, 0)$. Let x^* denote the customer whose shift sequence results in the maximal gain. Generate a new assignment of customers to routes in S by performing the shift sequence associated with x^*.

Step 3 (OR-OPT) Apply the adapted Or-opt heuristic to every route in S modified during Step 1 or Step 2 and return the updated solution S.

3.2. Tabu search

Function OPTIMIZEVRP is embedded in a tabu search process that follows the guidelines proposed by Glover [15,16] and Glover et al. [17]. The general strategy of tabu search is to explore all possible moves from the current to a neighboring solution. The move leading to the best neighboring solution is accepted, even if this results in a deterioration of the objective function. To prevent

the search process from cycling, a tabu list is used that stores the inverse move for a certain number of iterations. Thus, all solutions that can be obtained by applying a move stored in the tabu list are excluded from the search. An aspiration criterion allows to override the tabu status of a move, for example, if the move leads to a new best solution.

In our algorithm the tabu list consists of a 2-dimensional array. For a solution S with routes $R_1, \ldots, R_s$ Tabu(x, j) records the tabu status of customer x with respect to route R_j, $1 \leq j \leq s$. For example, if Tabu$(x, j) = m$, then customer x is not allowed to be inserted in route R_j until iteration m. In Step 2.2 of function SHIFT a sequence will be discarded, if it contains only tabu moves, except the sequence does not only lead to an improvement of G_{best}, but also to a new best solution (aspiration). If customer x is removed from route R_j at iteration m, then Tabu(x, j) is set to $m + (5 + \rho)$, where ρ is a random value uniformly chosen in the interval $[0, 9]$.

Besides the tabu list management, diversification and intensification techniques are fundamental components of tabu search algorithms. Diversification drives the search to explore new regions of the solution space, whereas intensification focuses the search on regions around good solutions. In our algorithm dynamic diversification can be achieved by penalizing customer shifts that are performed frequently. For this, we use a penalty-function originally proposed by Taillard et al. [36]. Let $fr_{(x,j)}$ denote the number of insertions of customer x in route R_j and let ϱ be a random value uniformly chosen in the interval $]0.0, 0.5]$. Furthermore, let $fr_{\max} = \max\{fr_{(x,j)};\ x \in N, 1 \leq j \leq s\}$. If $\Delta_{\max}$ is the maximal absolute difference between the objective values of two consecutive solutions encountered during the search process, then the insertion of x in R_j will be penalized by

$$p_{(x,j)} = \varrho \cdot \Delta_{\max} \cdot \frac{fr_{(x,j)}}{fr_{\max}}.$$

Here, $fr_{\max}$ normalizes the frequencies according to the neighborhood size and $\Delta_{\max}$ adjusts the penalty with respect to the maximal change of the objective function. Functions ROUTEELIMINATION and OR-OPT are used to intensify the search in the region around the current solution.

3.3. Using a set covering heuristic for the construction of new solutions

During each execution of OPTIMIZEVRP all generated routes R are collected in a pool Ω. The pool provides a large variety of feasible routes which might be used for constructing a new initial solution. The idea of recombining the routes of already found solutions has originally been proposed by Rochat and Taillard [30]. In their approach pairwise disjoint routes (i.e., routes that have no customer in common) are randomly selected from Ω giving a higher probability to routes belonging to good solutions. Another possibility to obtain a subset Ω' of Ω so

that each customer belongs to exactly one route in Ω' is to solve a set partitioning problem (SPP). For this, let $w = |\Omega|$, and let $A = (a_{ij})$ denote a 0-1 matrix of size $n \times w$. In A each column j is associated with a route R_j from Ω in the following way: $a_{ij} = 1$, if customer i is supplied on route R_j and 0 otherwise. The cost λ_j of route R_j is set to $C_{\text{VRPTW}}(R_j) + M$. Again, M is added to give highest priority to the minimization of the fleet size. Now, the solution of the linear program

$$\begin{aligned} \text{Min} \quad & \sum_{j=1}^{w} \lambda_j x_j \\ \text{s.t.} \quad & A \cdot x = 1 \\ & x = (x_1, \ldots, x_w)^T \in \{0,1\}^w \end{aligned}$$

provides an optimal solution of a restricted VRPTW in which the set of feasible routes is limited to those in Ω. Several exact algorithms for the VRP and for the VRPTW are based on the set partitioning formulation given above (see, e.g., Desrochers et al. [6]). In these algorithms a column generation scheme is used to gradually enlarge Ω.

Instead of the SPP, we actually solve the corresponding set covering problem (SCP). SPP differs from SCP only in the first constraint, where $Ax = 1$ is replaced by $Ax \geq 1$. The SCP is less restrictive, because it allows to combine routes even if this results in serving a few customers on more than one route. In fact, during our computational experiments we observed that often routes belonging to good solutions have one or two customers in common. From such a route-overlapping solution we can easily construct a feasible one by deleting customers from all but one route. Unfortunately, SPC as well as SPP is a NP-hard problem and so there is little hope to obtain an exact solution in short time. Therefore, we use the Lagrangian heuristic proposed by Beasley [3]. For an introduction to Lagrangian relaxation with applications to the SCP we refer to Beasley [4].

3.4. Building the parallel algorithm

A natural way to parallelize the basic algorithm is a master/slave approach, in which the master processor holds the pool Ω. The initial solutions $S_1, \ldots, S_k$ are evenly distributed among the remaining processors (slaves) of the parallel machine. Each slave performs the tabu search heuristic and returns the best solution found to the master. This scheme has a low communication overhead but one main disadvantage: Before starting the next iteration the master has to apply the SCP-heuristic to the routes in Ω leaving all slaves idle.

To avoid idle times Ω is distributed among the processors $1, \ldots, P$. Each processor p, $1 \leq p \leq P$, starts the SCP-heuristic on its own small part Ω^p of Ω and thus no master process is needed anymore. To enable an efficient information sharing between the independent search threads Ω^p is splitted into a global part Ω^p_{global} and a local part Ω^p_{local}. Each time a processor p terminates its local optimization process the routes of the optimized solutions are sent to all other

processors $p' \neq p$ (Broadcast). When a processor receives new routes they are stored in the global part of the pool. All routes generated during the optimization process are stored in the local part of the pool and no other processor is granted access to this part. Thus, the global part collects all best solutions found by any processor $p' \neq p$, whereas the local part contains all routes generated by p during the local optimization process. As a consequence, the local part grows much faster and routes are overwritten in a cyclic fashion. Also, the quality of the routes stored in the two parts can differ significantly. Because tabu search allows a deterioration of the objective function, some routes stored in the local part may belong to bad solutions. During our computational experiments we observed that these routes are sometimes very helpful for obtaining a good solution of the SCP. We are now ready to formulate our parallel algorithm.

Algorithm 4 PARTS(p).

Step 1 Read initial solutions $S_1^p, \ldots, S_{k_p}^p$ from file. Set $\Omega_{\text{global}}^p := \emptyset$ and store routes of the initial solutions in Ω_{local}^p. Let $S_{\text{best}}^p \in \{S_1^p, \ldots, S_{k_p}^p\}$ denote the best solution found on this processor.

Step 2 While stopping criterion $Stop_1$ is not met do

Step 2.1 While stopping criterion $Stop_2$ is not met do
For each i, $1 \leq i \leq k_p$, set $S_i^p :=$ OPTIMIZEVRP(S_i^p) and update S_{best}^p if necessary. Store all modified routes in Ω_{local}^p using a cyclic (wrap around) storage scheme.

Step 2.2 Send the routes of optimized solutions $S_1^p, \ldots, S_{k_p}^p$ to all processors $p' \neq p$ (Broadcast).

Step 2.3 Update Ω_{global}^p with all received routes.

Step 2.4 Apply the Lagrangian SCP-heuristic to the routes in $\Omega^p = \Omega_{\text{global}}^p \cup \Omega_{\text{local}}^p$. Remove customers from overlapping routes to generate a new feasible solution $\tilde{S}^p$.

Step 2.5 Let i^* denote the index of the worst solution. Set $S_{i^*}^p := \tilde{S}^p$ and update S_{best}^p if necessary.

Step 3 If stopping criterion $Stop_1 \neq$ *termination-signal*, send *termination-signal* to all other processors.

Step 4 If $p \neq 1$, send S_{best}^p to processor 1.

Step 5 If $p = 1$, select best solution $S_{\text{best}} \in \{S_{\text{best}}^1, \ldots, S_{\text{best}}^P\}$ and output it.

Function PARTS(p) is executed on every processor p, $1 \leq p \leq P$. An important parameter of the function is the number k_p of search threads performed on processor p. In general, k_p is the same for all processors and with increasing P this number decreases. On heterogeneous platforms different values for k_p can be used to account for processors with different efficiencies. In §4 we present com-

putational results for $P = 1\,(2, 4, 8)$ processors each performing $k_p = 8\,(4, 2, 1)$ search threads, respectively. Our results demonstrate that – while preserving solution quality – the tabu search process can be accelerated significantly when 8 processors are used each executing only one search thread.

In our implementation two additional functions SEND(p) and RECEIVE(p) are executed at the same time on processor p. All routes to be sent during Step 2.2 are delivered to SEND(p). Now SEND(p) is responsible for distributing the routes among all other processors. RECEIVE(p) collects all routes sent by other processors and stores them in a buffer. During Step 2.3 this buffer is emptied and all routes are appended to Ω^p_{global}. Any processor that fulfills stopping criterion $Stop_1$ can initiate the termination of PARTS by sending a signal. As soon as a processor receives this termination-signal it interrupts the local optimization process and sends the best solution found to processor 1. Processor 1 selects the overall best solution and outputs it.

To generate an appropiate number of initial solutions we use Solomon's I1 insert heuristic [34], Potvin and Rousseau's parallel route building heuristic [28], and a modified savings heuristic adapted for handling time window constraints [34]. In a sequential preprocessing step each heuristic is called several times with different parameter settings. The savings heuristic is used to account for problem instances with wide time windows. Here, the geographical aspects of the routing problem dominate the temporal aspects. Additionally, there is a better chance to obtain different initial solutions when using algorithms with different route building strategies.

4. Computational results

In this section we present computational results for six problem sets introduced by Solomon [34]. Each set contains several problem instances consisting of 100 customers. The geographical data for sets R1 (12 instances) and R2 (11 instances) were randomly generated according to a uniform distribution. Sets C1 (9 instances) and C2 (8 instances) contain clustered problems, and sets RC1, RC2 (8 instances, respectively) were composed of mixed problems with randomly distributed and clustered customers. The travel time between two customers corresponds to the Euclidean distance. Additionally, a fixed service time is given for each customer. The service time is set at 90 time units for problems in C1, C2 and at 10 time units for all other problems. R1, C1, and RC1 have a narrow scheduling horizon allowing only a few customers to be served on the same route. Conversely, in all problem sets of type 2 (i.e., R2, C2, and RC2) the depot has a wide time window so that many customers can be supplied on the same route.

All tests were performed on a Parsytec CC system with 8 nodes. Each node consists of a Motorola PowerPC 604 (133MHz) with 64MB memory (further technical information can be found at *http://www.uni-paderborn.de/pc2/*). For

our experiments we used the following parameter settings:

(1) Stopping criterion $Stop_1$ is met when 20 iterations of the outer while loop have been performed. Additionally, $Stop_1$ is met, when several processors have found a solution whose cost is equal to the cost of the overall best solution $S_{\text{best}} \in \{S^1_{\text{best}}, \ldots, S^P_{\text{best}}\}$ and the processors fail to improve it. To detect such situations, each time a new solution S is stored in Ω^p_{global} processor p compares $F_{\text{VRPTW}}(S)$ with $F_{\text{VRPTW}}(S_{\text{best}})$. If both values are equal, a counter will be incremented. $Stop_1$ is set to true, as soon as the counter exceeds $3 \cdot P$. If $F_{\text{VRPTW}}(S)$ is lower than $F_{\text{VRPTW}}(S_{\text{best}})$, processor p updates S_{best} and sets the counter back to 0. Furthermore, the loop is interrupted when a termination-signal has been received.

(2) Stopping criterion $Stop_2$ is met when 100 iterations of the inner while loop have been performed or when OPTIMIZEVRP has failed to improve any solution in $\{S^p_1, \ldots, S^p_{k_p}\}$ during 70 consecutive iterations. Additionally, the loop is interrupted when a termination-signal has been received.

(3) The maximal length of a shift sequence is limited to 2 ($L_{\max} = 2$).

(4) On each processor maximal 500 routes are stored in the local part of the pool. If more routes are generated during the local optimization process, routes will be overwritten in a cyclic fashion.

We compared our tabu search heuristic with some of the most powerful VRPTW heuristics described in literature. In all algorithms the minimization of the fleet size is considered to be the primary objective. However, some algorithms minimize travel time (distance traveled measured in time units) and others the sum of travel time and waiting time. Furthermore, the algorithms differ with regard to rounding of distances and/or final solution values. The following list provides information concerning author, applied metaheuristic, secondary objective (minimization of travel time or minimization of travel and waiting time), and rounding:

Bachem et al. [1], simulated trading, travel and waiting time, no rounding
Chiang, Russell [5], simulated annealing, travel and waiting time, no rounding
Potvin et al. [27], tabu search, travel and waiting time, rounding of final solution
Rochat, Taillard [30], tabu search, travel time, no rounding
Taillard et al. [36], tabu search travel time, no rounding
Thangiah et al. [37], genetic algorithms, travel time, rounding of final solution

In a more practical setting the two objectives can be interpreted as follows: In the first objective costs associated with a route are mainly costs for fuel, whereas the second objective additionally considers the wages for the truck drivers. When minimizing the sum of travel time and waiting time, we actually try to minimize

Table 1
Best averages published for problems of type 1

	R1		C1		RC1	
	routes	length	routes	length	routes	length
Chiang, Russell	12.42	1289.95	10.00	885.86	12.38	1455.82
Rochat, Taillard	12.25	1208.50	10.00	828.38	11.88	1377.39
Taillard et al.	12.17	1209.35	10.00	828.38	11.50	1389.22
Thangiah et al.	12.33	1238.00	10.00	832.00	12.00	1284.00
Our method	12.25	1239.15	10.00	828.94	11.75	1409.26
Bachem et al.	12.58	1392.00	–	–	12.13	1501.60
Potvin et al.	12.50	1390.70	10.00	850.20	12.60	1507.60
Our method	12.25	1260.67	10.00	828.94	11.75	1413.49

the working time of a truck driver. This is important, for example, when any expensive overtime is to be avoided.

Tables 1 and 2 compare the averages of the best solutions produced by our algorithm with those produced by the other methods listed above. For each problem set the first column reports the average number of vehicles and the second the average length of the obtained solutions. Both tables are divided in two parts. In the upper part the length of a route is measured in terms of the distance traveled (travel time). In the lower part the length of a route is computed as the sum of travel time and waiting time. In order to include the service time at each customer, a value of 9000 must be added for problems in sets C1 and C2, and a value of 1000 for problems in R1, R2, RC1, and RC2. The best results reported for Chiang and Russell do not include any waiting time (although the objective considers minimization of travel time and waiting time) and are cited from Taillard et al. The results reported for our algorithm in the upper part of each table are produced by simply subtracting all waiting times. However, it should be noticed that minimization of travel time and minimization of travel time and waiting time are two different objectives. In fact, it is very easy to construct a problem instance with solutions $S_1 \neq S_2$ so that the total distance traveled is shorter in S_1 while the total time needed to supply all customers is shorter in S_2.

Tables 1 and 2 show that the averages of the best solutions produced by our algorithm are better than those produced by Bachem et al., Chiang and Russell, and Potvin et al. A comparison with the other algorithms is somewhat difficult since different objectives have been used. Fortunately, in all algorithms the minimization of the fleet size is considered to be the primary objective. Only the method proposed by Taillard et al. was able to produce solutions with less routes (see, e.g., R1, RC1).

Tables 3 and 4 demonstrate how the quality of a solution is related to the computational effort. To compare our parallel algorithm with the sequential

Table 2
Best averages published for problems of type 2

	R2		C2		RC2	
	routes	length	routes	length	routes	length
Chiang, Russell	2.91	1135.14	3.00	658.88	3.38	1361.14
Rochat, Taillard	2.91	961.72	3.00	589.86	3.38	1119.59
Taillard et al.	2.82	980.27	3.00	589.86	3.38	1117.44
Thangiah et al.	3.00	1005.00	3.00	650.00	3.38	1229.00
Our method	2.82	1066.68	3.00	589.93	3.38	1286.05
Bachem et al.	3.00	1100.60	–	–	3.38	1500.10
Potvin et al.	3.10	1185.70	3.00	594.60	3.40	1459.80
Our method	2.82	1070.78	3.00	589.93	3.38	1295.89

Table 3
Solution quality versus CPU time – problems of type 1

	R1		C1		RC1	
	time	routes, length	time	routes, length	time	routes, length
Rochat, Taillard	450	12.83, 1208.43	540	10.00, 832.59	430	12.75, 1381.33
	1300	12.58, 1202.31	1600	10.00, 829.01	1300	12.50, 1368.03
	2700	12.58, 1197.42	3200	10.00, 828.45	2600	12.38, 1369.48
Taillard et al.	2296	12.64, 1233.88	2926	10.00, 830.41	1877	12.08, 1404.59
	6887	12.39, 1230.48	7315	10.00, 828.59	5632	12.00, 1387.01
	13774	12.33, 1220.35	14630	10.00, 828.45	11264	11.90, 1381.31
Bachem et al.	611	12.75, 1417.00	–	–	570	12.50, 1532.00
Chiang, Russell	206	12.50, 1444.90	139	10.00, 929.10	177	12.38, 1562,20
Our method	403	12.50, 1268.42	179	10.00, 828.94	358	12.25, 1396.07

Table 4
Solution quality versus CPU time – problems of type 2

	R2		C2		RC2	
	time	routes, length	time	routes, length	time	routes, length
Rochat, Taillard	1600	3.18, 999.63	1200	3.00, 595.38	1300	3.62, 1207.37
	4900	3.09, 969.29	3600	3.00, 590.32	3900	3.62, 1155.47
	9800	3.09, 954.36	7200	3.00, 590.32	7800	3.62, 1139.79
Taillard et al.	3372	3.00, 1046.56	3275	3.00, 592.75	1933	3.38, 1248.34
	10116	3.00, 1029.65	8187	3.00, 591.14	5798	3.38, 1220.28
	20232	3.00, 1013.35	16375	3.00, 590.91	11596	3.38, 1198.63
Bachem et al.	674	3.09, 1249.50	–	–	499	3.38, 1512.00
Chiang, Russell	273	2.91, 1258.25	166	3.00, 703.60	238	3.38, 1572.70
Our method	968	3.09, 1055.90	265	3.00, 589.93	734	3.38, 1308.31

heuristics described in literature we run PARTS on a single node of the CC system. During our experiments $k_1 = 8$ independent search threads have been performed on that node. For each problem set the first column reports the running time in sec. and the second the average number of vehicles, and the average length of the obtained solutions (separated by a comma). Again, in the upper part of each table the length of a route is measured in terms of the distance traveled, whereas in the lower part the length is computed as the sum of travel and waiting time. For the algorithms proposed by Rochat and Taillard, and by Taillard et al. the computational effort is rather high. Therefore, averages are presented for three different points in time.

A direct comparison of the results reported in Tables 3 and 4 is difficult, because they have been obtained on different computer platforms. Furthermore, the number of independent runs used to calculate the average solution values differs from author to author. For each author the following list provides information concerning the computer used during the computational experiments and concerning the number of independent runs used to calculate the average solution values.

Bachem et al., Sun SPARC 10/50, 2 independent runs
Chiang and Russell, 486-66 PC, number of runs not mentioned in paper
Rochat and Taillard, Silicon Graphics Indigo (100MHz), 1 run
Taillard et al., Sun SPARC 10/50, 5 independent runs
Our method, Motorola PowerPC 604 (133MHz), 5 independent runs

We have run the sequential version of our algorithm ($P = 1, k_1 = 8$) on a Silicon Graphics Indigo (100MHz) and on a Sun SPARC 10/50. We observed that our algorithm runs 6.5 times faster on a single node of the CC system as compared to the Sun SPARC 10/50 and 2.5 times faster as compared to the Silicon Graphics Indigo. Therefore, we can compare our results with those reported by Rochat and Taillard in row 2 of Tables 3 and 4, and with those reported by Taillard et al. in row 4 of Table 3 and row 5 of Table 4. As can be seen, our algorithm performs better or at least equal to the heuristic proposed by Rochat and Taillard for all problem sets except C1 (but here the results differ only slightly). When comparing our results with those reported by Taillard et al. we can observe that our method converges faster for problems in R1. On the other hand, the method proposed by Taillard et al. performs better on problem sets RC1 and R2.

It is interesting to observe that the short running heuristic proposed by Chiang and Russell provides excellent results for several problem sets. For example, Chiang and Russell report an average of only 2.91 vehicles for problem set R2. This value is not met by any other heuristic (c.f. Table 4). Conversely, the average length obtained by Chiang and Russell for C1 and C2 exceeds the length reported for other heuristics by roughly 100 time units. Altogether, we can conclude that the long running heuristics provide good results for all problem types represented by Solomon's test bench.

Table 5
Best results published for selected problems

	R102	R201	RC106	RC204
	routes, length	routes, length	routes, length	routes, length
Rochat, Taillard	17, 1486.12	4, 1281.58	12, 1384.92	3, 806.75
Taillard et al.	17, 1487.60	4, 1254.80	11, 1448.26	3, 831.69
Thangiah et al.	17, 1493.00	4, 1354.00	13, 1420.00	3, 897.00
Our method	17, 1581.00	4, 1526.98	12, 1404.68	3, 933.14
Bachem et al.	17, 1991.70	4, 1975.30	12, 1491.70	3, 1065.00
Our method	17, 1637.76	4, 1563.18	12, 1408.12	3, 935.05

Table 6
Performance of the parallel algorithm with respect to the selected problems

	R102		R201		RC106		RC204	
	time	length	time	length	time	length	time	length
$P = 1,\ k_p = 8$	375	1690.51	651	1601.71	318	1429.14	1068	949.71
$P = 2,\ k_p = 4$	213	1720.07	398	1600.95	173	1430.20	406	942.71
$P = 4,\ k_p = 2$	122	1668.85	221	1610.88	82	1429.41	299	957.65
$P = 8,\ k_p = 1$	82	1705.27	127	1613.33	45	1470.43	222	937.19

In the following we present computational results for $P = 1$ $(2, 4, 8)$ processors each performing $k_p = 8$ $(4, 2, 1)$ independent search threads, respectively. To show the benefits obtained by our parallel tabu search algorithm we have chosen one problem instance from each set and display the progress of the optimization process in Figures 2–5. Problem sets C1,C2 have been omitted, because on these well-structured problems the sequential version of our algorithm detects good local optima in very short time. Table 5 introduces the selected problem instances together with the best results reported by various authors. Again, the table is splitted in an upper and a lower part according to the secondary objective.

Table 6 reports the results obtained by a single run of our parallel tabu search algorithm. A comparison with the best results reported in Table 5 (see, e.g., Bachem et al.) indicates that high quality solutions can be obtained by our parallel algorithm for all combinations of P and k_p. Furthermore, the running time is reduced significantly when 8 processors are used each performing only one search thread. We omitted the calculation of any speed-up factors, because the asynchronous communication scheme exploited by our algorithm implicitly adds a certain degree of randomness to the search process. In general, solutions generated using P_1 processors and solutions generated using P_2 processors differ completely. The same holds for the order in which the solutions are encountered during the search process. Despite this, it is interesting to observe that the quality of the final solutions varies only slightly.

Because of termination criterion $Stop_2$ a situation may occur in which the

inner while-loop of PARTS is interrupted after 70 iterations for several consecutive iterations of the outer while-loop. If this only happens when many processors are used, it may result in a "super-linear" speed-up. For example, consider problem RC204. When using $P = 1$ processors the algorithm terminates after 1068 sec., but when using $P = 2$ processors the algorithm stops after 406 sec. Conversely, this may also reduce the original speed-up if more iterations of the inner while-loop are performed when using more processors.

Figures 2–5 show the progress of the optimization process for the selected problem instances. In each figure the quality of a solution, measured as *number of routes* $\times$ *10000* + *length*, is plotted against the time in sec. Figures 3 and 5 demonstrate that the search process converges faster to its local optima when several processors are used. This is simply because the independent search threads are traced faster. On the other hand, Figures 2 and 4 show that the minimization of the fleet size remains a big challenge even to the parallel algorithm. In the case of R102 it is interesting to observe that each time when the number of processors has been increased a solution with less routes is found earlier (except when increasing the number of processors from 2 to 4). Conversely, Figure 4 shows that this is not true for RC106. Here, the parallel algorithm with 8 processors succeeds to reduce the fleet size only a few seconds before termination. After route elimination there is not enough time left to improve the new solution. This explains the relative large gap between the solution value 1429.41 obtained for $P = 4, k_p = 2$ and 1470.43 obtained for $P = 8, k_p = 1$.

5. Conclusion

In this paper we have introduced a new neighborhood structure for the VRPTW based on sequences of simple customer shifts. Furthermore, we presented a new technique for recombining the routes of already visited solutions. Our technique uses an efficient SCP-heuristic and allows to combine routes even if this results in serving a few customers on more than one route. Our problem-solving methodology is embedded in a parallel tabu search heuristic. We proposed a completely asynchronous communication scheme that does not need any master processor coordinating the search. In this way, idle times or bottlenecks are avoided and the efficiency of the overall algorithm is improved. Computational results for our parallel algorithm show that significant speed-ups can be achieved while maintaining solution quality.

Acknowledgements

We would like to thank J. Rese and S. Tschöke for providing their implementation of Beasley's SCP-heuristic. Also, special thanks to the staff of

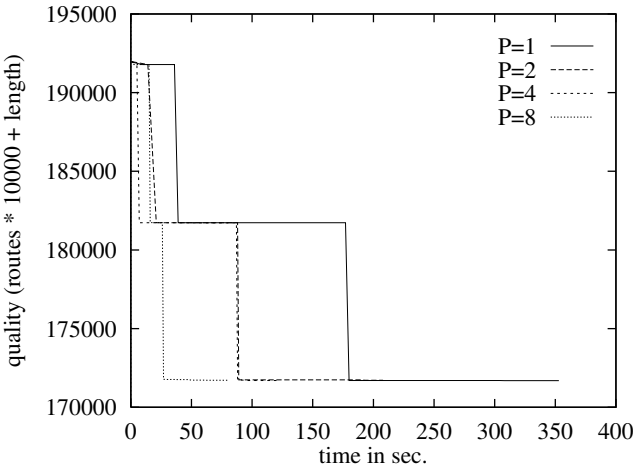

Figure 2. Progress of the optimization process concerning R102

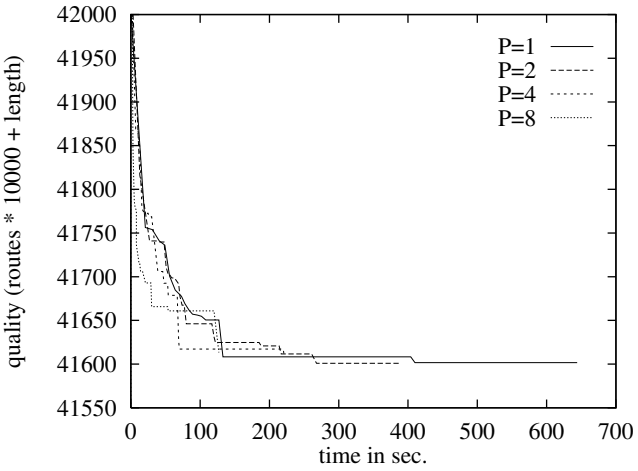

Figure 3. Progress of the optimization process concerning R201

the *Paderborn Center for Parallel Computing* (PC^2) for the support during our computational experiments.

References

[1] A. Bachem, W. Hochstättler, M. Malich, The simulated trading heuristic for solving vehicle routing problems, Discrete Applied Mathematics, 65(1–3), 47–72, 1996.

[2] P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin, E. Taillard, A parallel tabu search heuristic for the vehicle routing problem with time windows, Technical Report CRT-95-84, Université de Montréal, 1995.

[3] J.E. Beasley, A Lagrangian heuristic for set covering problems, Naval Research Logistic, 37:151–164, 1990.

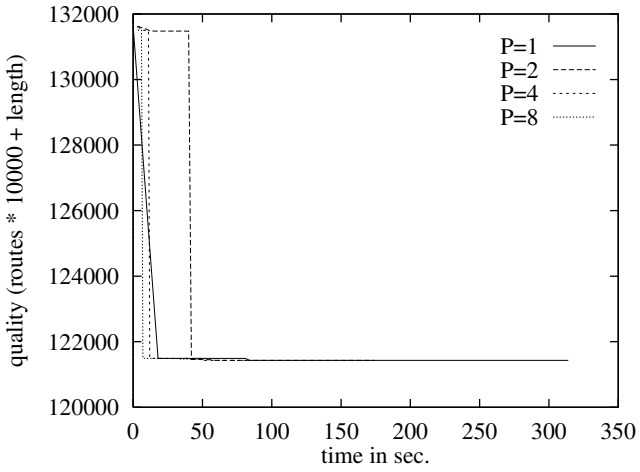

Figure 4. Progress of the optimization process concerning RC106

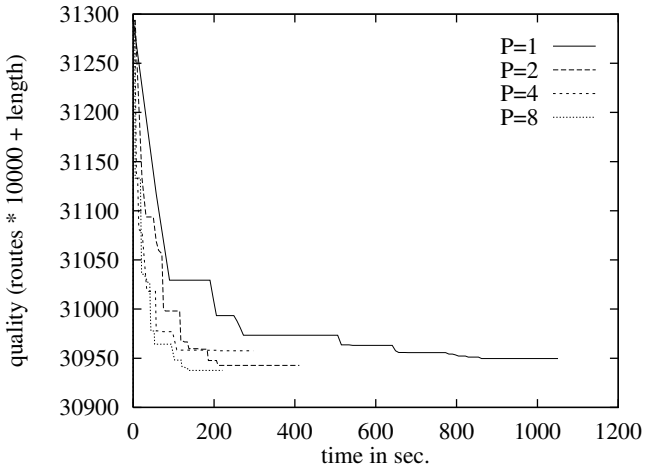

Figure 5. Progress of the optimization process concerning RC204

[4] J.E. Beasley, Lagrangean Relaxation, in *Modern heuristic techniques for combinatorial problems*, C.R. Reeves ed., John Wiley & Sons, 243–303, 1993.

[5] W.C. Chiang and R.A. Russell, Simulated annealing metaheuristics for the vehicle routing problem with time windows, Annals of Operations Research, 63, 3–27, 1996.

[6] M. Desrochers, J. Desrosiers, M.M. Solomon, A new optimization algorithm for the vehicle routing problem with time windows, Operations Research, 40, 342–354, 1992.

[7] J. Desrosiers, Y. Dumas, M.M. Solomon, F. Soumis, Time constrained routing and scheduling, in *Network Routing, Handbooks on Operations Research and Management Science 8*, M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser eds., North-Holland, Amsterdam, 35–139, 1995.

[8] M.L. Fisher, Vehicle routing, in *Network Routing, Handbooks on Operations Research and Management Science 8*, M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser eds.,

North-Holland, Amsterdam, 1–33, 1995.
[9] M.L. Fisher, K.O. Jörnsten, O.B.G. Madsen, Vehicle Routing with time windows: two optimization algorithms, Operations Research, 45, 488–492, 1997.
[10] B.-L. Garcia, J.-Y. Potvin, J.-M. Rousseau, A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints, Computers Ops. Res., 21, 1025–1033, 1994.
[11] A. Geist, A. Beguelin, J. Dongarra, W. Jaing, R. Mancheck, V. Sunderam, PVM 3.0 user's guide and reference manual, Oak Ridge National Laboratory, Tennessee, 1993.
[12] S. Gélinas, M. Desrochers, J. Desrosiers, M.M. Solomon, A new branching strategy for time constrained routing problems with application to backhauling, Annals of Operations Research, 61, 91–109, 1995.
[13] M. Gendreau, A. Hertz, G. Laporte, A tabu search heuristic for the vehicle routing problem, Management Science, 40, 1276–1290, 1994.
[14] M. Gendreau, G. Laporte, J.-Y. Potvin, Vehicle routing: modern heuristics, in *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra eds., John Wiley & Sons, Chichester, 311–336, 1997.
[15] F. Glover, Tabu search, part I, ORSA J. Computing, 1, 190–206, 1989.
[16] F. Glover, Tabu search, part II, ORSA J. Computing, 2, 4–32, 1990.
[17] F. Glover, E. Taillard, D. de Werra, A user's guide to tabu search, Annals of Operations Research, 41, 3–28, 1993.
[18] G.A.P. Kindervater, M.W.P. Savelsbergh, Vehicle routing: handling edge exchanges, in *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra eds., John Wiley & Sons, Chichester, 337–360, 1997.
[19] N. Kohl, O.B.G. Madsen, An optimization algorithm for the vehicle routing problem with time windows based on Lagrangian Relaxation, Operations Research, 45, 395–406, 1997.
[20] G. Laporte, The vehicle routing problem: an overview of exact and approximate algorithms, European J. Operational Res., 59, 345–358, 1992.
[21] G. Laporte, I.H. Osman, Routing problems: A bibliography, Annals of Operations Research, 61, 227–262, 1995.
[22] J.K. Lenstra, A.H.G. Rinnooy Kan, Complexity of vehicle routing and scheduling problems, Networks, 11, 221–227, 1981.
[23] S. Lin, Computer solutions of the traveling salesman problem, Bell Systems Technical Journal, 44, 2245–2269, 1965.
[24] S. Lin, B.W. Kernighan, An effective heuristic algorithm for the traveling salesman problem, Operations Research, 21, 498–516, 1973.
[25] I.H. Osman, G. Laporte, Metaheuristics: A bibliography, Annals of Operations Research, 63, 513–628, 1996.
[26] I. Or, Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking, Ph.D. Thesis, Dept. of Industrial Engineering and Management Sciences, Northwestern University, Evanston, 1976.
[27] J.-Y. Potvin, T. Kervahut, B.-L. Garcia, J.-M. Rousseau, The vehicle routing problem with time windows – Part 1: Tabu search, INFORMS J. Computing, 8, 158–164, 1996.
[28] J.-Y. Potvin, J.-M. Rousseau, A parallel route building algorithm for the vehicle routing and scheduling problem with time windows, European J. Operational Res., 66, 331–340, 1993.
[29] J.-Y. Potvin, J.-M. Rousseau, An exchange heuristic for routeing problems with time windows, Journal of the Operational Research Society, 46, 1433–1446, 1995.
[30] Y. Rochat, E.D. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, J. of Heuristics, Vol. 1, No. 1, 147–167, 1995.
[31] C. Rego, C. Roucairol, A parallel tabu search algorithm using ejection chains for the vehicle routing problem, in *Meta-Heuristics: Theory & Applications*, I.H. Osman, J.P. Kelly eds.,

Kluwer Academic Publishers, 661–675, 1996.

[32] R.A. Russell, Hybrid heuristics for the vehicle routing problem with time windows, Transportation Science, 29(2), 156–166, 1995.

[33] M.W.P. Savelsbergh, An efficient implementation of local search algorithms for constrained routing problems, European J. of Operational Res., 47, 75–85, 1990.

[34] M.M. Solomon, Algorithms for the vehicle routing and scheduling problem with time window constraints, Operations Research, 35, 254–265, 1987.

[35] E. Taillard, Parallel iterative search methods for vehicle routing problems, Networks, 23, 661–673, 1993.

[36] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin, A new neighborhood structure for the vehicle routing problem with time windows, Technical Report CRT-95-66, Université de Montréal, 1995.

[37] S.R. Thangiah, I.H. Osman, T. Sun, Metaheuristics for vehicle routing problems with time windows, Research Report UKC/IMS/OR94/8, Institute of Mathematics and Statistics, University of Kent, 1995.

[38] M. Toulouse, T.G. Crainic, M. Gendreau, Communication issues in designing cooperative multi-thread parallel searches, in *Meta-Heuristics: Theory & Applications*, I.H. Osman, J.P. Kelly eds., Kluwer Academic Publishers, 503–522, 1996.