# A parallel local search algorithm for the travelling salesman problem

Eindhoven University of Technology

Department of Mathematics and Computing Science

# A Parallel Local Search Algorithm for the Travelling Salesman Problem

by

Edwin van der Sluis

Computing Science Note 91/12
Eindhoven, July 1991

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515


Editors: prof.dr.M.Rem
prof.dr.K.M. van Hee

# A Parallel Local Search Algorithm for the Travelling Salesman Problem

Edwin van de Sluis
Dept. of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

**Abstract**

A parallel local search algorithm for the Travelling Salesman Problem is presented. A TSP solution is distributed over an array of processors such, that transitions can be carried out in parallel, while communication is reduced to a minimum. Simulated annealing is used as optimization technique. It is shown that the algorithm satisfies the asymptotic convergence conditions. Performance figures for three TSP instances show a linear speed-up, at the cost of a small decrease in the quality of the final solutions.

# 1 Introduction

In the $N$-city Travelling Salesman Problem, or TSP, we are given a number of cities $0, \ldots, N-1$ and for each pair of cities $i, j$ a distance $D(i,j)$. Our goal is to find a tour $T(i : 0 \leq i < N)$ that visits each city exactly once and that minimizes:

$$(\sum i : 0 \leq i < N : D(T(i), T(i+1))) + D(T(N-1), T(0)).$$

Here $T(i)$ denotes the $i^{th}$ city being visited. We will concentrate on the *symmetric* TSP in this paper, where the distances satisfy $D(i,j) = D(j,i)$ for $0 \leq i, j < N$.

The TSP can be considered as the prototypical NP-hard problem, so polynomial-time algorithms for finding optimal solutions are unlikely to exist. Therefore, much effort has gone into the design of efficient *approximation algorithms*, that attempt only to find near-optimal solutions.

The best approximation algorithms for the TSP have been based on a general technique known as *local search*. In order to derive a local search algorithm for a combinatorial optimization problem, one defines a *neighbourhood structure* that specifies for each solution a set of solutions that can be obtained by applying some pre-defined transition mechanism. Given this a neighbourhood structure, a local search algorithm operates as follows. We start off with a given initial solution, which is typically chosen randomly. Next we search the neighbourhood structure of the current solution to find a better solution (a solution of lower cost). If such a solution is found, we replace the current solution by this solution. If such a solution does not exist the algorithm terminates with the current solution, which is identified as *locally optimal*.

The most famous local search algorithms for the TSP are the '2-opt', the '3-opt', and the 'Lin-Kernighan' algorithms [Lin65, Lin73]. The corresponding neighbourhood structures are as follows (cf. [Joh90]):

**2-Opt** Two tours are neighbours if one can be obtained from the other by replacing two edges by two other edges such that again a tour is obtained.

**3-Opt** Two tours are neighbours if one can be obtained from the other by replacing *three* edges by three other edges such that again a tour is obtained.

**Lin-Kernighan** Two tours are neighbours if one can be obtained from the other by performing a sequence of edge changes, going to arbitrary depth, with the exponential explosion of the neighbourhood controlled by a complex greedy criterion. See [Lin73] for details.

In [Joh90] Johnson surveys the state of the art with respect to the TSP, with emphasis on the performance of the three local search algorithms mentioned above, and two "new" competitors: *simulated annealing* and *genetic algorithms*. These approaches can be considered as variants on local search. With simulated annealing [Kir83, Laa87] we occasionally

replace the current solution by a *worse* solution, thus allowing us to escape from local optima. With the genetic approach [Müh88] we perform some fixed number of independent runs (possibly in parallel) with some traditional local search algorithm, and then derive new starting solutions by "mating" the solutions found.

The experiments of Johnson show that the local search algorithms give high-quality solutions, with the 3-opt and Lin-Kernighan algorithms as the champions. A drawback is that to require these high-quality solutions, often large computation times are required, especially when simulated annealing is used.

One way to obtain a substantial decrease in computation time is to parallelize the local search algorithms. In this paper we present such a parallel algorithm. We use simulated annealing as local search algorithm and the 2-opt as neighbourhood structure. It is well-known that the transition mechanism that underlies this neighbourhood structure (the 2-change) can be applied locally to tours (see e.g. [Lin73]).

We begin this paper with a description of the simulated annealing algorithm and our approach to its parallelization. Next we describe how the 2-change transition mechanism can be parallelized to support this approach. In Section 4 we show that the resulting algorithm satisfies the so-called asymptotic convergence conditions. This means that our simulated annealing algorithm converges asymptotically to the set of optimal tours, for any TSP instance. Readers that are only interested in (parallel) local search can skip this section. We implemented the algorithm on a network of 50 (T800) transputers. Computational results of our experiments on three TSP instances are given in Section 5. These results are discussed, and we end this paper with some conclusions and ideas for further research.

# 2 Parallelization of simulated annealing

At a first glance, the general simulated annealing algorithm does not seem to have any 'parallel features'. Basically, the algorithm works as follows. Suppose we have a starting solution to the optimization problem we are interested in. Then simulated annealing operates by continuously attempting to replace this solution by a new one. This new solution is generated from the current one by applying a predefined transition mechanism. A transition is accepted with a certain probability which depends on two factors: the cost difference between the two solutions, and the current value of a control parameter (traditionally called the temperature). The control parameter is decreased after a number of transitions.

When designing a parallel annealing algorithm, we distinguish two general strategies [Aar89]: *single-trial parallelism* and *multiple-trial parallelism*. In single-trial parallelism, we divide the work involved in evaluating a single trial (transition) over a number of processors, while in multiple-trial parallelism, different trials are evaluated in parallel.

Within the class of multiple-trial parallel annealing algorithms, two subclasses are distinguished: *general algorithms* and *tailored algorithms*. We can either decide to design a generally applicable parallel algorithm, or decide to design a parallel algorithm that is

tailored to a certain problem. Our goal is a tailored annealing algorithm for the TSP, with multiple-trial parallelism.

To derive such an algorithm, let us have a closer look at the general annealing algorithm. We see that the algorithm consists of four major steps (cf. [Aar89]):

1. propose a new solution by applying the transition mechanism,
2. calculate the cost difference between the old and the new solution,
3. decide (probabilistically) whether the new solution will be accepted, and
4. if the transition is accepted, substitute the old solution by the new one.

Suppose now that we can distribute a solution over the network such, that each processor can perform all steps in parallel, i.e., without communication with any of its neighbouring processors. Then we see that each processor can execute the annealing algorithm locally. However, two important aspects have to be considered before we can think about an implementation.

- The solution distribution has to be such that a large variety of transitions can be proposed locally. To apply simulated annealing successfully, we need such a large variety, otherwise we cannot expect annealing to find near-optimal solutions.
- The communication between the processors. If *no* communication takes place, each processor will obtain a near-optimal *subsolution*, which will typically not imply that a good *global* solution is found. Therefore, processors have to communicate parts of the solution to avoid this.

We see that our approach to parallelization amounts to a parallelization of the neighbourhood structure. We like to stress that our restriction here to simulated annealing is not essential, and that our approach can be used for other local search algorithms.

In the next section we show how the 2-opt neighbourhood structure of the TSP can be parallelized. From the discussion above we conclude that such a parallelization consists of (1) a distribution of a tour such, that a powerful transition mechanism can be applied locally by processors, and (2) a communication procedure to exchange parts of a tour.

# 3 Parallelization of the 2-opt neighbourhood structure

As explained earlier, a solution to a TSP instance is a *tour*; a closed walk that visits each city exactly once. Let $N$ be the number of cities, $T$ a tour, and let $T(j)$ denote the $j$-th city being visited. Furthermore, let $P$ be the number of processors and assume that $N$ is such that $N = 2k \cdot P$, for some $k \geq 1$. We assume that the $P$ processors are configured in an array, i.e., a processor $p$, $0 < p < P-1$, has two neighbours; processors $p-1$ and $p+1$.

We first give the *initial* distribution of a tour $T$. Each processor $p$ is given two parts of $T$, denoted by $T.part0_p$ and $T.part1_p$. Initially, each part consists of $k$ consecutive cities, as follows:

$$\forall p : 0 \leq p < P : \quad \forall i : 0 \leq i < k : \quad T.part0_p(i) = T(p \cdot k + i) \wedge \\ T.part1_p(i) = T(N - 1 - (p \cdot k + i))$$

In Figure 1, we give an example of this distribution over an array network of $P = 4$ processors, and with $N = 32$ cities (so, $k = 4$). For example, processor 1 obtains parts $T(4 \leq i < 8)$ and $T(24 \leq i < 28)$ of the global tour $T$.

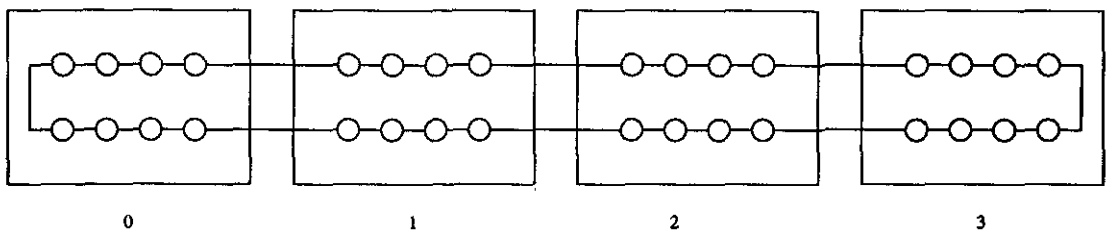


Figure 1: Initial distribution for $N = 32$, $P = 4$

A powerful transition mechanism for the TSP is the so-called 2-change. It is often used in combination with the annealing algorithm (see for instance [Laa87]). Informally, a 2-change transition on a tour $T$ can be described as the exchange of two existing connections in $T$ (each between two cities) by two that are not part of $T$. This of course under the restriction that $T$ remains a tour.

The main advantage of our tour distribution and the 2-change mechanism is that if we combine them, each processor can perform local 2-change transitions, without any communication to exchange cities or calculate cost differences.

To make this more clear, we show in Figure 2 a number of possible 2-change transitions on our initial tour of Figure 1.

Note that in the above example, we can extend the network to eight processors, thereby decreasing the number of cities per processor to four. As is shown in Figure 3, now eight processors can perform a 2-change transition. However, this transition is the only 2-change possible, since we prohibit communication when performing 2-changes.

We may conclude from this, that for larger values of $P$ more communication has to take place. This brings us automatically to the discussion of the communication behaviour of our algorithm. We decided to perform a so-called *rotate* operation after a fixed number of proposed transitions. The rotate procedure can be specified as follows:

$$\{pre : T = T'\}$$

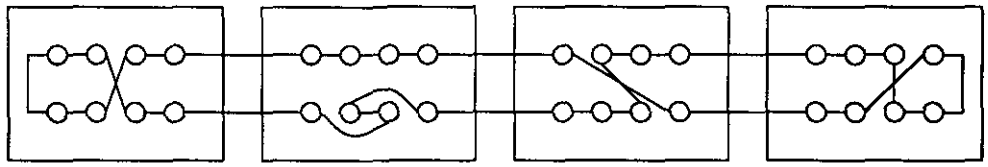

Figure 2: Four possible 2-change transitions

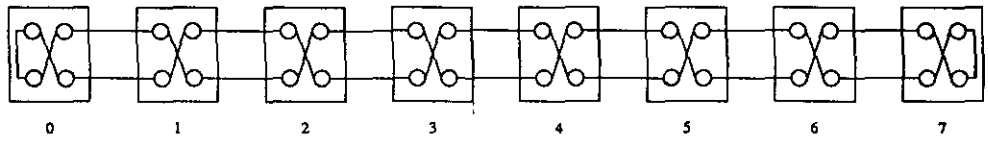


Figure 3: Possible 2-changes when $P = 8$

$$Rotate(T)$$

$$\{post : \forall i : 0 \leq i < N : T(i) = T'(i-1)\}$$

Here $T$ and $T'$ denote tours and subtraction is modulo $N$. As an example, we show in Figure 4, which cities of the tour given by Figure 2 are communicated between the processors. These cities are depicted as black dots.

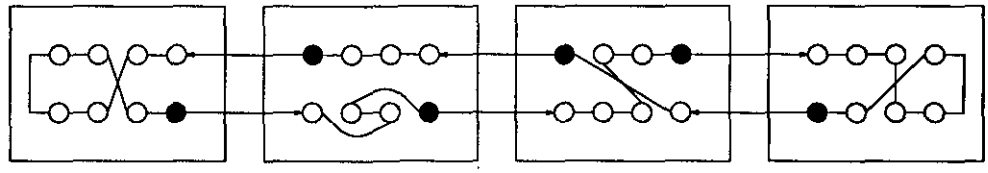

Figure 4: An example of a rotate operation

We end this section by giving the simplified program for an arbitrary processor $p$ in the network. Remember that each processor has two parts of the global tour $T$, which are denoted by $T.part0$ and $T.part1$, which both consist of $k$ cities. The program consists of threenested loops. The value of the control parameter $c$ is decreased after a total number of $L$ (global) transitions have been proposed. A rotation takes place after every $2k$ (local) transitions. The program below is given in a Pascal-like notation.

```
Initialize(T.part0, T.part1, c, L) ;
```

```
while c > minc do
  begin
   l := 0 ;
   while l < (L/P) / 2k do
     begin
      i := 0 ;
      while i < 2k do
        begin
         ProposeTransition(T.part0, T.part1, dC) ;
         if Accept(dC, c)
         then AcceptTransition(T.part0, T.part1, dC) ;
         i := i + 1
        end ;
      Rotate(T.part0, T.part1) ;
      l := l + 1
     end ;
   Decrease(c)
  end
```

# 4 Asymptotic convergence

It is well-known that under certain conditions the simulated annealing algorithm converges asymptotically to the set of optimal solutions (see e.g., [Laa87]) .

In practice, the only condition that has to satisfied is the connectivity of the neighbourhood structure, i.e., that it is possible, for every two solutions $i$ and $j$ to construct a finite sequence of transitions that transforms $i$ into $j$. For the application of the sequential annealing algorithm to the TSP, this property is proved by several authors (see e.g., [Aar89]).

In this section, we show that our parallel annealing algorithm satisfies the convergence condition by showing that it is possible, for every two tours $T$ and $T'$, to construct a finite sequence of transitions *and* rotations from $T$ to $T'$.

Before we show that the convergence property is satisfied, we first give a more formal description of the 2-change transition mechanism that we described in the previous section.

Recall from Section 3 that each processor is given two parts of the global tour $T$, which are called $T.part0$ and $T.part1$. In general, the parts consist of $k0$ and $k1$ consecutive cities respectively, where $k0, k1 \geq 2$ and $k0 + k1 = 2k$.

A 2-change transition by an arbitrary processor on its $T.part0$ and $T.part1$ is now defined by the following functional specification:

$$\begin{aligned}\{pre: \quad & T.part0 = T.p0 \wedge \\ & T.part1 = T.p1 \wedge \\ & 0 < i0 < k0 \wedge 0 < i1 < k1\}\end{aligned}$$

$$2\_change(T.part0, T.part1, i0, i1)$$

$$\begin{aligned}\{post: \quad & T.part0(0 \leq i < i0) = T.p0(0 \leq i < i0) \wedge \\ & T.part0(i0 \leq i < i0 + (k1 - i1)) = T.p1(i1 \leq i < k1) \wedge \\ & T.part1(0 \leq i < i1) = T.p1(0 \leq i < i1) \wedge \\ & T.part1(i1 \leq i < i1 + (k0 - i0)) = T.p0(i0 \leq i < k0)\}\end{aligned}$$

Note that application of a 2-change changes the lengths of parts $T.part0$ and $T.part1$ to $i0 + (k1 - i1)$ and $i1 + (k0 - i0)$ respectively. Note also that the *sum* of the lenghts remains $2k$.

Now, consider processor $P - 1$, which is the last one in the array of processors. For this processor, city $T.part1(k1 - 1)$ is the immediate successor of city $T.part0(k0 - 1)$ in the global tour. Then from the above specification of the 2-change, we see that application of

$$2\_change(T.part0, T.part1, k0 - 1, k1 - 1)$$

performed by processor $P - 1$ has the effect that cities $T.part0(k0 - 1)$ and $T.part1(k1 - 1)$ are swapped in the global tour.

Now, we give a procedure that transforms an aribitrary tour $T$ into another tour $T'$. This procedure has the following invariant $I0$:

$$I0 : \{T(0 \leq i < n) = T'(0 \leq i < n)\}$$

Invariant I0 states that the first $n$ elements of $T$ and $T'$ are equal. Then we know that there exists some $s$, $n \leq s < N$ such that $T(s) = T'(n)$. This is stated as a loop invariant $I1$ for the inner loop. This inner loop 'bubbles' city $T(s)$ to its right position, by rotating $T$ and by swapping $T(s - 1)$ and $T(s)$. This swapping is performed by processor $P - 1$, as described above. This process is repeated until all cities of $T$ are in their right position.

```
n := 0 ;
{I0}
while n < N do
  begin
    {I1: T(s) = T'(n) for n <= s < N}
    while s > n do
      begin
        while (T.part1(k1 - 1) of Processor P-1) <> T(s) do
          Rotate(T);
        Processor P-1: 2-change(T.part0, T.part1, k0 - 1, k1 - 1)
        {post: T(s-1) = T'(n)}
```

```
        s := s - 1;
        {I1}
      end;
      {I1 and s = n => T(n) = T'(n)}
      n := n + 1
      {I0}
  end
  {I0 and n = N => T = T'}
```

So, the above procedure guarantees that it is possible to construct a finite sequence of transitions and rotations leading from an arbitrary tour to another. As explained earlier, this condition satisfies asymptotic convergence of the annealing algorithm.

# 5 Computational results

In this section, we give some results of an implementation of the algorithm we described in the previous sections. We implemented the algorithm in the occam-2 language, and executed in on an array of T800 transputers. We had a total of 50 transputers at our disposal.

In the tables below, we give some computational results for TSP instances of 48, 120, and 532 cities respectively. They are called GRO48, GRO120, and GRO532 after their inventor, who proved that the optimal solutions to these problems are 5046, 6942, and 27,686 respectively [Grö77].

In the tables, the final solution cost average ($\overline{C}_{final}$) is computed from five runs of the algorithm. The average deviation from the global minimum is given by $\Delta\%$. The average execution time, which includes communication, is given in seconds by $\Delta t$.

| P | $\overline{C}_{final}$ | $\Delta\%$ | $\Delta t$ |
|---|---|---|---|
| 1 [Laa88] | 5094.8 | 0.97 | 93.8 |
| 1 | 5098.8 | 1.05 | 39.60 |
| 2 | 5102.2 | 1.11 | 19.02 |
| 3 | 5086.4 | 0.80 | 12.86 |
| 4 | 5086.2 | 0.80 | 10.02 |
| 6 | 5150.6 | 2.07 | 7.20 |
| 8 | 5132.2 | 1.71 | 5.84 |
| 12 | 5266.0 | 4.36 | 4.92 |

Table 1: Computational results for GRO48

In Tables 1 and 2 we have added some results from [Laa88] of a sequential annealing implementation. These results were obtained by running the annealing algorithm on a

VAX 11/785 computer.

| P | $\overline{C}_{final}$ | $\Delta\%$ | $\Delta t$ |
|---|---|---|---|
| 1 [Laa88] | 7057.2 | 1.66 | 1369.4 |
| 1 | 7014.6 | 1.05 | 958.4 |
| 5 | 7052.2 | 1.59 | 131.4 |
| 10 | 7070.4 | 1.85 | 65.8 |
| 15 | 7064.4 | 1.76 | 46.3 |
| 20 | 7101.2 | 2.30 | 36.7 |
| 30 | 7285.6 | 4.95 | 30.5 |

Table 2: Computational results for GRO120

The results in Table 1 and 2 are given for values for $P$ such that $P|N$. For the results in Table 3 this not the case. With a small modification, we could run our program for other values for $P$, as can be seen from Table 3. This enabled us to use the full strength of our transputer network, which means that we could take $P = 50$.

| P | $\overline{C}_{final}$ | $\Delta\%$ | $\Delta t$ |
|---|---|---|---|
| 10 | 28346.3 | 2.39 | 5348.8 |
| 20 | 28112.0 | 1.54 | 2392.1 |
| 30 | 28237.0 | 1.99 | 1549.8 |
| 40 | 28342.2 | 2.37 | 1154.5 |
| 50 | 28300.1 | 2.22 | 930.1 |

Table 3: Computational results for GRO532

For GRO532 no (sequential) annealing results are known to us. With his genetic algorithm, Mühlenbein reports a solution of length 27,702 in under three hours on a network of 64 transputers [Müh88]. This is less than 0.06% above optimal, and better than the best tour that Johnson finds in 20,000 independent runs of Lin-Kernighan (27,705), which took roughly 530 hours of Sequent processor time [Joh90]. We see that our algorithm cannot compete with these tailored heuristic algorithms with respect to the quality solution. This is, however, a well-known property of simulated annealing (cf. [Laa87, Laa88]).

# 6 Discussion

From the tables of the previous section we conclude that we obtained a speed-up that is more than linear. This can be explained as follows. Remember that each processor

has $2k$ cities at its disposal. This means, that the *acceptance* of a 2-change transition by a processor takes $\mathcal{O}(k)$ time. So, while the communication overhead *increases* with increasing $P$, the time to accept transitions actually *decreases*. In sequential implementations, tours are usually implemented as arrays. Hence, performing a 2-change (reversing a subpath) takes time proportional to the length of the tour. Our approach confirms the conclusion of Johnson that alternative representations might be cost-effective [Joh90].

Aonther conclusion is that communication overhead is too small to prohibit scalability of the algorithm: even for $\frac{N}{4}$ processors (the maximum) we have a linear speed-up.

With respect to the quality of final solutions, we conclude that the performance of the algorithm decreases if we let the number processors approach the maximum. This is due to the fact that for the maximum number of processors ($P = 30$ in our example), each processor has only four cities at its disposal, which means that each processor can only perform a single 2-change. Experiments on a larger transputer network will have to verify this conclusion for the 532-city TSP instance.

# 7 Conclusions

We presented in this paper a parallel simulated annealing algorithm for the TSP. The algorithm was shown to satisfy the asymptotic convergence conditions of simulated annealing. The computational results on two large TSP instances show that near-optimal solutions can be obtained with a substantial reduction in computation time. Our algorithm is scalable to a maximum of $\frac{N}{4}$ processors, for an $N$-city TSP instance. From the computational results we conclude that for $P \leq \frac{N}{6}$, the quality of solutions is comparable with the quality that is obtained by sequential annealing algorithms.

There are several topics for further research. First, our empirical study is rather limited: we investigated just three TSP instances. Tests on additional instances will have to verify our conclusions. A second interesting topic is the parallelization of other local search algorithms for the TSP (e.g., 3-opt and Lin-Kernighan), and their corresponding neighbourhood structures. The parallelization of the 2-opt neighbourhood structure that was presented in this paper can be used as a starting-point. An additional difficulty with the traditional local search algorithms is that we want to prove that they terminate with a locally optimal solution. In the parallel case this is far from trivial, and will require a more thorough investigation of the effect of parallelization on neighbourhood structures. Finally, we can investigate whether there are neighbourhood structures for other combinatorial optimization problems that are so 'easy' to parallelize as the 2-opt for the TSP. If so, simulated annealing can be used as local search algorithm, since one of the advantages of annealing is that it is general applicable.

## Acknowledgements

I would like to thank Emile Aarts and Martin Rem for their support and valuable comments on earlier versions of this paper.

## References

[Aar89] Emile Aarts and Jan Korst, *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons, 1989.

[Grö77] M. Grötschel, *Polyedrische Charakterisierungen Kombinatorischer Optimierungsprobleme* (in German), Hain, Meisenheim am Glan (1977).

[Joh90] David S. Johnson, *Local Optimization and the Traveling Salesman Problem*, Proc. $17^{th}$ Colloquium on Automata, Languages and Programming (ICALP '90), Springer Verlag, LNCS 447, pp. 446-461.

[Kir83] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi, *Optimization by Simulated Annealing*, Science 220(1983), pp. 671-680.

[Laa87] P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Company, Dordrecht, 1987.

[Laa88] P.J.M. van Laarhoven, *Theoretical and Computational Aspects of Simulated Annealing*, Ph.D. thesis, Erasmus University, Rotterdam, 1988.

[Lin65] S. Lin, *Computer solutions of the traveling salesman problem*, Bell Syst. Tech. J. 44(1965), pp. 2245-2269.

[Lin73] S. Lin and B.W. Kernighan, *An Effective Heuristic Algorithm for the Traveling Salesman Problem*, Oper. Res. 21(1973), pp. 498-516.

[Müh88] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, *Evolution algorithms in combinatorial optimization*, Parallel Computing 7(1988), pp. 65-85.

*In this series appeared:*

| | | |
|---|---|---|
| 89/1 | E.Zs.Lepoeter-Molnar | Reconstruction of a 3-D surface from its normal vectors. |
| 89/2 | R.H. Mak<br>P.Struik | A systolic design for dynamic programming. |
| 89/3 | H.M.M. Ten Eikelder<br>C. Hemerik | Some category theoretical properties related to a model for a polymorphic lambda-calculus. |
| 89/4 | J.Zwiers<br>W.P. de Roever | Compositionality and modularity in process specification and design: A trace-state based approach. |
| 89/5 | Wei Chen<br>T.Verhoeff<br>J.T.Udding | Networks of Communicating Processes and their (De-)Composition. |
| 89/6 | T.Verhoeff | Characterizations of Delay-Insensitive Communication Protocols. |
| 89/7 | P.Struik | A systematic design of a parallel program for Dirichlet convolution. |
| 89/8 | E.H.L.Aarts<br>A.E.Eiben<br>K.M. van Hee | A general theory of genetic algorithms. |
| 89/9 | K.M. van Hee<br>P.M.P. Rambags | Discrete event systems: Dynamic versus static topology. |
| 89/10 | S.Ramesh | A new efficient implementation of CSP with output guards. |
| 89/11 | S.Ramesh | Algebraic specification and implementation of infinite processes. |
| 89/12 | A.T.M.Aerts<br>K.M. van Hee | A concise formal framework for data modeling. |
| 89/13 | A.T.M.Aerts<br>K.M. van Hee<br>M.W.H. Hesen | A program generator for simulated annealing problems. |
| 89/14 | H.C.Haesen | ELDA, data manipulatie taal. |
| 89/15 | J.S.C.P. van der Woude | Optimal segmentations. |
| 89/16 | A.T.M.Aerts<br>K.M. van Hee | Towards a framework for comparing data models. |
| 89/17 | M.J. van Diepen<br>K.M. van Hee | A formal semantics for Z and the link between Z and the relational algebra. |

| | | |
|---|---|---|
| 90/1 | W.P.de Roever-<br>H.Barringer-<br>C.Courcoubetis-D.Gabbay<br>R.Gerth-B.Jonsson-A.Pnueli<br>M.Reed-J.Sifakis-J.Vytopil<br>P.Wolper | Formal methods and tools for the development of distributed and real time systems, p. 17. |
| 90/2 | K.M. van Hee<br>P.M.P. Rambags | Dynamic process creation in high-level Petri nets, pp. 19. |
| 90/3 | R. Gerth | Foundations of Compositional Program Refinement - safety properties - , p. 38. |
| 90/4 | A. Peeters | Decomposition of delay-insensitive circuits, p. 25. |
| 90/5 | J.A. Brzozowski<br>J.C. Ebergen | On the delay-sensitivity of gate networks, p. 23. |
| 90/6 | A.J.J.M. Marcelis | Typed inference systems : a reference document, p. 17. |
| 90/7 | A.J.J.M. Marcelis | A logic for one-pass, one-attributed grammars, p. 14. |
| 90/8 | M.B. Josephs | Receptive Process Theory, p. 16. |
| 90/9 | A.T.M. Aerts<br>P.M.E. De Bra<br>K.M. van Hee | Combining the functional and the relational model, p. 15. |
| 90/10 | M.J. van Diepen<br>K.M. van Hee | A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17). |
| 90/11 | P. America<br>F.S. de Boer | A proof system for process creation, p. 84. |
| 90/12 | P.America<br>F.S. de Boer | A proof theory for a sequential version of POOL, p. 110. |
| 90/13 | K.R. Apt<br>F.S. de Boer<br>E.R. Olderog | Proving termination of Parallel Programs, p. 7. |
| 90/14 | F.S. de Boer | A proof system for the language POOL, p. 70. |
| 90/15 | F.S. de Boer | Compositionality in the temporal logic of concurrent systems, p. 17. |
| 90/16 | F.S. de Boer<br>C. Palamidessi | A fully abstract model for concurrent logic languages, p. p. 23. |
| 90/17 | F.S. de Boer<br>C. Palamidessi | On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29. |

| | | |
|---|---|---|
| 90/18 | J.Coenen<br>E.v.d.Sluis<br>E.v.d.Velden | Design and implementation aspects of remote procedure calls, p. 15. |
| 90/19 | M.M. de Brouwer<br>P.A.C. Verkoulen | Two Case Studies in ExSpect, p. 24. |
| 90/20 | M.Rem | The Nature of Delay-Insensitive Computing, p.18. |
| 90/21 | K.M. van Hee<br>P.A.C. Verkoulen | Data, Process and Behaviour Modelling in an integrated specification framework, p. 37. |
| 91/01 | D. Alstein | Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14. |
| 91/02 | R.P. Nederpelt<br>H.C.M. de Swart | Implication. A survey of the different logical analyses "if...,then...", p. 26. |
| 91/03 | J.P. Katoen<br>L.A.M. Schoenmakers | Parallel Programs for the Recognition of *P*-invariant Segments, p. 16. |
| 91/04 | E. v.d. Sluis<br>A.F. v.d. Stappen | Performance Analysis of VLSI Programs, p. 31. |
| 91/05 | D. de Reus | An Implementation Model for GOOD, p. 18. |
| 91/06 | K.M. van Hee | SPECIFICATIEMETHODEN, een overzicht, p. 20. |
| 91/07 | E.Poll | CPO-models for second order lambda calculus with recursive types and subtyping, p. |
| 91/08 | H. Schepers | Terminology and Paradigms for Fault Tolerance, p. 25. |
| 91/09 | W.M.P.v.d.Aalst | Interval Timed Petri Nets and their analysis, p.53. |
| 91/10 | R.C.Backhouse<br>P.J. de Bruin<br>P. Hoogendijk<br>G. Malcolm<br>E. Voermans<br>J. v.d. Woude | POLYNOMIAL RELATORS, p. 52. |
| 91/11 | R.C. Backhouse<br>P.J. de Bruin<br>G.Malcolm<br>E.Voermans<br>J. van der Woude | Relational Catamorphism, p. 31. |
| 91/12 | E. van der Sluis | A parallel local search algorithm for the travelling salesman problem, p. 12. |
| 91/13 | F. Rietman | A note on Extensionality, p. 21. |
| 91/14 | P. Lemmens | The PDB Hypermedia Package. Why and how it was built, p. 63. |