Report No. 93.139

The Simulated Trading Heuristic for Solving Vehicle Routing Problems

by

A. Bachem, W. Hochstättler, M. Malich

1993

Mathematisches Institut
Universität zu Köln
Weyertal 80
50931 Köln, Germany

1991 Mathematics Subject Classification: 90B05, 90C27
Keywords: vehicle routing problem, heuristics, improvement heuristics, parallel computing

The Simulated Trading Heuristic for Solving Vehicle Routing Problems

A. Bachem, W. Hochstättler, M. Malich

Revised April 1994

Abstract

We present an improvement heuristic for vehicle routing problems. The heuristic finds complex customer interchanges to improve an initial solution. Our approach is modular, thus it is easily adjusted to different side constraints such as time windows, backhauls and a heterogeneous vehicle fleet. The algorithm is well suited for parallelization. We report on a parallel implementation of the Simulated Trading heuristic on a cluster of workstations using PVM. The computational results were obtained using two sets of vehicle routing problems which differ in the presence of time windows. Our results show that Simulated Trading is better suited for problems with time windows.

1 Introduction

Vehicle routing problems can be found in many variants suggested from applications (see [21, 14] for a survey on vehicle routing problems and algorithms). The standard vehicle routing problem is given as follows: a set of n customers with demands d_i has to be served from a depot using t trucks of capacity Q. The objective is to minimize the total distance traversed by the trucks or some other measure such as costs, time, etc. In this paper we present a new improvement heuristic for vehicle routing problems with additional side constraints. The main procedure performs insert resp. delete operations depending only on a cost function which is computed in a different module and thus it is easy to adapt Simulated Trading to common side constraints such as time windows, service times, backhauls, etc.

The vehicle routing problem is a generalization of the traveling salesman problem (TSP). Unfortunately, the ideas which led to the progress in solving large scale traveling salesman problems in recent years cannot be applied to vehicle routing problems. This is mostly due to the fact that all the improvement heuristics known for the TSP (k–opt, Lin/Kernighan, etc.) can — if at all — only be used for local improvement within one of the tours given so far. Moreover, time windows and backhauls usually define precedence constraints which restrict the possible interchanges. Changing customers between tours (global improvement) is essential for good improvement heuristics. Examples for such heuristics can be found in [23, 9, 16].

In the following Section 2 we will present the main idea of Simulated Trading by an example. In Section 3 we give some definitions and notation. The description of the basic heuristic is given in Section 4 followed by a section revealing some extensions of the algorithm. In Section 6 we will describe two parallel approaches for Simulated Trading, and finally in Section 7 we present our computational results using the test problem library of Solomon [19, 20] as well as fourteen standard vehicle routing problems taken from [4].

We assume some familiarity with vehicle routing (see e.g. [2]) and use standard graph theoretic notation (see [1]).

2 The Idea of Simulated Trading

The key idea of Simulated Trading is to apply the mechanisms of trading to optimizing the partition of the customers into the tours. To get an idea of this consider the following talk between four truck drivers:

Joe: "*I would be glad if someone could serve my customer A otherwise I will need an extra hour of driving time.*"

Jim: "*A is right on my way but my truck is full. I need 20 minutes additional time for customer B. If someone can satisfy these demands I will be able to insert A in my tour with only 10 minutes of extra time.*"

Jack: "*My truck has enough free capacity for B, but if I would insert B in my tour then I would violate the time window of C.*"

John: "*If I serve B then I would arrive at home too late. But if someone delivers my customer D then I would be able to take B and would be back in time.*"

Joe concludes: "*I'm able to serve customer C with costs of 15 minutes, and we have the possibility to make an overall improvement of 40 minutes.*"

The exchanges offered in the discussion define a leveled bipartite graph — we call it *trading graph* — (see Figure 1). The nodes corresponds to either an insertion (buy) of a customer into a tour or a deletion (sell). The edges representing the possible exchanges are weighted with the gain (possibly negative) that is obtained by the corresponding action.

Every matching of the trading graph corresponds to a number of interchanges of customers. If the value of the matching (i.e. the sum of the weights of the matching edges) is positive we can improve our current tourplan. Clearly not every matching leads to a *feasible* tourplan. Consider the matching $\{((A, 60), (A, 10))\}$: realizing this move of customer A from Joe to Jim, the load would exceed the capacity of Jim's truck. Thus, we have to look for weighted matchings respecting

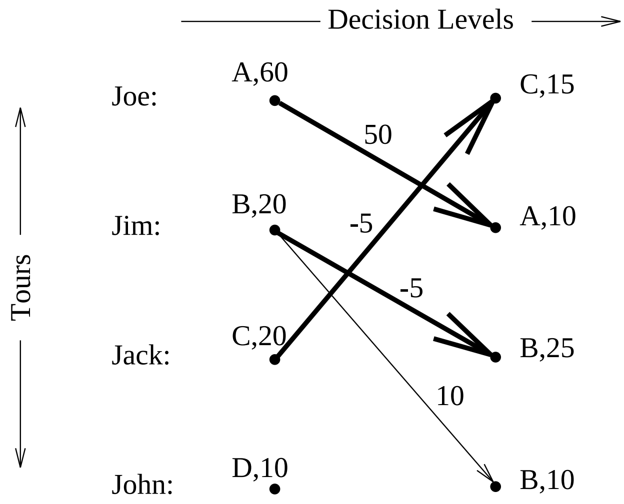


Figure 1: The trading graph of our example.

some additional level constraints. The bold printed edges in Figure 1 show such a *trading matching* improving the tourplan by 40.

3 Notation and Definitions

As mentioned above the main procedure of our heuristic performs insertion and deletion operations depending on some cost function.

Definition 3.1 *Let the set of customers be $V = \{1, \ldots, n\}$. A* **tour** *is any subset $T \subseteq V$. The* **cost** *of a tour is given by a weight function $c : 2^V \to \mathbb{R} \cup \{\infty\}$. A* **tourplan** *$\mathcal{T} = (T_1, \ldots, T_t)$ is a partition of V. The cost of the tourplan is given by $c(\mathcal{T}) = \sum_{i=1}^{t} c(T_i)$.*
If T is a tour and $i \in T$ the **deletion cost** $c^-(T, i)$ **of** i **from** T *is given by*

$$c^-(T, i) = c(T) - c(T - \{i\}).$$

If $i \notin T$ we get analogously the **insertion cost of** i **into** T.

$$c^+(T, i) = c(T + \{i\}) - c(T).$$

In our application the cost function is computed by a subroutine which routes a — possibly good — tour visiting all customers and respecting the given side constraints. For this task we use a simple cheapest insert procedure. But we could also have been used an insertion resp. deletion procedure which does some local improvement (see e.g. [8] for GENI improvement or [18] for Or–Exchange). Infeasible instances (e.g. violation of time windows constraints) can be taken into account by a cost of infinity.

4 Description of the Algorithm

4.1 Algorithm Outline

In the first place we would like to give an outline of the algorithm. Roughly our algorithm proceeds as follows:

```
T = (T_1, ..., T_t) is a feasible tourplan
repeat
    Sell–And–Buy phase
    Trading–Matching–Search phase
    if Trading–Matching M is found then
        Update tourplan T according to M
until timelimit is reached
```

In the Sell-And-Buy phase each tour is checked for "good" trading possibilities depending on its previous actions. This defines the trading graph which is searched for a maximum weighted trading matching in the next phase (the definition of a trading graph resp. a trading matching will be given later in this section).

We will now discuss these routines in more detail.

4.2 The Sell–And–Buy Phase

We construct the trading graph level by level until a given maximum level is reached. In each level we choose for each tour either a sell or a buy action or we do nothing at all. It has been proven advantageous not to process the tours in a fixed order. Thus, in each iteration first we shuffle the tours by choosing some permutation at random. Then for each tour in a sell action we try to sell customers which cause large costs. In a buy action we prefer customers which achieve a large gain (savings − insert costs).

After initializing $T_i^0 = T_i$ and $l_i = 0$ for all $i = 1, \ldots, t$ the computation of one level is done as follows:

```
procedure process_level
    Shuffle tours
    for k=1 TO t
        Choose buy or sell action
        if (sell action) then
            Choose "good" customer i ∈ T_k^{l_k}
            T_k^{l_k+1} = T_k^{l_k} − {i}
            Add customer i to selling list S, s(i) = c^−(T_k^{l_k}, i)
            Add sell node to graph
            l_k = l_k + 1
        else
            if we can choose "good" customer i ∉ T_k^{l_k} from S then
                Add buy node and appropriate edges to trading graph
                T_k^{l_k+1} = T_k^{l_k} + {i}
                l_k = l_k + 1
            endif
        endif
    endfor
endprocedure
```

During the procedure on one hand we store all the modified tours $T_k^{l_k}$ (l_k denotes the level) and, on the other hand, construct the bipartite trading graph where one color class consists of the buy nodes the other of the sell nodes. A sell and a buy node are adjacent if they refer to the same customer. The weight of an edge is given as the improvement of the cost function due to the customer exchange. In addition we keep a selling list S with savings $s(i)$ of offered customers.

Some comments:

- If the maximum decision level is too small (1–2) we, obviously, cannot expect complex customer interchanges. In order to reach complex interchanges and to keep the size of the trading graph and thus the computation time for the trading matching search in control a value of MaxLevel$\in \{3, \ldots, 8\}$ seems to be appropriate.
  Instead of using a fixed order of buy and sell actions we choose them at random. To control the ratio of buy and sell decisions a probability of 0.4 for a sell action performed quite well.
- The tourplan $(T_1^{l_1}, \ldots, T_t^{l_t})$ in general is not feasible. There may exist a customer which is served by more than one tour or a customer which is not served at all. Therefore, we store every tour after each decision to have an easy update when we realize a trading matching.

In the next two subsections we describe the strategy of selling and buying customers. Here again it turned out to be advantageous to add some randomness.

We do so by giving each possible buy resp. sell decision a probability according to their attractiveness.

4.2.1 Selecting a customer for sale

Let $T_i = (i_1, \ldots, i_k)$ be a tour. If $T_i = \emptyset$ we clearly cannot sell any customer. Otherwise, the probability p_h for customer i in tour T to be sold is given by:

If $\forall h = 1, \ldots, k : c^-(T, i_h) = 0$ we define $p_h = 1/k$, otherwise

$$p_h = \frac{c^-(T, i_h)}{\sum_{j=1}^{k} c^-(T, i_j)} \quad \forall h = 1, \ldots, k.$$

4.2.2 Selecting a customer from the selling list

Let $S = (i_1, \ldots, i_k)$ be the list of customers in the selling list with associated savings $s(1), \ldots, s(k)$ and T a tour. Analogously to the sell procedure we add some randomness.

If $\forall a, b \in \{1, \ldots, k\} : s(a) - c^+(T, i_a) = s(b) - c^+(T, i_b)$ we define $p_h = 1/k$, otherwise

$$\begin{aligned} m &= \min_{j=1,\ldots,k} s(j) - c^+(T, i_j) \\ p_h &= \frac{s(h) - c^+(T, i_h) - m}{\sum_{j=1}^{k}(s(j) - c^+(T, i_j)) - km} \quad \forall h = 1, \ldots, k. \end{aligned}$$

There are two disadvantages within this definition:

1. the least attractive customer gets always a probability of 0;
2. if there are only customers with negative gain offered, we would like to sometimes not buy anything, depending on the loss we had to face.

In order to overcome this we add a dummy sell customer i_{k+1} with $s(k+1) - c^+(T, i_{k+1}) = 0$ and reduce the value of m by 3% to ensure that the least attractive customer gets a positive probability. If the dummy customer is selected we do nothing at all and continue with the iteration.

4.2.3 Sell/Buy customer i

For each sell or buy decision we have to add one node to the trading graph holding the following information: tour identification, customer identification and the decision level. If a customer is going to be sold we have to add him to the selling list:

$$S = S + \{i\},\ s(i) = c^-(T, i)$$

In addition to the insertion of a buy node v we have to place an edge e from each node selling i to the new node v. The weight of e is given by the difference between the savings $s(i)$ associated with the sell node and the insert costs $c^+(T, i)$.

4.3 Trading–Matching–Search phase

In this section we formally introduce the trading graph and the trading matching and present an algorithm which finds the maximum weighted trading matching in a trading graph if one exists.

Definition 4.1 (trading graph) *Let t, n, $MaxLevel \in \mathbb{N}$ with $t \leq n$. Assume we have run the procedure process_level from level 1 up to MaxLevel–1. The corresponding* **trading graph** *is defined to be the graph G:*

$$G = (V = V_s \dot{\cup} V_b, E) \textit{ with } V \subset \{1, \ldots, t\} \times \{1, \ldots, MaxLevel\} \times \{1, \ldots, n\}.$$

where

$$\begin{aligned} v = (i, l, k) \in V_s &\Leftrightarrow \textit{tour } T_i^{l-1} \textit{ sells customer } k \textit{ in level } l & (1) \\ v = (i, l, k) \in V_b &\Leftrightarrow \textit{tour } T_i^{l-1} \textit{ buys customer } k \textit{ in level } l & (2) \\ (v, w) \in E &\Leftrightarrow v \in V_s, w \in V_b, v = (., ., k), w = (., ., k) & (3) \end{aligned}$$

Furthermore, the weight $\omega(e)$ of an edge $e = (v, w)$ with $v = (i, l, k) \in V_s$ and $w = (j, m, k) \in V_b$ is given by

$$\omega(e) = c^-(T_i^{l-1}, k) - c^+(T_j^{m-1}, k). \qquad (4)$$

Note, that G is bipartite and that the definition reflects the properties of the graph constructed during the Sell-And-Buy-Phase.

Every matching of the trading graph corresponds to an interchange of customers. If the value of the matching is positive we could improve the current tourplan. But as we have seen in Section 2 not every matching leads to a feasible tourplan. Therefore, we have to add some additional constraints to guarantee the feasibility of the new tourplan.

Definition 4.2 (trading matching) *Let $G = (V = V_s \dot{\cup} V_b, E)$ be a trading graph with $V \subset \{1, \ldots, t\} \times \{1, \ldots, MaxLevel\} \times \{1, \ldots, n\}$. Furthermore, let $\emptyset \neq M \subset E$ denote a set of edges. Let*

$$M_V = \{v \in V \mid \exists w \in V \textit{ with } (v, w) \in M\}$$

denote the set of saturated nodes. Then M is a **trading matching** $:\Longleftrightarrow$

$$\forall v \in M_V : \ \dot{\exists}(v, .) \in M \qquad (5)$$

$$\forall (i, l, k) \in M_V \ \ \forall l^* = 1, \ldots, l-1 : \ \ (i, l^*, .) \in M_V. \qquad (6)$$

The **value** *of a trading matching M is defined as:*

$$\omega(M) = \sum_{e \in M} \omega(e). \qquad (7)$$

Due to (5) M is a matching. If we match a node $v = (i, l, k)$ of the trading graph, (6) ensures that every preceding decision $w = (i, l^*, .)$ of tour i in a level $l^* < l$ is saturated by the matching, too.

In Figure 1 we gave an example for a trading graph and a trading matching M with $\omega(M) = 40$. To further clarify the concept we present a trading graph which does not admit any trading matching in Figure 2.

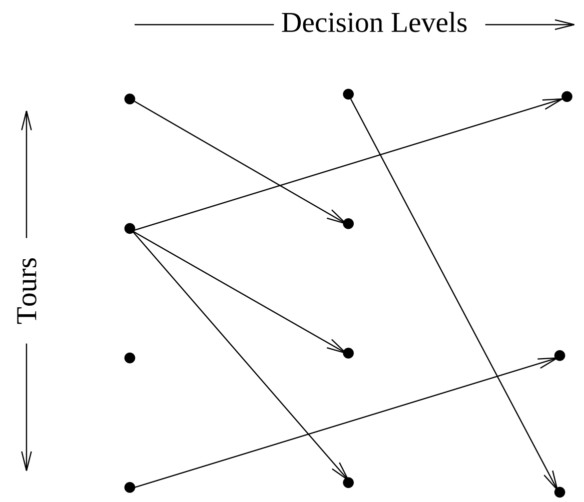


Figure 2: A trading graph without any trading matching.

Finding a maximal weighted matching in a bipartite graph is an easy task but, unfortunately, finding any trading matching at all is NP–complete.

Theorem 4.1 *The problem of finding a trading matching in a trading graph G is NP–complete even if G is a tree with 3 levels.*

Proof. Obviously the problem is in NP. To prove completeness we will reduce $SAT \leq 3$ to it. In this variant of the satisfiability problem every variable occurs in at most three clauses. This is easily seen to be NP-complete by replacing each variable x_i which occurs in d clauses in a standard SAT-instance with d variables $x_{i,k} \quad k = 1, \ldots, d$ — one for each clause — and the additional clauses

$$\{\neg x_{i,k}, x_{i,(k \bmod d)+1}\} \quad \forall k \in \{1, \ldots, d\}.$$

Given a $SAT \leq 3$ formula $\mathcal{C}$ we construct a trading graph $G(\mathcal{C})$ that has a feasible matching if and only if $\mathcal{C}$ is satisfiable.

Let the clauses be $c_1, \ldots, c_k$ and the variables $x_1, \ldots, x_m$. If a variable occurs only negative or only positive we obviously may assign the truth setting value to it and forget about its clauses. If a variable x_i appears twice positive we may

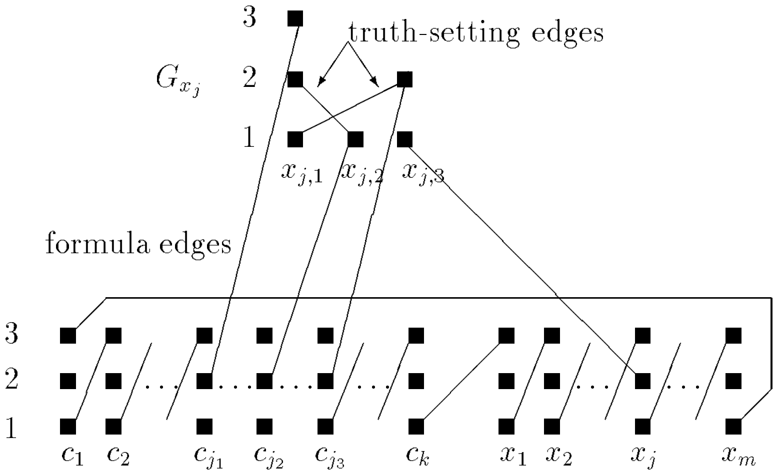


Figure 3: The sandwich graph $G(\mathcal{C})$

change the roles of x_i and $\neg x_i$. So altogether we may assume that every variable appears at least twice at all and exactly once positive.

At first we define a graph G_1 that we will refer to as the sandwich. Doing this we may restrict the data of a node to the number of the tour and its level. Thus, our nodes will be tuples (t, l). For every clause c_i and each variable x_j we define three vertices

$$(c_i, 1), (c_i, 2), (c_i, 3)$$

$$\text{resp. } (x_j, 1), (x_j, 2), (x_j, 3).$$

The sandwich edges are

$$((c_i, 1), (c_{i+1}, 3)) \text{ for } i = 1, \ldots, k-1,$$

$$((c_k, 1), (x_1, 3)),$$

$$((x_j, 1), (x_{j+1}, 3)) \text{ for } j = 1, \ldots, m-1,$$

$$((x_m, 1), (c_1, 3)).$$

Furthermore we define for every variable x_j a gadget $G(x_j)$ on six additional vertices

$$(x_{j,1}, 1), (x_{j,1}, 2), (x_{j,1}, 3),$$

$$(x_{j,2}, 1),$$

$$(x_{j,3}, 1), (x_{j,3}, 2)$$

with the truth setting edges

$$((x_{j,1}, 2), (x_{j,2}, 1)) \text{ and } ((x_{j,1}, 1), (x_{j,3}, 2)).$$

We will refer to the first of these as upper truth setting edge.

The graph $G(\mathcal{C})$ now consists of the sandwich G_1 the variable gadgets $G(x_j)$ and the following formula edges for $j = 1, \ldots, m$: If $x_j \in c_{j_1}$ and $\neg x_j \in c_{j_2}, c_{j_3}$, then the formula edges are:

$$((x_{j,1}, 3), (c_{j_1}, 2))$$

$$((x_{j,2}, 1), (c_{j_2}, 2)) \text{ and } ((x_{j,3}, 2), (c_{j_3}, 2))$$

$$((x_{j,3}, 1), (x_j, 2))$$

The second or third of these edges must be omitted if $\neg x_j$ occurs only once in the formula.

Now $G(\mathcal{C})$ obviously is a tree with three levels and hence a trading graph.

Claim. If $G(\mathcal{C})$ has a feasible matching then $\mathcal{C}$ is satisfiable.

Let M be a feasible matching. First note that if one vertex in the sandwich is matching saturated so are all of them. The only edges not adjacent to a sandwich vertex are the truth setting edges. But if one of them is in M say for example $((x_{j,2}, 1), (x_{j,1}, 2))$ so due to the level constraint must be $((x_{j,1}, 1), (x_{j,3}, 2))$, $((x_{j,3}, 1), (x_j, 2))$ and hence the whole sandwich must be matched anyway.

For $j = 1, \ldots m$ we assign the value true to variable x_j if the upper truth setting edge of the graph $G(x_j)$ is in M and false elsewhere. Now let c_i be any clause of the formula. As mentioned above the vertex $(c_i, 2)$ must be matched by a formula edge to a variable gadget $G(x_j)$. If x_j occurs positive in c_i then the other vertex of the edge is $(x_{j,1}, 3)$. Since M is feasible $(x_{j,1}, 2)$ has been matched by the upper truth setting edge and hence x_j has been set true. Otherwise one of $(x_{j,2}, 1), (x_{j,3}, 2)$ is matched to $(c_i, 2)$. So the upper truth setting edge cannot be in the matching M for this would block both $(x_{j,2}, 1)$ and $(x_{j,3}, 2)$ due to the level constraint.

Now we construct from the given satisfying truth assignment of $\mathcal{C}$ a feasible matching of $G(\mathcal{C})$. Take all sandwich edges and all formula edges of the type $((x_{j,3}, 1), (x_j, 2))$. For every variable x_j that has been set true by the truth assignment we add the truth setting edges of $G(x_j)$. At last we choose for every clause c_i a variable x_j that fulfills the clause in the truth assignment and add the corresponding formula edge. These edges altogether obviously form a feasible matching.

$\square$

The reader may wonder why we suggest a heuristic for an NP–complete problem which itself contains an NP–complete subproblem. In our computational practice the trading graphs are small and sparse. Our relaxed recursive enumeration, presented in the following, required less than 1/1000 seconds per iteration on a Sun Sparc 10 to solve this task for the problem set presented in the last section.

In our recursive procedure for each node v at level 1 of the trading graph we enumerate feasible matchings which match this node. This is done by calling FindMaxMatching(v, 0, $\emptyset$).

```
    procedure FindMaxMatching(v = (i, l, k), g, Q)
(1)     for all u ∈ {(i, 1, .), ..., (i, l − 1, .)}
            if u is unmatched then
                add u to Q
        if v is unmatched then
            mark v as matched
(2)         for all unmatched neighbors u of v
                mark u as matched
                FindMaxMatching(u, g + ω(u, v), Q)
        else
            if Q contains an unmatched node w then
(3)             FindMaxMatching(w, g, Q − {w})
            else
                if g > g* then
(4)                 update current matching and g*
(5)             for all unmatched u = (i, h, .) ∈ V with h = 1 ∨ (i, h − 1, .) matched
                    FindMaxMatching(u, g, Q)
            endif
        endif
(6)     mark v as unmatched
    endproc
```

The parameters of procedure *FindMaxMatching* are the node v, the weight g of a partial trading matching constructed so far, and a queue Q containing the nodes that have to be matched in order to make the partial trading matching feasible. Each time we process a node we have to put all unmatched nodes corresponding to the same tour in a lower level into the queue. This is done in (1).

If v is not matched yet we make a recursive call of our procedure for all of its unmatched neighbors, thus matching it in (2). Note, that the partial matching now has to be extended.

If v has already been matched, we get the next unmatched node from our queue and make another recursive call in (3). If no more such node exists we have found a feasible trading matching with weight g. If g improves the current best found matching we update our best solution in (4). Up to now we have only nodes which were “forced” from a node at the first level. In (5) we try to extend our matching M by reducing the trading graph by M and continuing our recursive search for all nodes which are in the first level in the reduced graph. If we reach the end of the procedure we have exhausted the possibilities to match current node and mark it as unmatched (6).

Obviously this method enumerates all possible trading matchings and hence finds the maximum. In our experiments we relaxed this procedure by leaving out (5) to speed up the computation.

4.4 Updating the current tourplan $\mathcal{T}$ according to a trading matching M

It is immediate how to improve a tourplan $\mathcal{T} = (T_1, \ldots, T_t)$ using a trading matching M:

for $i = 1$ **to** t
$\quad l^* = \max\{l \mid (i, l, .) \in M_V\} \cup \{0\}$
$\quad T_i = T_i^{l^*}$
endfor

Since by definition a customer exchange associated with a trading matching preserves feasibility we get:

Theorem 4.2 *The update step yields to a new feasible tourplan $\mathcal{T}\Delta M$ and*

$$c(\mathcal{T}\Delta M) = c(\mathcal{T}) - \omega(M) \ .$$

□

5 Extensions

Now that we have explained the basic concept of Simulated Trading we will present some extensions of the heuristic which increased its performance considerably. The reader may be reminded of the standard concepts of a penalty function, hill climbing and tabu search (see [11, 12]).

5.1 Infeasibility

Sometimes we could improve our current solution much faster allowing infeasibilities. In order to terminate with a feasible solution we penalize infeasibility, thus getting the modified cost function:

$$c^*(T) = c(T) + \mu I(T)$$

where μ is a penalty factor and $I(T)$ is defined to be 0 if T is feasible, otherwise greater than zero. In our experiments we chose $I(T)$ to be the sum of the capacity overload, the violation of a time window and the violation of the maximal tour time. Similarly to [9] we oscillate μ i.e. : if we have maintained a number of feasible solutions we set $\mu = \mu/2$. Otherwise, if the recent solutions all were infeasible we increase the penalty by setting $\mu = 2 * \mu$.

5.2 Deterioration

Our experiments have shown that Simulated Trading is much faster, if we allow trading matchings with negative weights. Therefore, we introduce a lower bound ω^* for the acceptance of a weighted trading matching. The weight of a found weighted matching has to be greater or equal to ω^* or it will not affect the tourplan. We initialize the value with $\omega^* = -0.02 * c(\mathcal{T})$. During the algorithm we continuously increase ω^* until it is positive.
Because of possible deterioration we store the current best tourplan as $\mathcal{T}^*$. If no improvement of the current best tourplan $\mathcal{T}^*$ has been achieved for a number of iterations, we go back and restart from $\mathcal{T}^*$.

5.3 Tabu Search

Because of the allowed deterioration we have to prevent cycling. Therefore we have implemented a hash function $h : \mathbb{P}(V) \to \mathbb{N}$ and store $h(\mathcal{T})$ in a tabu list $\mathcal{L}$ after each update of the tourplan. Moves to a tourplan $\mathcal{T}^*$ with $h(\mathcal{T}^*) \in \mathcal{L}$ are forbidden.
A tabu list length of 10–20 has been used for the computational results and performed quite well.

5.4 Dynamic decision level

If we allow deterioration and we are far away from the value of the current best solution we loose confidence in our current tourplan. Therefore, we do no longer exploit the full decision depth *MaxLevel* and watch out only for quick improvements. We use a new maximal decision level L^* and set it dynamically between 1 and *MaxLevel* according to the difference between the solution value of the current tourplan and the value of the best found solution.

6 Parallelization of Simulated Trading

We have done two different kinds of parallelization:

6.1 Master/Slave Parallelization

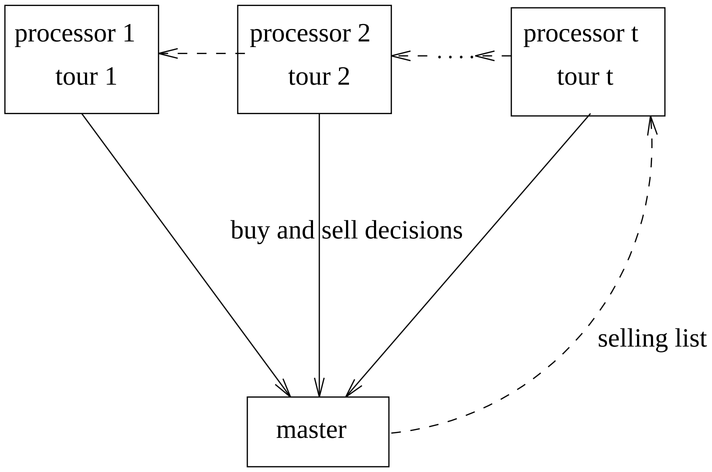


Figure 4: Master/Slave Parallelization.

Our first approach — and actually our first implementation of Simulated Trading at all — was to map each tour onto one slave processor. As it is shown in Figure 4 the slave processors have to make sell and buy decisions and send them to an additional master processor which broadcasts a selling list and searches for a trading matching.
We have implemented this approach in Modula–2 on a transputer cluster with 32 nodes under Helios. The obtained results and experiences have shown that master/slave parallelization works and can improve an initial tourplan quite well if the single tours are big enough. Otherwise, the communication overhead slows down the heuristic significantly. From our experiments we have learned that the slave processors were idle most of the time and waiting for the master transputer.

In this — our first — implementation we did not explicitly use any randomness, it turned out to occur implicitly from the asynchronous communication.

6.2 Stock Parallelization

Due to our former experiences and the common belief that in the nearby future parallel machines will be coupled networks of computers we looked for a parallel approach using less communication. The idea of the stock parallelization, shown

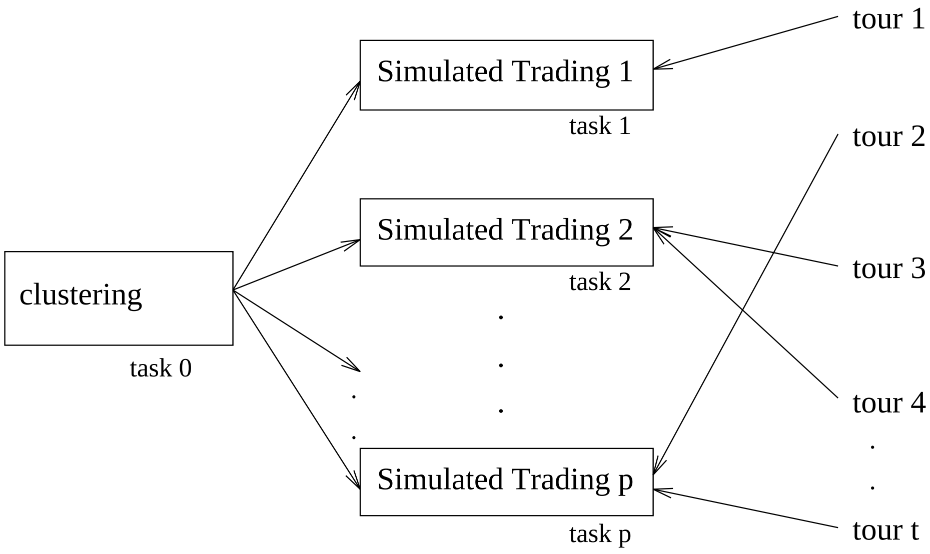


Figure 5: Stock Parallelization.

in Figure 5, is to partition the current tourplan, such that each task gets about the same number of tours and to start a Simulated Trading algorithm on each task in parallel. A few remarks have to be made on that:

- The partitioning has to be done dynamically. After some time (1–3 seconds) depending on the problem size and the total runtime we have to collect the actual tours from all tasks to repartition them.
- In a reasonable partition, tours which are — geometrically — close should be mapped onto the same task.
- The partitioning has to be done fast.

Obviously this parallel approach is useful only if the problems (number of tours) are large enough.
For our computational results we have used PVM [7] for the network communication and the following *farthest customer sweep* clustering (see Figure 6 for an example clustering):

> Let $\mathcal{T} = (T_1, \ldots, T_t)$ be the current tourplan
> **for** $i = 1$ **to** t
>   Calculate the farthest customer f_i of tour T_i
> Sort $(f_1, \ldots, f_t)$ by polar angles to the depot
> Select a random customer of the sorted list $(f_1^*, \ldots, f_t^*)$ and make m clusters of customers along ascending polar angles

Independently from the authors É. Taillard presented two partition methods which he used to speed up an iterative tabu search method applied to vehicle routing problems with capacity constraints (see [22] for details).

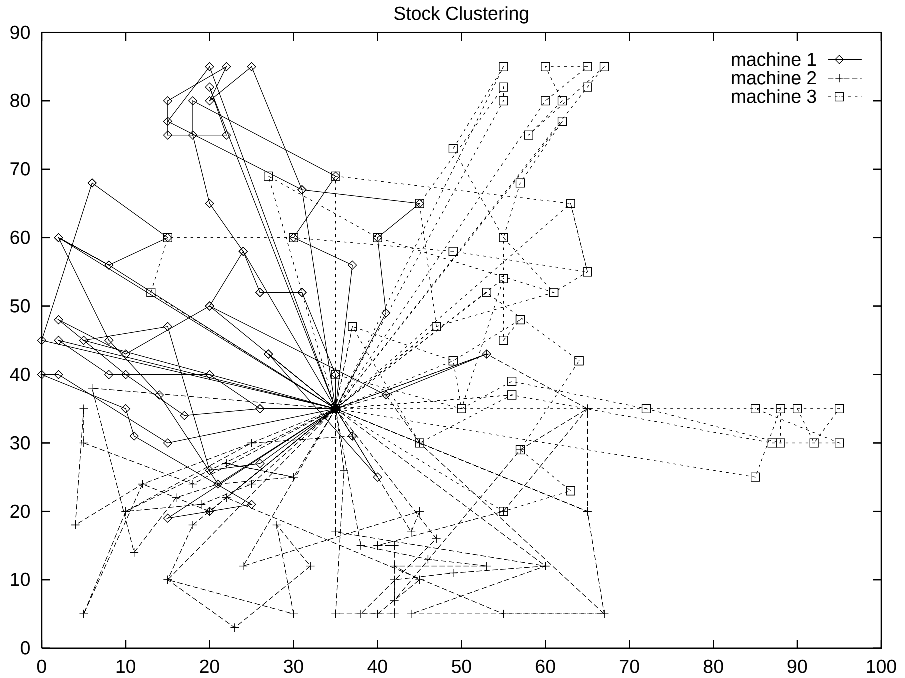


Figure 6: Clustering of a tourplan with 200 customers and time window constraints.

7 Computational Results

Our sequential computational results were done on two different sets of test problems taken from the literature while the parallel approach was tested on a set of new generated instances. The first set we have used was randomly generated by Solomon [19, 20] and includes vehicle routing problems with time windows, service times, capacity constraints and a restriction on the maximal tour time. Each problem consists of 100 customers. The second problem set was first published by Christofides and Eilon [3] and later extended by Christofides et al. [4] and consists of fourteen problems of size 50–199 customers with various restrictions. Table 1 gives a summary on the appearance of side constraints.

All distances are calculated as Euclidean distances using the given coordinates of the customers resp. the depot. Unfortunately, not all publications contain information about the used rounding technique and it's obvious that truncated

additional constraints	Solomon all problems	Christofides et al. 1,2,3,4,5,11,12	6,7,8,9,10,13,14
capacity	x	x	x
service times	x		x
tour duration time	x		x
time windows	x		

Table 1: Vehicle routing problems and their additional constraints.

distances can lead to significantly better results. Therefore all results in this paper were obtained using floating point arithmetic without rounding or truncating any distances or times. The problem libraries and our obtained solutions are available from the authors.

7.1 Vehicle Routing Problems with Time Windows

Conforming to other published results we have used the minimization of the number of tours as a first objective and the reduction of the total time as a second goal. Thus a solution which needs more time than a second solution might be better if it uses less tours.

The Tables 2–5 show *average* values for Solomon's test sets R1, R2, RC1 and RC2 and allow a comparison between Simulated Trading and other heuristics. The following heuristics are compared (we have omitted some older heuristics like Savings or Sweep because their results are not better than Solomon's heuristic (see [19])):

Solomon: a parallel insertion heuristic by M. Solomon (see [19])

Potvin and Rousseau: a parallel route building heuristic by J. Y. Potvin and J. M. Rousseau (see [17])

GRASP_Routing: a randomized adaptive search procedure by G. Kontoravdis and J. F. Bard (see [13])

Tabu Search: a tabu search method by J. Y. Potvin, T. Kervahut, B. L. Garcia and J. M. Rousseau (see [16])

"Tabu Search 2" resp. "Simulated Trading 2" are the best solutions of two runs. "Tabu Search Best" resp. "Simulated Trading Best" are the best results which have been obtained during the experimentation phase.

We have to mention that each run of Simulated Trading was done on the entire set of problems R1, R2, RC1 resp. RC2 without changing any parameter of the heuristic. Because of the different computers that were used we give only the computation times for Simulated Trading and Tabu Search which are both obtained on a Sun Sparc 10. We can clearly point out that the solutions found by Simulated Trading outperform the solutions of the other heuristics.

In Figure 7 you can see typical improvement curves of the total tour time. We can clearly state that Simulated Trading does more than 90% percent of its improvement in the first 50–100 seconds. Whereas Figure 7 shows only improvements of the total route time, Figure 8 contains the change of the cost function after each accepted matching and the infeasibility of the current solution.

R1: 12 Problems	#Tours	Route Time	Time (min:sec)
Initial Solution	13.5	1697.0	—
Solomon	13.5	1685.4	—
Potvin and Rousseau	13.3	—	—
GRASP_Routing	13.1	—	—
Tabu Search	12.92	1417.3	4:33
Tabu Search 2	12.83	1412.5	9:58
Tabu Search best	12.75	1403.9	—
Simulated Trading	12.75	1427.7	5:58
Simulated Trading 2	12.75	1417.0	10:11
Simulated Trading best	**12.58**	**1392.0**	—

Table 2: 12 test problems R1.

R2: 11 Problems	#Tours	Route Time	Time (min:sec)
Initial Solution	3.27	1555.8	—
Solomon	3.27	1555.0	—
Potvin and Rousseau	3.1	—	—
GRASP_Routing	3.1	—	—
Tabu	3.18	1260.9	5:24
Tabu 2	3.09	1242.4	11:14
Tabu best	3.09	1198.3	—
Simulated Trading	3.09	1262.3	5:21
Simulated Trading 2	3.09	1249.5	6:04
Simulated Trading best	**3.00**	**1199.6**	—

Table 3: 11 test problems R2.

RC1: 8 Problems	#Tours	Route Time	Time (min:sec)
Initial Solution	13.5	1774.0	—
Solomon	13.5	1772.2	—
Potvin and Rousseau	13.4	—	—
GRASP_Routing	12.8	—	—
Tabu	12.88	1539.2	3:25
Tabu 2	12.75	1527.1	7:26
Tabu best	12.75	1515.4	—
Simulated Trading	12.63	1534.6	4:30
Simulated Trading 2	12.5	1532.0	9:30
Simulated Trading best	**12.13**	**1501.6**	—

Table 4: 8 test problems RC1.

RC2: 8 Problems	#Tours	Route Time	Time (min:sec)
Initial Solution	4.00	1976.0	—
Solomon	4.00	1982.2	—
Potvin and Rousseau	3.6	—	—
GRASP_Routing	3.6	—	—
Tabu	3.62	1598.3	4:30
Tabu 2	3.50	1582.4	9:18
Tabu best	**3.38**	**1477.6**	—
Simulated Trading	3.38	1541.4	3:26
Simulated Trading 2	3.38	1512.0	8:19
Simulated Trading best	3.38	1500.1	—

Table 5: 8 test problems RC2.

Problem	#Tours	Route Time	Problem	#Tours	Route Time
R101	19	2398.0	R201	4	1975.3
R102	17	1991.7	R202	3	1446.9
R103	14	1570.1	R203	3	1286.2
R104	10	1061.9	R204	3	962.7
R105	14	1596.0	R205	3	1263.7
R106	12	1356.9	R206	3	1094.0
R107	11	1145.3	R207	3	1026.7
R108	10	989.1	R208	3	825.9
R109	12	1262.1	R209	2	1147.2
R110	11	1162.9	R210	3	1236.2
R111	11	1164.4	R211	3	931.2
R112	10	1005.2	—	—	—
RC101	15	1927.1	RC201	4	2073.1
RC102	13	1671.5	RC202	4	1767.4
RC103	11	1361.6	RC203	3	1407.9
RC104	10	1196.2	RC204	3	1065.0
RC105	15	1847.0	RC205	4	1966.2
RC106	12	1491.7	RC206	3	1416.6
RC107	11	1199.8	RC207	3	1322.6
RC108	11	1198.9	RC208	3	1029.2

Table 6: The best results of Simulated Trading.

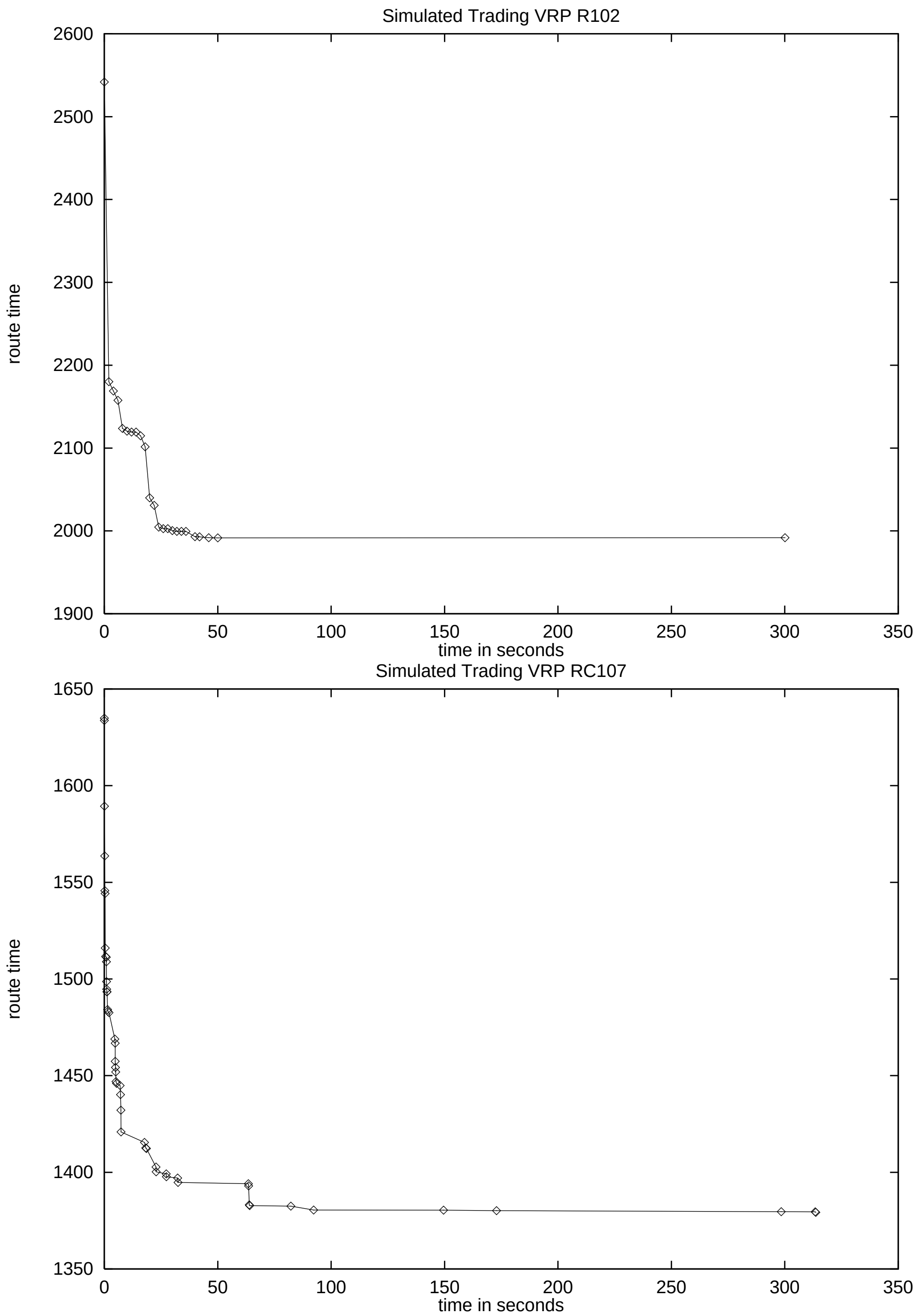


Figure 7: Typical improvement curves of Simulated Trading.

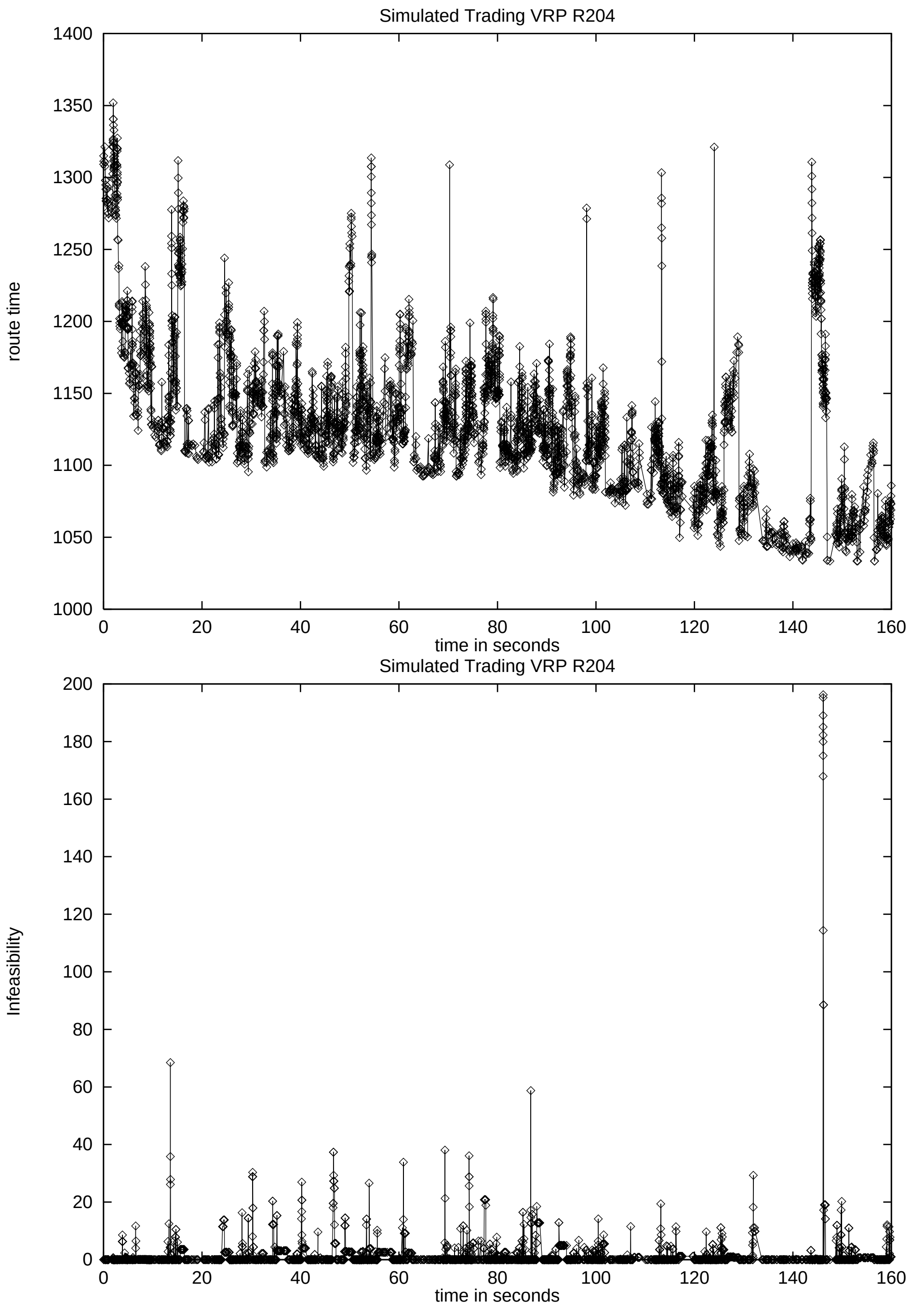


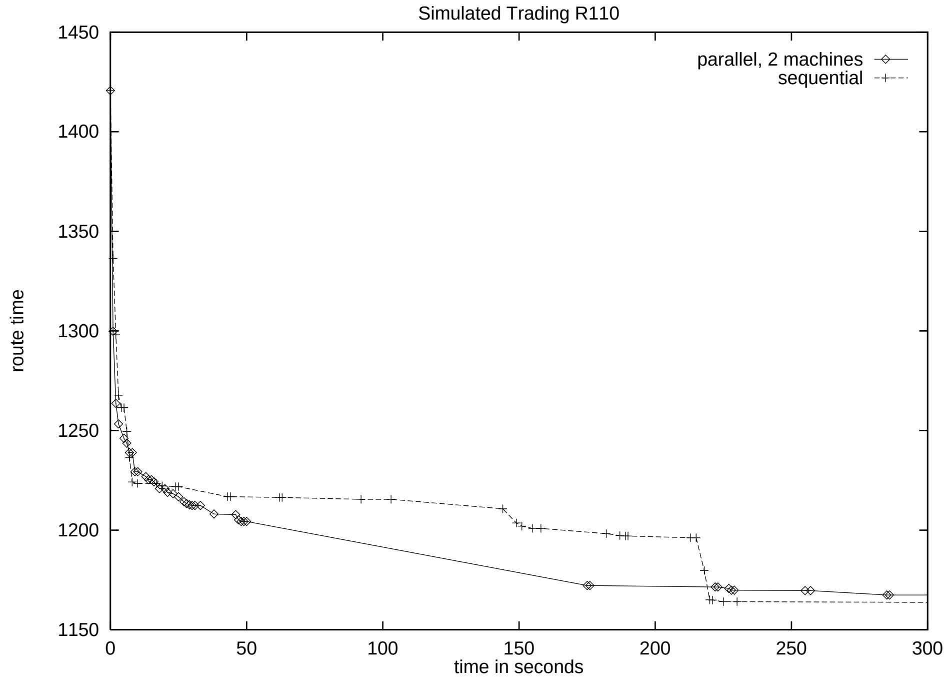
Figure 8: Typical tour cost resp. infeasibility curves of Simulated Trading.

We will now present some computational results for the stock parallelization of Simulated Trading. Solomon's problem set was not well suited for our parallel approach. Figure 9 shows a comparison of two typical runs of parallel Simulated Trading on two workstations and sequential Simulated Trading. The solution values obtained by sequential Simulated Trading could not be improved by distributing the problem among two or more workstations. The reason for that is the small problem size concerning the number of tours. Therefore we have build 8 new vehicle routing problems by concatenating in each case one R1 problem with one problem of the RC1 problem set. The vehicle capacities and the maximal tour time were taken from problem set R1 accept for the problem P5 where we had to extend the maximal tour time to 250 units in order to get a feasible tourplan.

The values given in Table 7 are the best results of 3 runs of Simulated Trading on the new generated test problem P1–P8 each containing 200 customers. Clearly the parallel Simulated Trading on a network of Sun Sparcs outperforms the sequential approach. Figure 10 shows that parallel Simulated Trading leads in a shorter time to better results.

	sequential		parallel 2		parallel 3	
Prob.	#Tours	Route Time	#Tours	Route Time	#Tours	Route Time
P1	33	6110	33	5995	32	5865
P2	31	5591	29	5290	29	5246
P3	24	4584	24	4517	23	4448
P4	21	4094	19	3976	19	3972
P5	27	5180	27	5087	26	4956
P6	23	4451	24	4428	23	4347
P7	22	4184	21	4113	21	4074
P8	19	3891	19	3804	19	3792
avrg.	25	4760	24.5	4651	24	4587

Table 7: Results for the new vehicle routing problems P1–P8.



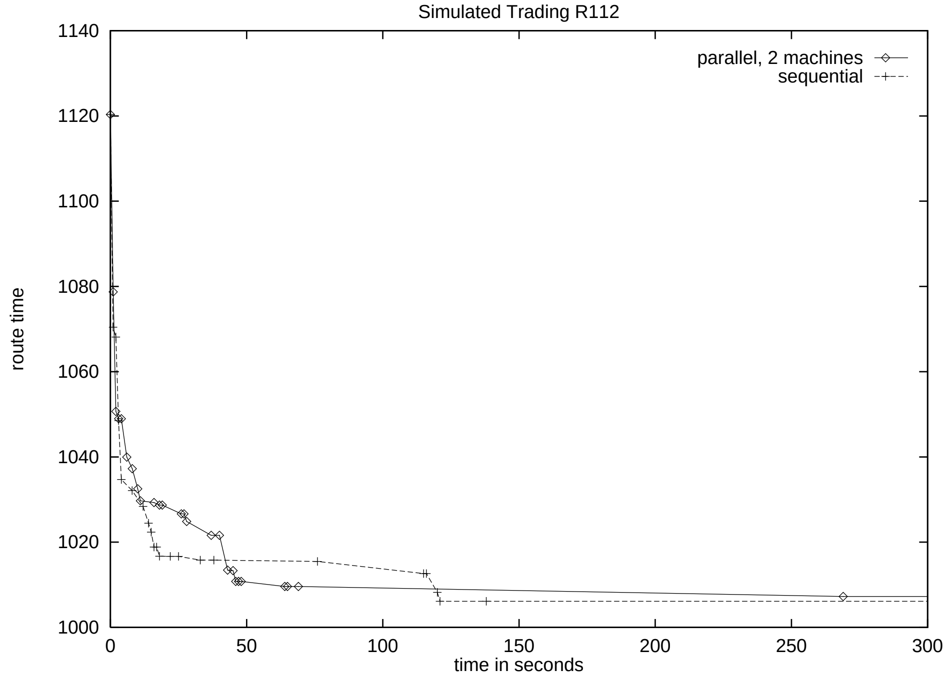


Figure 9: A comparison of sequential and parallel Simulated Trading on a 100 customer problem.

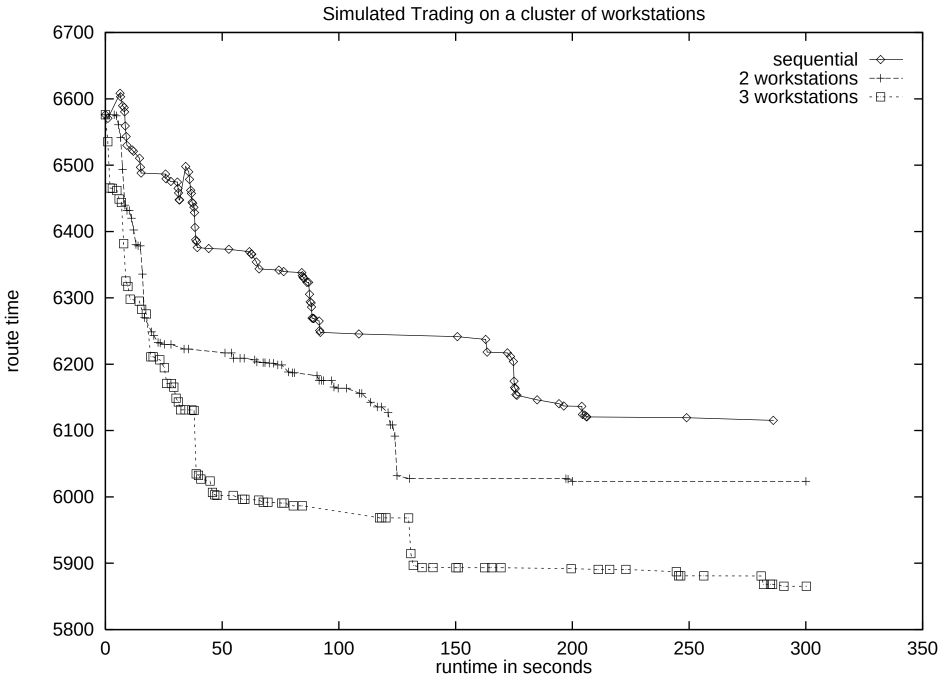


Figure 10: A comparison of parallel and sequential Simulated Trading on the 200 customer problem P1.

7.2 Standard Vehicle Routing Problems

The following computational results were done using fourteen VRP instances published in [4]. The objective is to minimize the total distance using an infinite number of vehicles. Because the standard vehicle routing problem without time windows behaves locally like the well–known travelling salesman problem we have implemented a 3–Opt improvement procedure which optimizes every tour each time a new best solution has been found. Our initial solutions were generated using a simple sweep heuristic.

Table 8 shows a comparison between Simulated Trading and various heuristics from the literature:

SAV: the savings algorithm by G. Clarke and J. W. Wright (see [5])

SW: the sweep heuristic by B. E. Gillet and L. R. Miller (see [10])

TREE: the incomplete tree search algorithm by N. Christofides, A. Mingozzi and P. Toth (see [4])

SA: simulated annealing algorithm by I. H. Osman (see [15])

TS: the tabu search algorithm by É. Taillard (see [22])

TR: the tabu search algorithm by M. Gendreau, A. Hertz and G. Laporte (see [9])

The performance measurement of a randomized algorithm like Simulated Trading is difficult because the results of two runs are often not the same. Thus we chose to perform each run a couple of times and then to take the average over all solution values. Column *avrg* in Table 8 gives the average results over five runs of Simulated Trading. The column *std* resp. *best* gives the results of one single run resp. the best solution values obtained in any run during the experimentation phase. Furthermore we have a strong believe that the solution value 816 obtained by **Tree** with problem 12 was obtained with truncated distances.

Our initial solutions obtained with a simple sweep heuristic were rather poor with an average solution value of 1124. The results of Simulated Trading are better than the solutions of some older heuristics as **Saving**, **Sweep** and **Tree** but they cannot compete with the results of the specialized tabu search heuristics **TS** resp. **TR**. This is not very surprising since Simulated Trading does not exploit the geometrical properties of the standard vehicle routing problem.

	SAV	**SW**	**TREE**	**SA**	**TS**	**TR**		**ST**	
						std	*best*	*avrg*	*best*
1	585	532	534	528	524.61*	524.61*	524.61*	524.6	524.61*
2	900	874	871	838.62	835.32*	835.77	835.32*	837.9	835.32*
3	886	851	851	829.18	828.98	829.45	826.14*	831.7	829.42
4	1204	1079	1064	1058	1029.64*	1036.16	1031.07	1044.9	1037.45
5	1540	1389	1386	1378	1300.89*	1322.65	1311.35	1345.1	1322.69
6	619	560	560	555.43*	555.43*	555.43*	555.43*	556.9	555.43*
7	976	933	924	909.68*	909.68*	913.23	909.68*	917.5	909.68
8	973	888	885	866.75	865.94*	865.94*	865.94*	872.6	868.39
9	1426	1230	1217	1164.12	1164.24	1177.76	1162.89*	1190.2	1169.93
10	1800	1518	1509	1417.85	1403.21*	1418.51	1404.75	1457.5	1438.84
11	1079	1266	1092	1176	1073.05	1073.47	1042.11*	1165.6	1122.78
12	831	937	816?	826	819.56*	819.56*	819.56*	820.2	819.56*
13	1634	1770	1608	1545.98	1550.15	1573.81	1545.93*	1605.2	1562.12
14	877	949	878	890	866.37*	866.37*	866.37*	879.2	867.17
avrg	1095	1055	1014	999	981	987	979	1004	990

Table 8: Comparison of Simulated Trading with other heuristics.

Due to the different computer hardware we omit a comparison of running times. However Table 9 shows our average computation time in minutes on a Sun Sparc 10 to find the solutions given in column *avrg* of Table 8. The total times are the same as given in [9] by Gendreau et al. who used a Silicon Graphics Workstation to obtain their results.

Problem	Size	Average computation times over 5 runs (in minutes)	
		To obtain the best solution	Total time
1	50	0:23	6
2	75	40	53.8
3	100	15	18.4
4	150	58	58.8
5	199	90	90.9
6	50	2	13.5
7	75	22	54.6
8	100	10	25.6
9	150	70	71
10	199	90	99.8
11	120	20	22.2
12	100	10	16
13	120	45	59.2
14	100	40	65.7

Table 9: Computation times of Simulated Trading on a Sun Sparc 10.

8 Conclusions

We have presented a new iterative search heuristic which is based on a complex neighborhood structure. The results presented in this paper show that Simulated Trading produces high quality solutions for the vehicle routing problem with time windows and good solutions for standard vehicle routing problems. Due to the nature of Simulated Trading it will produce the best results for problems which contain a difficult structure like time windows which disturb any geometrical approach. Because of the meta structure of Simulated Trading additional constraints like heterogeneous vehicle fleet, pickup–and-delivery,... can be handled without any changes. The heuristic is well suited for parallelization consuming only few communication resources. Clustering methods and results for the parallel implementation on a cluster of workstations have been given.

Although we have only presented computational results for solving vehicle routing problems Simulated Trading is capable to solve other clustering problems as well e.g. capacitated minimum spanning tree and bin packing.

Acknowledgements
The authors would like to thanks Marius M. Solomon for sending us his vehicle routing problem library and Matthias Middendorf for some help in designing the proof of theorem 4.1.

References

[1] C. Berge, *Graphs*, vol. 6 of Mathematical Library, North–Holland, 1985.

[2] L. Bodin and B. Golden, *Classification in vehicle routing and scheduling*, Networks, 11 (1981), pp. 97–108.

[3] N. Christofides and S. Eilon, *An algorithm for the vehicle–dispatching problem*, Operations Research Quarterly, 20 (1969), pp. 309–318.

[4] N. Christofides, A. Mingozzi, and P. Toth, *The vehicle routing problem*, in Combinatorial Optimization, N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, eds., Wiley, 1979, ch. 11, pp. 315–338.

[5] G. Clarke and J. W. Wright, *Scheduling of vehicles from a central depot to a number of delivery points*, Operations Research, 12 (1964), pp. 568–581.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP–Completeness*, Freeman, San Francisco, 1979.

[7] A. Geist, A. Beguelin, J. Dongarra, W. Jaing, R. Manchek, and V. Sunderam, *PVM 3.0 user's guide and reference manual*, Tech. Rep. ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, 1993.

[8] M. Gendreau, A. Hertz, and G. Laporte, *New insertion and post–optimization procedures for the traveling salesman problem*, Operations Research, 40 (1992).

[9] M. Gendreau, A. Hertz, and G. Laporte, *A tabu search heuristic for the vehicle routing problem*, Tech. Rep. 777, Centre de Recherche sur les Transports, Université de Montréal, C.P. 6128, Succ. "A", Montréal (Québec), Canada H3C 3J7, November 1993.

[10] B. E. Gillett and L. R. Miller, *A heuristic for the vehicle–dispatch problem*, Operations Research, 22 (1974), pp. 340–349.

[11] F. Glover, *Tabu search part 1*, ORSA Journal on Computing, 1 (1989), pp. 190–206.

[12] ——, *Tabu search part 2*, ORSA Journal on Computing, 2 (1989), pp. 4–32.

[13] G. Kontoravdis and J. F. Bard, *Improved heuristics for the vehicle routing problem with time windows*, tech. rep., Operations Research Group, Department of Mechanical Engineering, The University of Texas, Austin, TX78712-1063, Sept. 1992.

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

[15] I. H. Osman, *Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem*, Annals of Operations Research, 41 (1993), pp. 421–451.

[16] J. Y. Potvin, T. Kervahut, B. L. Garcia, and J. M. Rousseau, *A tabu search heuristic for the vehicle routing problem with time windows*, tech. rep., Centre de Recherche sur les Transports, Université de Montréal, C.P. 6128, Succ. "A", Montréal (Québec), Canada H3C 3J7, 1993.

[17] J. Y. Potvin and J. M. Rousseau, *A parallel route building algorithm for the vehicle routing and scheduling problem with time windows*, European Journal of Operational Research, 66 (1993), pp. 331–340.

[18] M. W. P. Savelsbergh, *An efficient implementation of local search algorithm for constrained routing problems*, European Journal of Operational Research, 47 (1990), pp. 75–85.

[19] M. M. Solomon, *Algorithms for the vehicle routing and scheduling problems with time window constraints*, Operations Research, 35 (1987), pp. 254–265.

[20] ——, *Development of vehicle routing problems with time windows*. 1987.

[21] M. M. Solomon and J. Desrosiers, *Time window constrained routing and scheduling problems*, Transportation Science, 1 (1988), pp. 1–13.

[22] E. Taillard, *Parallel iterative search methods for vehicle routing problems*, Networks, 23 (1993), pp. 661–673.

[23] A. Wren and A. Holliday, *Computer scheduling of vehicles*, Operations Research Quarterly, 23 (1972), pp. 333–344.