A Parallel 2-opt Algorithm for the Traveling Salesman Problem

M.G.A. Verhoeven [1] E.H.L. Aarts[a, 2] P.C.J. Swinkels[a]

[1]Eindhoven University of Technology, Department of Mathematics and Computing Science, P.O. Box 513, 5600 MB Eindhoven, Netherlands

[2]Philips Research Laboratories, P.O. Box 80000, 5600 JA Eindhoven, Netherlands

1. Introduction

In a combinatorial optimization problem one must find an optimal solution among a finite, possibly large, number of alternatives [Papadimitriou & Steiglitz, 1982]. An important achievement is the conjecture that, unless $\mathcal{P} = \mathcal{NP}$, there exists a class of combinatorial optimization problems for which no algorithms exist that solve each instance of the problem to optimality in a time that is polynomially bounded in the size of the instance. These problems are denoted as *NP-hard* [Garey & Johnson, 1979]. As a consequence of this, the larger instances of such a problem can only be solved to proximity in reasonable time.

The Traveling Salesman Problem (TSP) is probably the best-known NP-hard combinatorial optimization problem. In the TSP, a salesman has to visit a number of cities once, and such that the length of the resulting tour is minimal. Much of the theory for combinatorial optimization has been developed using the TSP as a proving ground [Lawler, Lenstra, Rinnooy Kan & Shmoys, 1985].

We distinguish between two options for constructing algorithms to handle the TSP. Either one searches for optimal solutions, at the risk of very large, possibly intractable, computation time, or one is satisfied with relatively quickly obtainable solutions at the risk of sub-optimality. The first option constitutes the class of optimization algorithms; examples are implicit enumeration methods using branch and bound techniques. To indicate the impact of the use of optimization algorithms, we mention the 3038 cities instance solved to optimality by Applegate [Sangalli, 1992], using approximately one-and-a-half years of computation time. The second option constitutes the class of approximation algorithms; examples are local search algorithms, or more elaborated versions of it, such as simulated annealing [Kirkpatrick, Gelatt & Vecchi, 1983], tabu search [Glover, 1989], and variable depth search algorithms [Lin & Kernighan, 1973]. A classical example of a local search algorithm for the TSP is the 2-opt algorithm in [Lin, 1965].

However, large real-world instances, such as those originating from circuit lay-outing [Reinelt, 1992] and crystallography [Bland & Shallcross, 1989], require substantial amounts of computation time. One way to reduce computation time is by using parallelism. Our goal is to design a parallel algorithm that finds the same quality solutions as the best sequential algorithm, and that is scalable for massively parallel machines. Other approaches from the literature [Felten, Karlin & Otto, 1985; Allwright & Carpenter, 1989; Ravikumar, 1992; Fiechter, 1994; Bachem, Steckemetz & Wottawa, 1994] to design a parallel local search algorithm for the TSP, either are not scalable for a massively parallel machine, or do not obtain the same quality results as a sequential algorithm, or both.

The outline of this paper is as follows. In Section 2, we discuss the issue of local search, and its application for the TSP in particular. In Section 3, we discuss parallel local search algorithms, and introduce some notions. In Section 4 we present some numerical results, and in Section 5 we conclude with a discussion of topics for future research.

2. Local search

Local search is a general applicable approximation technique for hard combinatorial optimization problems. Local search gives good quality results for a wide variety of problems [Reeves, 1993]. The use of local search presupposes the specification of an instance of a combinatorial optimization problem and a neighborhood structure.

Definition 2.1. An instance of a combinatorial optimization problem is denoted as a pair $(\mathcal{S}, f)$, where the *solution space* $\mathcal{S}$ is the set of feasible solutions and $f : \mathcal{S} \to \mathbb{R}$ is a function that gives the cost of a feasible solution. The problem is to find a solution with minimal cost, in the case of minimization. A *neighborhood structure* $\mathcal{N} : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ assigns to each feasible solution a set of solutions that can be directly obtained from it. $\Box$

Definition 2.2. A solution $t \in \mathcal{S}$ is a *local minimum* w.r.t. $\mathcal{N}$, if $\forall_{t' \in \mathcal{N}(t)}\ f(t') \geq f(t)$. $\Box$

Local search algorithms constitute a class of approximation algorithms that are based on continually improving a solution by searching its neighborhood for a solution with lower cost. The simplest form of local search is iterative improvement which can be formulated as follows.

proc *Local_Search*$(\mathcal{N} : \mathcal{S} \to \mathcal{P}(\mathcal{S}); s, t \in \mathcal{S})$
$[\![t, W := s, \emptyset$;
while $\mathcal{N}(t) \setminus W \neq \emptyset$ **do**
$t' :\in \mathcal{N}(t) \setminus$ W ;
if $f(t') \geq f(t)$ **then** $W := W \cup \{t'\}$
else $t, W := t', \emptyset$
od$\{$ t is a local minimum w.r.t. $\mathcal{N}$ $\}$
$]\!]$

There are many variations on this basic local search algorithm known in the literature, such as simulated annealing, tabu search, genetic local search, and some types of recurrent neural networks. For an overview we refer to [Aarts, Korst & Zwietering, 1993]. Here we restrict our discussion to variants that follow the basic iterative improvement algorithm presented above.

2.1. Local search for the TSP

The TSP can be reformulated as the problem of finding a Hamiltonian cycle of minimal length in a complete weighted graph.

Definition 2.3 (TSP). Let V be a set of N cities and $d(i, j) \in \mathbb{R}$ a distance for each pair of cities i, j in V. Let $\pi : \{0, \cdots, N-1\} \rightarrow V$ be a bijection. A *tour* $t \subseteq V \times V$ is a set of N edges that constitutes a Hamiltonian cycle in the complete graph $(V, V \times V)$ and is defined as $t = \{e_s \in V \times V \mid 0 \leq s < N\}$, with an edge $e_s = (\pi_s, \pi_{(s+1) \bmod N})$. The solution space $\mathcal{S}$ of a TSP instance is the set of all tours, and the cost function $f : \mathcal{S} \rightarrow \mathbb{R}$ is given by $f(t) = \sum_{(i,j)\in t} d(i, j)$. Find a tour $t \in \mathcal{S}$ for which $f(t)$ is minimal. □

$\pi(i)$ is the city visited at the i^{th} position in the tour. We concentrate on the symmetric TSP, i.e., the distances satisfy $d(i, j) = d(j, i)$ for each $i, j \in V$. The best-known neighborhood structures for the TSP are based on the exchange of a number of edges. We mention the following ones.

- *2-opt* ($\mathcal{N}_2$). A tour t' is a neighbor of tour t if it can be obtained from t by removing two edges and inserting two edges. This transition is called a 2-exchange [Lin, 1965].
- *3-opt* ($\mathcal{N}_3$). A tour t' is a neighbor of tour t if it can be obtained from t by removing three edges and inserting three edges. This transition is called a 3-exchange [Lin, 1965].
- *Lin-Kernighan* ($\mathcal{N}_{LK}$). This is a variable depth neighborhood structure in which the number of edges that is exchanged to obtain a neighbor is variable, depending on the current tour t. Basically a series of 2-exchanges is constructed by a combination of first improvement, best improvement, and limited backtracking [Lin & Kernighan, 1973].

The empirical average-case time complexity of local search algorithms using the above neighborhood structures is low order polynomial. Furthermore, empirical results show that good quality solutions can be found with these neighborhood structures. The best approximation algorithm for the TSP is the iterated Lin-Kernighan algorithm [Johnson, 1990], which uses multiple runs of the Lin-Kernighan algorithm with a random 4-exchange to obtain a new starting solution.

3. Parallel local search

In the study of parallel local search algorithms we distinguish between algorithms with function and data parallelism. In an algorithm with data parallelism, data is distributed over a number of processes, and each process executes a local search algorithm. In an algorithm with function parallelism one step of the sequential local search algorithm is executed in parallel. Also, we distinguish between general and tailored parallelism. A general technique can be applied to a wide variety of problems, whereas a tailored approach strongly depends on the problem at hand.

3.1. Distributed neighborhoods

In this paper, we concentrate on tailored data parallelism. The idea of tailored data parallelism is to partition a solution into a number of disjoint partial solutions. Subsequently, the neighborhoods of all partial solutions are searched simultaneously. Since transitions involving elements of different partial solutions are not examined in this way, the partitions are changed in the course of the search.

Definition 3.1. Assume that $t \in \mathcal{S}$ is a set. Define a *partial solution* τ as $\tau \subseteq t$, and the set of all partial solutions $\mathcal{U} = \bigcup_{t\in\mathcal{S}} \mathcal{P}(t)$. Let $g : \mathcal{U} \rightarrow \mathbb{R}$ be such that

$\forall_{\tau,\tau'\in\mathcal{U}|t\backslash\tau=t'\backslash\tau'}\ f(t) - f(t') = g(\tau) - g(\tau')$

for all $t, t' \in \mathcal{S}$. $\Box$

If we can define such a function g for the problem at hand, then the cost difference between partial solutions can be computed locally. Subsequently, we introduce the following definitions to design a local search algorithm with tailored data parallelism.

Definition 3.2. Ω is a set of P processes with $P \geq 1$. Define a *distribution* $\delta : \Omega \to \mathcal{U}$, a *local neighborhood structure* $\mathcal{K} : \mathcal{U} \to \mathcal{P}(\mathcal{U})$ with $\tau \in \mathcal{K}(\tau)$ for all $\tau \in \mathcal{U}$, and a *distribution structure* Δ, with $\Delta(t, \Omega) \subseteq \{\delta : \Omega \to \mathcal{U} \mid \{\delta(p) \mid p \in \Omega\}$ is a partition of $t\}$ for $t \in \mathcal{S}$. Then, a *distributed neighborhood structure* $\mathcal{D}$ is a pair $(\Delta, \mathcal{L})$, where Δ is a distribution structure and $\mathcal{L}_p : \mathcal{U} \to \mathcal{P}(\mathcal{U})$ a local neighborhood structure for all $p \in \Omega$. $\Box$

According to Definition 3.2, a distribution is a mapping that assigns a partial solution to each process. A local neighborhood structure assigns to each partial solution a set of partial solutions —local neighbors— that can be reached in a single step of the search process. A distribution structure assigns to each $t \in \mathcal{S}$ a set of distributions where each distribution gives a partition of t in P partial solutions. A distributed neighborhood structure consists of a distribution structure and a mapping that assigns a local neighborhood structure to each process. Next, we formalize the notion of local optimality with respect to distributed neighborhood structures.

Definition 3.3. Let $\mathcal{K}$ be a local neighborhood structure and τ be a partial solution, then τ is a *partial local optimum* w.r.t. $\mathcal{K}$, if and only if $\forall_{\tau' \in \mathcal{K}(\tau)}\ f(\tau') \geq f(\tau)$. $\Box$

Definition 3.4. Let $\mathcal{D} = (\Delta, \mathcal{L})$ be a distributed neighborhood structure and $t \in \mathcal{S}$, then t is a *global local optimum* w.r.t. $\mathcal{D}$, if and only if $\forall_{\delta \in \Delta(t,P)} \forall_{p \in \Omega}\ \delta_p$ is a partial local optimum w.r.t. $\mathcal{L}_p$. $\Box$

Using these definitions we can formulate the following template for parallel local search algorithms with tailored data parallelism.

proc *Parallel_Local_Search*$(t \in \mathcal{S})$
$[\![D := \emptyset$;
 while $\Delta(t, P) \setminus D \neq \emptyset$ **do**
  $\delta :\in \Delta(t, P) \setminus D$;
  par$_{p \in \Omega}$ *Local_Search*$(\mathcal{L}_p, \delta_p, \delta'_p)$ **rap**;
  if $\forall_{p \in \Omega}\ f(\delta'_p) = f(\delta_p)$ **then** $D := D \cup \{\delta\}$
  else $D, t := \emptyset, \bigcup_{p \in \Omega} \delta'_p$
 od$\{$ t is a global local optimum w.r.t. $\mathcal{D}$ $\}$
$]\!]$

The algorithm terminates, because in each step the cost of a partial solution decreases and the number of partial solutions is finite. We have to prove that a global local optimum is feasible, i.e., that it belongs to the solution space. Therefore, we require that each solution found by the algorithm, is feasible. This condition is sufficient but not necessary, since we only need to prove the feasibility of the final solution. The feasibility of each solution found by the algorithm is guaranteed by the *safety* of a distributed neighborhood structure. The safety of a distributed

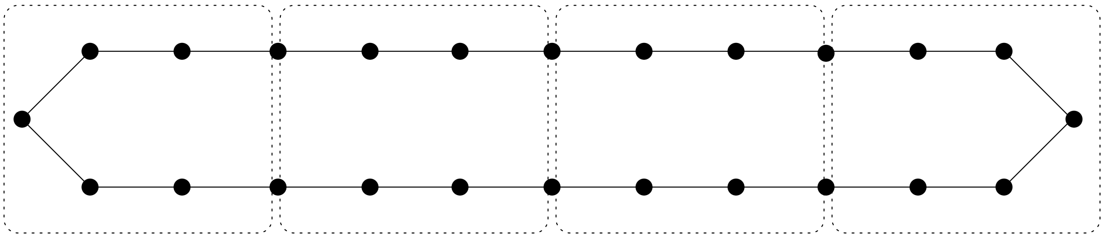

Figure 1. Distribution of a tour

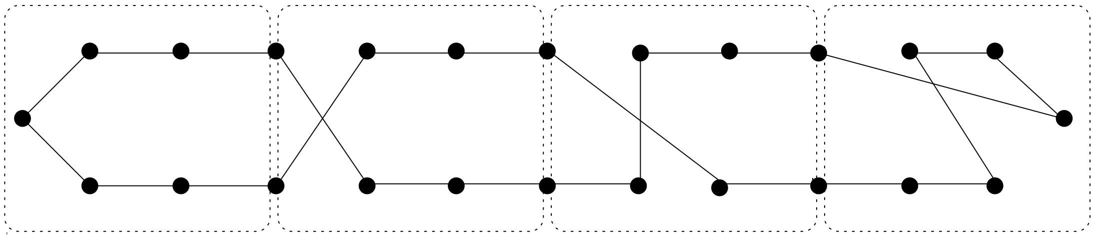

Figure 2. Local neighbors

neighborhood structure expresses that the composition of local neighbors of partial solutions is a feasible solution.

Definition 3.5. A distributed neighborhood structure $(\Delta, \mathcal{L})$ is *safe*, if and only if
$\forall_{\delta \in \Delta(t,P)} \forall_{\delta' : \Omega \to \mathcal{U} | \forall_{p \in \Omega}\, \delta'_p \in \mathcal{L}_p(\delta_p)} \, (\bigcup_{q \in \Omega} \delta'_q \in \mathcal{S})$.

□

An important issue is raised by the question how to compare a distributed neighborhood structure $\mathcal{D}$ with a neighborhood structure $\mathcal{N}$ in order to compare the solutions found by a parallel local search algorithm with those found by a sequential algorithm.

Definition 3.6. $\mathcal{D}$ is called *isomorphic* with $\mathcal{N}$, if $\forall_{t \in \mathcal{S}}\, t$ is a global local optimum w.r.t. $\mathcal{D} \Leftrightarrow t$ is a local optimum w.r.t. $\mathcal{N}$. □

If $\mathcal{D}$ is isomorphic with $\mathcal{N}$ and we assume that a locally optimal solution is found with the same probability by a sequential and a parallel algorithm, then the expected cost of final solutions are equal for $\mathcal{D}$ and $\mathcal{N}$.

3.2. A distributed neighborhood structure for the TSP

The concepts we have introduced in Section 3.1, have been applied to the TSP. Here, we present some of the main results. Other results can be found in [Verhoeven, Aarts, Van de Sluis & Vaessens, 1992]; technical details can be found in a forthcoming paper [Verhoeven, Swinkels & Aarts, 1995].

Tailored data parallelism requires a distribution of a solution over a number of processes. A partial solution in our distribution of a tour consists of two paths. The number of edges in each path of a partial solution is not fixed, only the total number edges of a partial solution is fixed. The sizes of partial solutions should be roughly equal to obtain an algorithm with a good load-balance. Figure 1 gives an example of a distribution with four partial solutions. The network topology that is derived from this distributed neighborhood structure is a linear topology. To obtain a new distribution, each process sends one edge to its left adjacent process; to obtain the next distribution each process sends one edge its right adjacent process, such that the tour is rotated. The distribution structure consists of all distributions that can be obtained in this way. The local neighbors of a partial solution are the partial solutions that can be obtained by applying a 2-exchange that involves the most recently received edge. Figure 2 gives some examples of local neighbors obtained by applying a 2-exchange on some of the partial solutions. In our distribution it is not necessary that both paths of a partial solution contain an equal number of edges (cf. Figure 2), which makes it possible for a process to pass an edge to an adjacent process, even if it has not received an edge from the other adjacent process. This is an important observation, because otherwise the communication complexity would be $O(P)$.

In [Verhoeven, Swinkels & Aarts, 1995] we define a distributed neighborhood structure $\mathcal{D}_2$ that uses this distribution structure. Also, we show that $\mathcal{D}_2$ is isomorphic with $\mathcal{N}_2$, which ensures the same quality of solutions as obtained with the sequential 2-opt algorithm. The algorithm is scalable for at most $N/2$ processes, because each partial solution must contain at least two edges.

3.3. Complexity

To conclude Section 3, we discuss the complexity of a parallel local search algorithm that uses the $\mathcal{D}_2$ neighborhood.

An unsolved issue in local search is the complexity to find a local optimum. To this end, a new complexity class $\mathcal{PLS}$ has been introduced in [Johnson, Papadimitriou & Yannakakis, 1988], which contains the problems for which local optimality can be verified in polynomial time. Next, a PLS-completeness notion has been introduced, and an important unsolved issue is whether PLS-complete problems are also NP-hard. Hence, it is not possible to give a non-trivial upper bound on the number of iterations needed to find a local optimum, and therefore we restrict ourselves to investigating the time complexity to verify global local optimality of a solution.

The time complexity to verify partial local optimality is $O(\frac{N}{P})$. When a process has examined its local neighborhood, it is able to concurrently send and receive an edge, if each path of its partial solution contains at least two edges. Therefore the complexity of the communication is

$$1 + \max_{p,q \in \Omega | \forall_{p \leq i < q}\, l_i = 1} q - p,$$

where l_i is minimum number of edges of a path belonging to process i. Hence, the worst case complexity of communication is $O(P)$. The empirical average case complexity is $O(1)$, because during the course of the search mostly holds $l_i > 1$, which gives a time complexity for communication of $O(1)$. Since the number of distributions of a tour that has to be examined is N, the worst case time complexity to verify global local optimality w.r.t. $\mathcal{D}_2$ is $O(N * (\frac{N}{P} + P))$. Since the average case communication complexity is $O(1)$, the average case complexity of verifying global local optimality is $O(\frac{N^2}{P})$, and therefore we expect an almost linear speed-up and scalability. Furthermore, we see that the complexity of our algorithm for $P=1$ is equal to the complexity of verifying local optimality for the sequential 2-opt algorithm, which shows that our parallel algorithm examines the same number of 2-exchanges as the sequential 2-opt algorithm.

4. Numerical results

The algorithms sketched in Section 3, have been implemented in C on a Parsytec GCel, consisting of 512 INMOS T805 transputers, that has been installed at SARA.

A tour is represented in a linked list which makes it possible to construct a 2-exchange neighbor in $O(1)$. Initial tours are constructed using the nearest neighbor heuristic [Lawler, Lenstra, Rinnooy Kan & Shmoys, 1985]. We used instances with several thousands of cities, originating from the TSP library of [Reinelt, 1991]. We discuss speed-up, efficiency, and the influence of the number of processors on the average cost of the final solutions. The discussion is based on the empirical results obtained for the solved instances att532 (27686), rl1002 (259945), rl1304 (252948) pr2392 (378032), pcb3038 (137694), and the unsolved instances d2103 (79687), rl5934 (552082), and rl11849 (913981); the number in the name of an instance denotes the number of cities of this instance, and the number in parentheses gives the cost of the optimal solution, or the best known lower bound. The distances for att532 are given in a matrix, but the distances for the other instances are computed using coordinates, due to limited memory capacity.

In the tables below we present the empirical results. Here, P is the number of processors, and f is the average cost of the final solutions computed from 5 runs. The average excess over the minimal cost or its lower bound, measured in percentages, is given by $\Delta(\%)$. The average running time in seconds is given by $T(\mathrm{s})$. We define speed-up as suggested in [Barr & Hickman, 1993], i.e., the running time of an efficient sequential algorithm divided by the running time of an algorithm on P processors. We can, however, not guarantee that the algorithm gives the same output for each number of processors, as a consequence of the data dependent behavior of a local search algorithm. The efficiency, defined as speed-up divided by P, is denoted by eff. For large instances we were not able to run the algorithm for $P=1$, due to limited amounts of computation time, so in these cases the efficiency is computed relatively to the smallest number of applied processors.

In Tables 1, 2, 3, and 4, the results obtained with $\mathcal{D}_2$ are listed. From these tables we observe that the number of processes has no influence on the average cost of the final solution. This is explained by the theorem which states that $\mathcal{D}_2$ is isomorphic with $\mathcal{N}_2$ [Verhoeven, Swinkels & Aarts, 1995]. Also, we obtain a good efficiency, viz., more than 50 percent if the partial solutions are sufficiently large. Roughly, we can state that the partial solution size ($N \mathrm{div} P$) should at least be 10, i.e., each process should own at least 10 edges, to obtain a good efficiency.

5. Discussion

The objective of our research is to design parallel local search algorithms that can compete with the best existing algorithms for large TSP instances with respect to quality of final solutions, and that can outperform these algorithms with respect to computation time. For this purpose,

P	f	$\Delta(\%)$	T(s)	eff
1	29482	6.5	144.2	1
16	29861	7.8	13.0	0.69
64	29747	7.4	5.2	0.43
128	29801	7.6	5.5	0.20

P	f	$\Delta(\%)$	T(s)	eff
1	279532	7.9	1436	1
64	278296	7.4	29	0.77
128	278569	7.5	18	0.62
256	279250	7.8	19	0.29

Table 1

Computational results for att532 and pr1002.

P	f	$\Delta(\%)$	T(s)	eff
1	282390	11.6	1960	1
64	280339	10.8	52	0.59
128	280403	10.8	31	0.49
256	281951	11.5	32	0.24

P	f	$\Delta(\%)$	T(s)	eff
16	84026	5.4	481	1
64	83804	5.2	103	1.17
128	83332	4.6	69	0.87
256	83917	5.3	76	0.40

Table 2

Computational results for rl1304 and d2103.

P	f	$\Delta(\%)$	T(s)	eff
16	411334	8.8	685	1
64	412423	9.1	189	0.91
128	410403	8.6	109	0.79
256	413039	9.3	66	0.65

P	f	$\Delta(\%)$	T(s)	eff
64	149757	8.8	312	1
128	149573	8.6	168	0.93
256	149480	8.6	108	0.72
512	149647	8.7	79	0.49

Table 3

Computational results for pr2392 and pcb3038.

P	f	$\Delta(\%)$	T(s)	eff
128	601920	9.0	645	1
256	603542	9.3	315	1.02
512	599291	8.6	215	0.75

P	f	$\Delta(\%)$	T(s)	eff
128	994243	8.7	2671	1
256	994230	8.7	1396	0.96
512	998387	9.2	819	0.82

Table 4

Computational results for rl5934 and rl11849.

we have designed and implemented a parallel 2-opt algorithm that is efficiently scalable for a massively parallel machine, as is shown by our empirical results.

The most effective algorithm to handle the TSP is the iterated Lin-Kernighan algorithm [Johnson, 1990], and therefore our current research concentrates on designing and implementing a parallel local search algorithm that finds the same quality solutions and has a good speed-up. This algorithm requires a more complex communication network than the network topology for the parallel 2-opt algorithm, because a processor has to communicate not only with physically adjacent processors, but also with other processors. A machine equipped with the T9000 transputer provides a fast communication between processors that are not physically connected. This assures a good balance between communication and computation, which is essential to obtain a good speed-up. In addition, to handle even larger TSP instances, a larger computational power is required, which is supplied by the T9000. Also, we have observed from our experiments that, although theoretically asynchronous communication should result in a faster algorithm, in practice it results in a slower algorithm than synchronous communication. We believe that this phenomenon is due to the larger operating system overhead of asynchronous communication, and we expect that this will no longer be the case with a machine equipped with the T9000.

In future research we will concentrate on the design and implementation of parallel local search algorithms based on tailored data parallelism, where in addition to the TSP we also plan to study other combinatorial optimization problems. An important issue is the scalability of these algorithms, and the relation between quality and number of processors, i.e., the effect of increasing the number of processors on the quality of final solutions.

Furthermore, we plan to study other techniques to introduce parallelism in local search algorithms, and to study their performance and scalability. The main goal of our research is to investigate the potentials of massively parallel processing for local search. An empirical performance analysis of the results obtained by implementations of parallel local search algorithms on massively parallel machines is considered as an important aspect of this research.

References

AARTS, E.H.L., J.H.M. KORST, AND P.J. ZWIETERING [1993], Deterministic and randomized local search, in: P. Smolensky, M. Mozer, and D. Rumelhart (eds.), *Mathematical Perspectives of Neural Networks*, Erlbaum Associates.

ALLWRIGHT, J.R.A., AND D.B. CARPENTER [1989], A distributed implementation of simulated annealing for the traveling salesman problem, *Parallel Computing* **10**, 335–338.

BACHEM, A., B. STECKEMETZ, AND M. WOTTAWA [1994], *An efficient parallel cluster heuristic for large traveling salesman problems*, Report 94-150, Zentrum für Paralleles Rechnen, Universität Köln, Germany.

BARR, R.S., AND B.L. HICKMAN [1993], Reporting computational experiments with parallel algorithms, *ORSA Journal on Computing* **5**, 2–18.

BLAND, R.G., AND D.F. SHALLCROSS [1989], Large traveling salesman problems arising from experiments in x-ray crystallography, *Operations Research Letters* **8**, 123–133.

FELTEN, E., S. KARLIN, AND S.W. OTTO [1985], The traveling salesman problem on a hypercubic mimd machine, *Proc. Int. Conference on Parallel Processing*, 6–10.

FIECHTER, C.N. [1994], A parallel tabu search algorithm for large traveling salesman problems, *Discrete Applied Mathematics* **51**, 243–267.

GAREY, M.R., AND D.S. JOHNSON [1979], *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco.

GLOVER, F. [1989], Tabu search – part I, *ORSA Journal on Computing* **1**, 190–206.

JOHNSON, D.S. [1990], Local optimization and the traveling salesman problem, *Proc. 17th Colloquium on Automata, Languages, and Programming*, LNCS 443, Springer-Verlag, 446–461.

JOHNSON, D.S., C.H. PAPADIMITRIOU, AND M. YANNAKAKIS [1988], How easy is local search?, *Journal of Computer and System Sciences* **37**, 79–100.

KIRKPATRICK, S., C.D. GELATT, JR., AND M.P. VECCHI [1983], Optimization by simulated annealing, *Science* **220**, 671–680.

LAWLER, E.L., J.K. LENSTRA, A.H.G. RINNOOY KAN, AND D.B. SHMOYS [1985], *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

LIN, S. [1965], Computer solutions of the traveling salesman problem, *Bell System Technical Journal* **44**, 2245–2269.

LIN, S., AND B.W. KERNIGHAN [1973], An effective heuristic algorithm for the traveling salesman problem, *Operations Research* **21**, 498–516.

PAPADIMITRIOU, C.H., AND K. STEIGLITZ [1982], *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs.

RAVIKUMAR, C.P. [1992], Parallel techniques for solving large scale travelling salesperson problems, *Microprocessors and Microsystems* **16**, 149–158.

REEVES, C.R. (ed.) [1993], *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell, London.

REINELT, G. [1991], TSPLIB: A traveling salesman problem library, *ORSA Journal on Computing* **3**, 376–384.

REINELT, G. [1992], Fast heuristics for large geometric traveling salesman problems, *ORSA Journal on Computing* **4**, 206–217.

SANGALLI, A. [1992], Short-circuiting the traveling salesman problem, *New Scientist* **6**, 16–17.

VERHOEVEN, M.G.A., E.H.L. AARTS, E. VAN DE SLUIS, AND R.J.M. VAESSENS [1992], Parallel local search and the travelling salesman problem (extended abstract), *Parallel Problem Solving from Nature 2*, North-Holland, Amsterdam, 543–552.

VERHOEVEN, M.G.A., P.C.J. SWINKELS, AND E.H.L. AARTS [1995], *Parallel Local Search and the Traveling Salesman Problem*, working document.