A New Parallel Tour Construction Algorithm for the Vehicle Routing Problem with Time Windows

Jürgen Antes
Ulrich Derigs

Universität zu Köln
Lehrstuhl für Wirtschaftsinformatik und Operations Research
Pohligstr. 1
D 50969 Köln

Tel.: +49 221 470 5330
Internet: antes@informatik.Uni-Koeln.DE

March 31, 1995

Abstract

In this paper we present a new algorithm for the single depot vehicle routing problem with uniform vehicle capacity and single time windows per customer (*VRPTW*). The procedure is based on classical insertion principles but introduces the concept of negotiation between customers and tours leading to an algorithmic scheme which constructs and improves several tours simultaneously.

The structure of the entire procedure is modular with the several modules playing different roles with respect to the general objectives of the problem

- minimizing the number of vehicles and/or
- minimizing operating cost.

The single modules and the whole algorithmic flow are controlled through several parameters by which the logic and the intensity of the different operators can be varied and thus the quality of the output can be influenced. We report computational results showing that our procedure is a flexible and effective means for solving the *VRPTW*.

Essentially, every vehicle routing problem (*VRP*) involves the determination of a set of vehicle routes for a fleet of vehicles to service the demands of a set of customers. Motivated by concrete requests from practice numerous specifications introducing different objectives and side-constraints have been proposed in the OR-literature with a classification given by Desrochers et al. [2].

One complicating type of constraints is the introduction of so-called time windows for the service at the customers. Here the complication is twofold, the data for a problem instance involve not only spatial information about travel-distance between locations for calculating the efficiency of schedules, i.e. tour cost etc. but also information about travel-time for determining feasibility of schedules. While distance is a time-invariant measure—at least under the time horizon of the concrete planning situation—travel time may vary even over the day and thus the introduction of time windows puts a higher demand on the availability and quality of data.

Even when assuming fixed, i.e. time-invariant travel-time between any two locations, the problem has a higher complexity from an algorithmic point of view, since the combinatorial structure of the set of feasible route-combinations becomes more complicated. This statement is certainly not true under the standard model of "computational complexity". Yet, this argument stems from empirical observation showing that often the construction of feasible solutions is already hard, and the problem is not optimization, i.e. the search among an exploding number of feasible configurations, but simply the construction of a feasible schedule.

Surveys on the area of vehicle routing methods are given by Christofides [1] and Laporte [5], a survey in the field of time window constrained routing and scheduling problems is given by Solomon and Desrosiers [9]. There is also a collection of articles on vehicle routing including time window constrained problems edited by Golden and Assad [4].

In this paper we present a new algorithm for the single depot vehicle routing problem with uniform vehicle capacity and single time windows per customer (*VRPTW*). The procedure is based on classical insertion principles but introduces the concept of negotiation between customers and tours leading to an algorithmic scheme which constructs and improves several tours simultaneously.

The structure of the entire procedure is modular with the several modules playing different roles with respect to the general objectives of the problem

- minimizing the number of vehicles and/or
- minimizing operating cost.

The single modules and the whole algorithmic flow are controlled through several parameters by which the logic and the intensity of the different operators can be varied and thus the quality of the output can be influenced. We report computational results showing that our procedure is a flexible and effective means for solving the *VRPTW*.

The paper is organized as follows. In section 1 we describe the *VRPTW* more precisely by formulating a mathematical model. In section 2 we present the algorithm on a conceptual level as well as in pseudocode format. Section 3 summarizes our computational experiments on standard problems from literature.

1 *VRPTW*—Problem Definition

We now precisely formulate *VRPTW* stating a mathematical model first introduced by Solomon and Desrochers [9]. Throughout the text we assume that the problem under consideration is of the delivery type. Hence, we are given a set $C = \{1, ..., n_c\}$ of *customers* which have to be delivered with goods from a *depot* and we define $N := C \cup \{0, n_c + 1\}$ the set of *locations* or *nodes* where the index 0 and $n_c + 1$ resp., represents the depot. Associated with every customer, say i, $i = 1, \ldots, n_c$, there is a positive *demand* d_i, a *service time* s_i and a *time window* defined by the *earliest time* e_i and the *latest time* l_i for beginning of service.

At the depot a fleet $V = \{1, \ldots, n_v\}$ of vehicles is available, each one having the same *capacity*, say Q with $Q \geq \max_{i \in C} d_i$. For a concrete problem instance the number n_v may be unspecified in which case we assume n_v to be sufficiently large.

For any two locations i and j, $i, j \in N$, we know the *cost* $c_{i,j}$ of direct travel from i to j and the *travel time* $t_{i,j}$ assuming $t_{0,n_c+1} = c_{0,n_c+1} = 0$. Note that for certain instances distances $d_{i,j}$ between locations may be given together with a function to calculate $c_{i,j}$ and $t_{i,j}$, respectively.

The problem is to find an assignment of customers to vehicles such that the capacity constraints are fulfilled and for every vehicle a routing through the customer locations beginning and ending at the depot such that the time window constraints are fulfilled.

This problem is modelled through a set of decision variables:

$$x_{ij}^v = \begin{cases} 1 & \text{if vehicle } v \in V \text{ travels directly from } i \text{ to } j \quad (i, j \in N) \\ 0 & \text{else} \end{cases}$$

b_i = time at which service begins at customer $i \in C$.

b_0^v = time at which vehicle v leaves the depot.

$b_{n_c+1}^v$ = time at which vehicle v returns to the depot.

If a vehicle travels directly from customer i to customer j and it arrives to early at j, i.e. before e_j, it will have to wait, that is $b_j = \max\{e_j, b_i + s_i + t_{i,j}\}$ holds with $b_j - b_i + s_i + t_{i,j}$ the so-called *waiting time* at customer j.

Defining y_i to be the maximum possible load of the vehicle servicing node $i \in C$ after servicing i, we can model the set $X = \left(x_{i,j}^v\right)$ of feasible solutions as follows:

$$\sum_{v \in V} \sum_{j \in N} x_{i,j}^v = 1 \qquad \forall i \in C \tag{1}$$

$$\sum_{j \in N} x_{i,j}^v - \sum_{j \in N} x_{j,i}^v = 0 \qquad \forall i \in C, v \in V \tag{2}$$

$$\sum_{j \in N} x_{0,j}^v = 1 \qquad \forall v \in V \tag{3}$$

$$\sum_{j \in N} x_{j,n+1}^v = 1 \qquad \forall v \in V \tag{4}$$

$$x_{i,j}^v = 1 \Longrightarrow \qquad b_i + s_i + t_{i,j} \leq b_j \qquad \forall i, j \in C, v \in V \tag{5}$$

$$
\begin{array}{rcll}
x^v_{0,j} = 1 \Longrightarrow & b^v_0 + t_{0,j} \leq b_j & \forall j \in C, v \in V & (6) \\
x^v_{i,n+1} = 1 \Longrightarrow & b_i + s_i + t_{i,n+1} \leq b^v_{n+1} & \forall i \in C, v \in V & (7) \\
e_i \leq b_i \leq l_i & & \forall i \in C & (8) \\
e_0 \leq b^v_0 \leq l_0 & & \forall v \in V & (9) \\
e_{n+1} \leq b^v_{n+1} \leq l_{n+1} & & \forall v \in V & (10) \\
x^v_{i,j} = 1 \Longrightarrow & y_i - d_j = y_j & \forall i, j \in N, v \in V & (11) \\
y_0 = Q, 0 \leq y_i & & \forall i \in C & (12) \\
x^v_{i,j} \in \{0,1\} & & & (13)
\end{array}
$$

The efficiency of every feasible solution $X = \left(x^v_{i,j}\right)$ can now be evaluated on the basis of several different criteria which can play the role of an objective function for the above model:

$$\text{cost } (X) = \sum_{v \in V} \sum_{i \in N} \sum_{j \in N} c_{i,j} * x^v_{i,j}$$

measures the operating cost of a solution eventually based on the travel distances and travel time as well as waiting time.

$$\text{num\_vehicles } (X) = \sum_{v \in V} \sum_{j \in C} x^v_{0,j}$$

gives the total number of vehicles actually used for service, and

$$\text{schedule\_time } (X) = \sum_{v \in V} \left(b^v_{n_c+1} - b^v_0\right)$$

gives the total schedule time.

In our approach we assume the minimization of the number of vehicles to be the most important goal and we treat the problem

$$\text{lex min} \left\{ \left(\begin{array}{c} \text{num\_vehicles } (X) \\ \text{cost } (X) \end{array} \right) | X \text{ fulfills (1)–(13)} \right\}$$

The schedule time of a solution is measured and reported only, but can be influenced through the parameter setting.

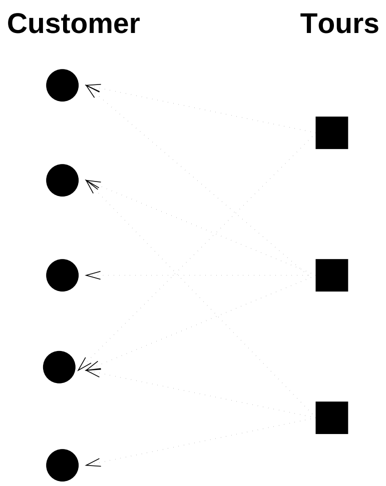
We attack this multi-criteria problem by heuristically solving a sequence of single cost objective problems under an additional constraint for the number n_v of available vehicles. As every heuristic our procedure is not guaranteed to produce the optimal solution, but it is designed to confront the decision maker with a small set of alternative solutions which are good with respect to all criteria mentioned above. This is achieved by running the procedure with different parameter settings.

2 Parallel Tour Building through an Auctioning Process

2.1 The Conceptual View

A similar idea of parallel tour building, i.e. the simultaneous construction of several tours via insertion has been proposed by Potvin and Rousseau [7] recently.

Figure 1: make_requests



Since for a problem instance the (minimal) number of routes, i.e. the number of vehicles necessary to fulfill all service requests is not known a priori, Potvin and Rousseau propose to initialize the solution process with an estimation for this number obtained through an application of the sequential algorithm of Solomon [8].

In our approach we avoid this preprocessing and create a new route requesting an additional vehicle every time we find an unrouted customer with no feasible insertion position in any of the previously constituted tours.

From a conceptual point of view, our procedure for inserting customers into a schedule of existing tours can be described as the sequential application of an auctioning process or market game with proposals and acceptance as follows:

Phase 1 make_requests Each unrouted customer u requests and receives from every tour in the schedule a price for a potential service, i.e. an insertion.

Phase 2 make_proposals Each unrouted customer who did receive a (finite) offer sends a proposal to that tour which offered him the lowest price.

Phase 3 accept_proposals Each tour accepts among all proposals received the customer u, whose insertion is most efficient.

Note that in phase 1 we can assume an infinite price if no insertion is possible. For implementing phase 3 the concept of efficiency has to be specified. The figures 1 to 4 illustrate the three phases of the procedure.

The auctioning process described above would be feasible from an economic point of view if customers and tours would be independent agents in a market acting according to individual objectives. Yet, in our case we have to alter the basic market mechanism due to the presence of the global objective to serve all customers using a limited (minimal) number of tours. Therefore we introduce the concept of a "master" or "global control" with the following operators:

Figure 2: make_proposals

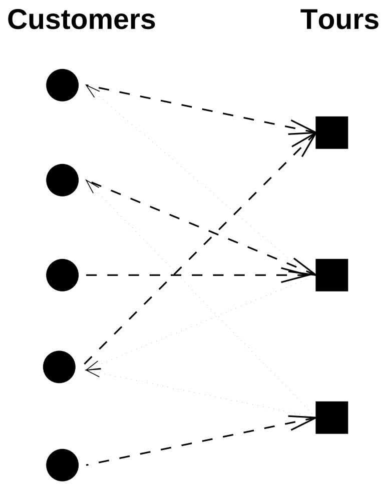


Figure 3: accept_proposals

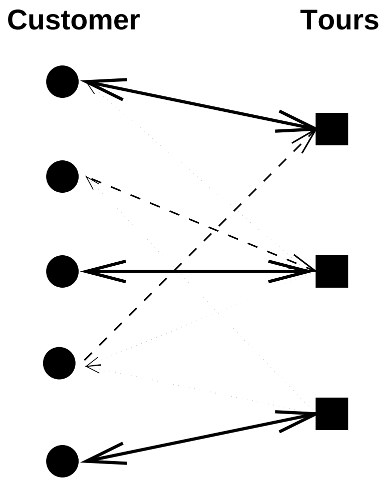
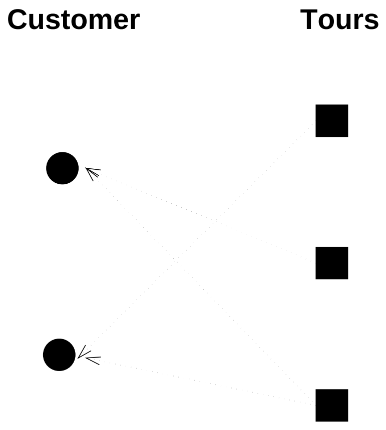


Figure 4: next round: make_requests



- the number of tours acting as agents in the game may be set to the maximum number n_v or may be enlarged one by one if there is a non routed customer without any possible insertion
- with their proposals customers inform the tour about the number of offers they received and tours have to select among the proposals of those customers with the fewest number of alternatives and
- if the auction does not succeed to produce a feasible solution with at most n_c tours, then the tours are restructured by eliminating a certain number of (most) inefficient customers from tours through an operator `unschedule` and the process is initiated again.

Obviously it would be ineffective using the same efficiency criterion for insertion and elimination. Thus unscheduling is done with respect to a different local efficiency measure.

So far we have described the auctioning (with restructuring) procedure which is the core of our *VRPTW*-heuristic. Besides this construction mechanism the entire procedure employs three additional concepts for improving feasible solutions, reducing the number of vehicles and for combining different criteria.

1. The unschedule-operator is also used to enable improvements of a feasible solution, i.e. a complete schedule produced by the construction phase, through another application of the auctioning.
2. Assume that we have found a feasible solution using n_r tours (vehicles), then we can try to resolve the problem instance setting $n_v = n_r - 1$.
3. For an implementation of the above concept the (local) criteria and decision rules, i.e. price function etc. have to be specified. Here rerunning the procedure with different parameterisations of the decision rules may improve the quality of the solution.

2.2 Specification

Our method is based on certain local efficiency criteria for a potential insertion of a customer into a tour and a potential deletion of a customer from a tour, respectively. Such heuristically motivated decision rules have been used successfully in construction heuristics for solving the traveling salesman problem (cf. [3]) and were adequately adapted by Solomon [8] to the more complex situation in *VRPTW*. Here we extend this principle applying similar rules for deleting customers from tours and thereby enabling improvements.

Throughout the following $(i_0, \ldots, i_m)$ will always denote a feasible tour with $i_0 = 1$ and $i_m = n_c + 1$. The first customer, say j, assigned to a vehicle and thus constituting a tour $(1, j, n_c + 1)$ is called the "seed-customer" for that tour and may be selected among the unrouted customers according to special seeding criteria, which we do not discuss here.

In contrast to the traveling salesman situation checking feasibility of a potential insertion is a nontrivial and possibly time consuming task in the presence of time window constraints. Here, Solomon [8] has given necessary and sufficient conditions for an insertion to be feasible. Verification of these conditions is obviously part of any local efficiency consideration. We do not discuss this issue here and refer to [8] for details.

Now assume *tour*$= (i_0, \cdots, i_m)$ to be a feasible tour and u to be an unrouted customer. Then we define

$$\text{price } (i_{p-1}, u, i_p) = \text{cost of inserting } u \text{ into } tour \text{ at position } p \ ,$$

i.e. between i_{p-1} and i_p with the cost being set to infinity if an insertion is not feasible and

$$\text{efficiency } (i_{p-1}, u, i_p) = \text{efficiency of inserting } u \text{ into } tour \text{ at position } p \ .$$

Based on these values we can define for every unrouted customer with at least one feasible position for insertion the optimal position $p^*(u)$ and the associated cost via

$$\begin{aligned} \text{price } (u, tour) &= \text{price } (i_{p^*(u)-1}, u, i_{p^*(u)}) \\ &= \min_{p=1\ldots m} \{\text{price } (i_{p-1}, u, i_p)\} \ . \end{aligned}$$

A proper definition of price and efficiency is crucial for the quality of the procedure. To allow comparisons of our algorithm with Solomon's insertion algorithm we follow in the computational experiments presented in this paper the proposals of Solomon, who suggested the following measures:

$$\begin{aligned} \text{price } (i, u, j) &= \alpha_1(t_{i,u} + t_{u,j} - \mu t_{i,j}) + \alpha_2(b_{j,u} - b_j) \\ \text{efficiency } (i, u, j) &= \lambda t_{o,u} - \text{price } (i, u, j) \end{aligned}$$

where $\alpha_1, \alpha_2, \lambda, \mu \geq 0$ and $\alpha_1 + \alpha_2 = 1$ are user defined parameters and $b_{j,u}$ is the updated time for service to begin at j after insertion of u.

Thus the cost for inserting a customer u into a tour is measured by the necessary detour and the possible delay when servicing u, while the efficiency is determined by the gain the insertion gives versus the service of u by a separate tour.

Figure 5: procedure negotiate

```
bool
procedure negotiate (vrp, unrouted, schedule, p)
begin
  feasible = make_proposals (vrp, unrouted, schedule, p)
  while (feasible ∧ ∅ ≠ unrouted) do
    accept_proposal (vrp, unrouted, schedule, p)
    feasible = make_proposals (vrp, unrouted, schedule, p)
  endwhile
  return feasible
end negotiate
```

In addition we define an efficiency coefficient for deleting a customer $w = i_p$ from a tour as follows

$$\text{expenditure } (w, tour) = b_{i_{p+1}} - b_{i_{p-1}} - s_{i_{p-1}} \quad .$$

The larger this value is the more likely is the possibility for inserting a customer u which is unrouted so far into tour at position p, i.e. the possibility of replacing w by u. Note that also criteria different from the type introduced above may be incorporated into our method.

2.3 Pseudocode

In the following we formulate our procedure more precisely using high level pseudocode with the following parameters and data:

vrp representation of the problem instance including data for customers, demands, time-windows, vehicles etc.

schedule representation of the current (partial) solution by a set of tours

unrouted the set of presently unrouted customers

p the set of parameter settings for the local efficiency criteria/decision rules (i.e. for our implementation the specification of $\mu, \lambda, \alpha_1, \alpha_2$ etc.)

max_tours maximum number of allowed tours

max_trials user defined parameter for the number of trials for certain subroutines

The description of the whole procedure is divided into three modules `construction`, `improvement` and `tour_reduction` which are then combined to the procedure `vrptw_solver`.

Figure 6: procedure make_proposals

```
bool
procedure make_proposals (vrp, unrouted, schedule, vrp)
begin
  feasible = true
  for all (customer ∈ unrouted) do
    for all (tour ∈ schedule) do
      e = cost(p) of servicing customer by tour
      store cheapest (e,tour) tuple
    endfor
    if (there is a feasible tuple (e, t) stored)
      request service of customer by tour t
    else
      feasible = false
    endif
  endfor
  return feasible
end make_proposals
```

Figure 7: procedure accept_proposals

```
procedure accept_proposal (vrp, unrouted, schedule, p)
begin
  for all (tour ∈ schedule) do
    insert the cheapest customer
    requesting service in tour
  endfor
end accept_proposal
```

Figure 8: procedure construction

```
bool
procedure construction (vrp, schedule, p, max_tours, max_trials)
begin
  feasible = true
  trial    = 0
  unrouted = all customers of the vrp
  while (feasible ∧ ∅ ≠ vrp) do
    feasible = negotiate (vrp, unrouted, schedule, p)
    if (¬ feasible)
      if (# tours in schedule < max_tours)
        open another tour in schedule
      else
        if (trial < max_trials)
          trial = trial + 1
          if (trial mod 5)
            number = 1
          else
            number = 1 + trial / 5
          endif
          unschedule (vrp, unrouted, schedule, number)
          feasible = true
        endif
      endif
    endwhile
    return feasible
  end construction
```

2.3.1 Construction

At the core of the construction-procedure we iterate through a cycle with customers requesting prices, selecting a tour for proposal and tours selecting customers among received proposals. This module called `negotiate` is described by the pseudo-code in figure 5. Note that due to the higher efficiency in a non-distributed environment the calculations for phase 1 and phase 2 of the auctioning are done in parallel in a single subroutine `make_proposals` (cf. figure 6).

Procedure `negotiate` is processed until one of two conditions occur. Either `make_proposals` returns false in which case we have found an unrouted customer with no feasible insertion, or we have routed all customers.

While procedure `make_proposals` uses the price function, the selection in procedure `accept_proposal` (cf. figure 7) is based on the priority of the customers and the local efficiency function.

Figure 8 gives the complete pseudocode for our construction heuristic based on the `negotiate`-procedure. In the case that `negotiate` stops with an unroutable customer we either increase the number of vehicles if possible and reenter `negotiate` or we rearrange the schedule by first unscheduling a certain number of most expensive customers using the expenditure function and then reentering `negotiate`. To prevent cycling we alter the number of customers to become unscheduled with increasing number of trials using a simple criterion specified in figure 8.

2.3.2 Improvement

Using the modules `unschedule` and `negotiate` we can easily formulate an improvement method as shown in figure 9. Given a feasible schedule we repeatedly try to unschedule a certain number of (expensive) customers which are then rerouted through an application of `negotiate`.

2.3.3 Tour Reduction

Figure 10 gives the pseudocode for tour reduction using the simple idea described by Potvin and Rousseau [7]. After obtaining a feasible solution for the problem using n_t vehicles/tours, the construction procedure (using the same parameter setting) is applied with $n_t := n_t - 1$.

2.3.4 The Complete Solver

In figure 11 we assemble the parts introduced before and obtain `vrptw_solver` a quite flexible control procedure. Following the ideas of Solomon [8] `vrptw_solver` tries to find a good schedule by performing multiple runs with varying parameter settings for the local efficiency criteria.

To achieve a good performance we first try to identify a parameter setting which is well suited for the problem instance by running the pure construction procedure for all parameter settings. Then the modules `tourreduction` and/or `improvement` are applied for the best parameter setting only, i.e. variants "best tour reduction"/ "best improvement". Alternatively we apply them for all parameter settings, i.e. variants "all tour reduction"/"all improvement".

Figure 9: procedure improvement

```
bool
procedure improvement (vrp, schedule, p, max_tours, max_trials)
begin
  improved   = false
  trial      = 0
  cur_sched = schedule
  unrouted   = ∅
  while (trial ≤ max_trials) do
    trial = trial + 1
    if (trial mod 5)
      number = 1
    else
      number = 1 + trial / 5
    endif
    unschedule (vrp, unrouted, cur_sched, number)
    feasible = negotiate (vrp, unrouted, cur_sched, p)
    if (feasible ∧ cur_sched is better than schedule)
      schedule = cur_sched
      improved = true
    endif
  endwhile
  return improved
end improvement
```

Figure 10: procedure tourreduction

```
bool
procedure tourreduction (vrp, schedule, p, max_trials)
begin
  n_tours = number of tours in schedule
  improved = false
  feasible = true
  while (feasible) do
    n_tours = n_tours - 1
    feasible = construction (vrp, s, p, n_tours ,max_trials)
    if (feasible)
      schedule = s
      improved = true
    endif
  endwhile
  return improved
end tourreduction
```

Figure 11: procedure vrptw_solver

```
schedule
procedure vrptw_solver (vrp, pset, max_trials)
begin
  for all (p ∈ pset) do
    construction (vrp, cur_sched, p, ∞, max_trials)
    if (all tour reduction)
      tourreduction (vrp, cur_sched, p, max_trials)
    endif
    if (all improvement)
      improvement (vrp, cur_sched, p, max_trials)
    endif
    if (cur_sched is better than best_sched)
      best_sched = cur_sched
      best_p     = p
    endif
  endfor
  if (best tour reduction)
    tourreduction (vrp, best_sched, best_p, max_trials)
  endif
  if (best improvement)
    improvement (vrp, best_sched, best_p, max_trials)
  endif
  return best_sched
end vrptw_solver
```

Table I: Reference Results

group	(#)	**I1_S**		**I1_AD**		**prb**	
		tours	schedule t.	tours	schedule t.	tours	schedule t.
c1	9	10	10104.2	10.00	10104.61	10.7	10610.3
c2	8	3.1	9921.4	3.25	9886.88	3.4	10477.6
r1	12	13.6	2695.5	13.83	2736.25	13.3	2696.0
r2	11	3.3	2578.1	3.18	2539.20	3.1	2513.3
rc1	8	13.5	2775.0	13.5	2803.54	13.4	2877.9
rc2	8	3.9	2955.4	4.00	2968.60	3.6	2807.4

Table II: Direct Comparison on Groups Averages

	I1_AD						**prb**					
method	c1	c2	r1	r2	rc1	rc2	c1	c2	r1	r2	rc1	rc2
construction only	-	+	+	-	-	+	+	+	-	-	-	-
best improvement	-	+	+	+	+	+	+	+	-	+	+	-
best tour reduction	-	+	+	+	+	+	+	+	+	+	+	-
best tr. + best impr.	-	+	+	+	+	+	+	+	+	+	+	-
all improvement	-	+	+	+	+	+	+	+	-	+	+	-
all tour reduction	-	+	+	+	+	+	+	+	+	+	+	+
all tr. + all impr.	+	+	+	+	+	+	+	+	+	+	+	+

3 Computational Results

For our computational tests we have used the standard set of problems provided by Solomon [8]. The 56 problem instances in this set are all 100 customer problems with Euclidean distances. The set is divided into six groups of problems:

- the locations are either randomly (*r*) generated or clustered (*c*) or a mix of both (*rc*)
- these alternatives are then combined with either a short scheduling horizon (*1*) allowing only a few customers per tour or a long scheduling hori-

Table III: comparison against insertion

method	runtime factor	comparison
construction only	1	$18_{[6,12]}$: $37_{[9,28]}$
best improvement	2.4	$33_{[8,25]}$: $22_{[4,18]}$
best tour reduction	3.5	$30_{[19,11]}$: $25_{[1,24]}$
best tr. + best impr.	4.8	$35_{[20,15]}$: $21_{[1,20]}$
all improvement	6.5	$38_{[9,29]}$: $17_{[3,14]}$
all tour reduction	12.2	$38_{[23,15]}$: $17_{[0,17]}$
all tr. + all impr.	17.4	$49_{[23,26]}$: $6_{[0,6]}$

zon (*2*).

Details on the characteristics of the test problems can be found in Solomon [8].

All our procedures were coded in ANSI-C and all runs were performed on a `RS6000/530` running `AIX 3.2.5` using the `xlc` compiler with the simple optimization `-O` mode. Throughout our computational test reported here we have used `max_trials` = 100. Our experiments have indicated that in the implementation of our method, further on refered to as **vrptw_solver**, an explicit seeding of tours is not necessary, yet the quality of the schedules vary with the values for $\mu, \lambda, \alpha_1, \alpha_2$. In order to produce results which are comparable, we introduce the following four parameter settings proposed by Solomon [8] into the set p:

- $\mu = 1.0$, $\lambda = 1.0$, $\alpha_1 = 1.0$, $\alpha_2 = 0.0$
- $\mu = 1.0$, $\lambda = 2.0$, $\alpha_1 = 1.0$, $\alpha_2 = 0.0$
- $\mu = 1.0$, $\lambda = 1.0$, $\alpha_1 = 0.0$, $\alpha_2 = 1.0$
- $\mu = 1.0$, $\lambda = 2.0$, $\alpha_1 = 0.0$, $\alpha_2 = 1.0$

We compare our results with the results published for

I1_S the insertion procedure I1 by Solomon [8]

prb the parallel route building algorithm by Potvin and Rosseau [7].

Based on the published data a comparison for every single problem instance is not possible, since only averages for the different groups of problems are reported in [8] and [7]. Moreover due to the different computer hardware and software a comparison of the running times based on the reported CPU-times, only, would be critical. Therefor we have coded an implementation of the insertion procedure I1 refered to as **I1_AD** using data type `double` for time and distance. The results for this comparison are given in table I.

For our tables we use the following column layout:

groups	partition of problem instances as described above
#	number of instances in groups
tours	(average) number of tours/vehicles used
schedule time	(average) schedule time
distance	(average) cost of schedule, i.e. sum over travel distances
waiting	(average) waiting time, i.e. sum over individual waiting times of the customers
run time	(average) CPU-time in seconds.

For **I1_AD** we obtain results slightly different from those published by Solomon [8], with respect to the number of tours, schedule time etc. (see table IV in the Appendix). We contribute this to the distinct hardware and different rounding schemes used. This effect was also observed by Potvin and Rosseau [7] and was analysed by Mole [6] and Waters [10] for the simple *VRP* case. Detailed results for **vrptw_solver** and **I1_AD** are listed in the appendix A. Here we only present aggregate information.

Table II summarizes the comparison between the different levels of **vrptw_solver** and **I1_AD** and **prb** on the basis of group averages. Here a "+"-sign indicates that the level described in the first column outperforms the method cited in the heading for the class of problems indicated for the respective column, i.e. the (average) number of tours for all problems in the class is smaller, or if these averages are equal, the total distance is less for **vrptw_solver**. A "–"-sign indicates the superiority of the other method.

Table II shows that already low levels of **vrptw_solver** outperform its competitors. Using all tour reduction and all improvement **vrptw_solver** is superior on all groups. We admit that this comparison based on quality only and not mentioning running time etc. is biased. Yet it is remarkable that the number of tours, i.e. the most important objective, can often be reduced significantly. Achieving this improvement the expenditure of more CPU-time should be a good investment. The excellent performance of I1 for the class c1 has already been observed by Potvin and Rosseau [7].

Table III gives a more detailed comparison of **vrptw_solver** and **I1_AD**. A comparison of the CPU-time for the different levels of **vrptw_solver** is given introducing a run time factor with respect to construction. The notation $18_{[6,12]}$, taken from row 1, indicates that the construction phase of **vrptw_solver** was strictly outperforming **I1_AD** on 18 examples, giving a smaller number of tours for 6 examples and a smaller schedule time for 12 examples, for which an identical number of tours was generated by **I1_AD**. Please note that for a small number of instances **vrptw_solver** and **I1_AD** generated identical solutions.

4 Final Remarks

The results which we have obtained and presented here indicate that our method is quite powerful and able to produce solutions of high quality for the *VRPTW*. The advantage over other methods stems from its flexibility with respect to possible implementations. Depending on the problem domain or the problem instance, special parameterizations, different efficiency criteria and a different global control may lead to improved solutions. First experiments with other control mechanisms indicate that the introduction of stochastic rules seems to be effective. Also, the implementation of the scheme in a distributed environment is evident and is under investigation by the authors.

A Detailed Results

Table IV: **I1_AD**

group	# tours	schedule time	distance	waiting	run time
c1	10.00	10104.61	960.81	107.35	9.35
c2	3.25	9886.88	740.93	143.37	24.93
r1	13.83	2736.25	1482.53	110.46	8.74
r2	3.18	2539.20	1355.24	134.03	39.86
rc1	13.50	2803.54	1610.78	54.20	8.18
rc2	4.00	2968.60	1684.43	132.48	30.96
all	8.16	4946.07	1314.90	114.40	20.36

Table V: **vrptw_solver**: construction only

group	# tours	schedule time	distance	waiting	run time
c1	10.22	10430.11	1253.52	139.81	7.91
c2	3.12	9847.18	815.05	32.13	15.34
r1	13.75	2793.22	1614.37	98.59	8.73
r2	3.18	2553.17	1437.29	96.16	18.36
rc1	13.87	2987.80	1887.07	48.10	7.59
rc2	3.87	3019.31	1830.11	93.53	15.28
all	8.19	5041.23	1477.18	87.31	12.21

Table VI: **vrptw_solver**: best improvement

group	# tours	schedule time	distance	waiting	run time
c1	10.11	10264.76	1103.18	98.38	26.93
c2	3.12	9802.79	769.65	33.14	20.28
r1	13.58	2685.74	1508.48	80.39	32.86
r2	3.18	2498.48	1366.69	106.89	34.34
rc1	13.37	2807.57	1636.41	49.99	25.34
rc2	3.87	2889.27	1700.25	79.50	33.41
all	8.07	4930.21	1355.61	77.27	29.41

Table VII: **vrptw_solver**: best tour reduction

group	# tours	schedule time	distance	waiting	run time
c1	10.11	10419.93	1248.78	134.37	34.37
c2	3.00	9829.55	783.62	45.94	38.78
r1	13.00	2691.28	1527.69	99.20	44.62
r2	3.09	2538.95	1423.89	95.34	48.72
rc1	12.75	2813.68	1734.78	40.63	34.50
rc2	3.62	2955.56	1813.42	99.55	49.75
all	7.78	4978.46	1426.58	88.17	42.23

Table VIII: **vrptw_solver**: best tour reduction + best improvement

group	# tours	schedule time	distance	waiting	run time
c1	10.11	10253.62	1104.40	96.45	53.22
c2	3.00	9791.95	742.04	49.92	43.19
r1	13.00	2638.03	1471.28	92.54	69.39
r2	3.09	2483.00	1349.53	108.58	65.23
rc1	12.75	2752.74	1652.63	37.49	51.77
rc2	3.62	2822.61	1691.85	74.34	68.41
all	7.78	4896.26	1341.64	79.76	59.57

Table IX: **vrptw_solver**: all improvement

group	# tours	schedule time	distance	waiting	run time
c1	10.11	10217.19	1047.43	98.38	94.36
c2	3.12	9798.03	764.89	33.14	41.88
r1	13.41	2648.70	1463.07	84.97	101.70
r2	3.18	2434.37	1369.72	45.44	75.63
rc1	13.37	2763.37	1609.58	40.48	79.89
rc2	3.87	2838.94	1625.91	126.52	74.39
all	8.03	4887.86	1322.39	71.54	79.84

Table X: **vrptw_solver**: all tour reduction

group	# tours	schedule time	distance	waiting	run time
c1	10.00	10210.11	1119.58	70.13	135.21
c2	3.00	9797.38	731.86	65.52	105.39
r1	12.83	2631.91	1487.25	96.73	157.44
r2	3.09	2515.63	1434.86	60.05	155.73
rc1	12.5	2720.24	1611.30	64.46	141.49
rc2	3.37	2774.64	1681.50	59.36	150.21
all	7.66	4883.64	1355.43	70.84	142.78

Table XI: **vrptw_solver**: all tour reduction all improvement

group	# tours	schedule time	distance	waiting	run time
c1	10.00	10052.48	955.39	51.66	220.75
c2	3.00	9762.40	717.31	45.09	131.62
r1	12.83	2583.64	1386.46	107.32	253.29
r2	3.09	2433.63	1366.48	50.27	219.86
rc1	12.5	2670.47	1545.92	56.62	213.40
rc2	3.37	2691.56	1598.06	87.71	217.22
all	7.66	4807.88	1270.67	68.23	213.26

References

[1] N. Christofides. 1985. Vehicle routing. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem*, chapter 12, pages 431–448. John Wiley & Sons Ltd., Chichester.

[2] M. Desrochers, J. K. Lenstra, and M. W. P. Savelsbergh. 1990. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research*, 46, 322–332.

[3] B. Golden, L. Bodin, T. Doyle, and W. Stewart Jr., 1980. Approximate traveling salesman algorithms. *Operations Research*, 28:3, 694–711.

[4] B. L. Golden and A. A. Assad, editors, 1988. *Vehicle Routing: Methods and Studies.* North-Holland, Amsterdam.

[5] G. Laporte. 1992. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59, 345–358.

[6] R. H. Mole. 1983. The curse of unintended rounding error: A case from the vehicle scheduling literature. *Journal of the Operational Research Society*, 34:7, 607–614.

[7] J.-Y. Potvin and Jean-Marc Rousseau. 1993. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66, 331–340.

[8] M. M. Solomon. 1987. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35:2, 254–265.

[9] M. M. Solomon and J. Desroisiers. 1988. Time window constrained routing and scheduling problems. *Transportation Science*, 22:1, 1–13.

[10] C. D. J. Waters. 1984. Vehicle scheduling revisited. *Journal of the Operational Research Society*, 35:2, 145–148,