

# A Distributed Chained Lin-Kernighan Algorithm for TSP Problems

Thomas Fischer  
 Department of Computer Science  
 University of Kaiserslautern  
 fischer@informatik.uni-kl.de

Peter Merz  
 Department of Computer Science  
 University of Kaiserslautern  
 pmerz@informatik.uni-kl.de

## Abstract

The Chained Lin-Kernighan algorithm (CLK) is one of the best heuristics to solve Traveling Salesman Problems (TSP). In this paper a distributed algorithm is proposed, where nodes in a network locally optimize TSP instances by using the CLK algorithm. We show that the distributed variant finds better tours compared to the original CLK given the same total amount of computation time. Hence, the cooperation of the processes in the distributed algorithm increases the effectiveness of the approach beyond the maximally achievable reduction in computation time due to parallelization. E.g. for TSP instance *f13795*, the original CLK got stuck in local optima in each of 10 runs, whereas the distributed algorithm found optimal tours in each run requiring less than 10 CPU minutes per node on average in an 8 node setup.

## 1 Introduction

The Traveling Salesman Problem (TSP) is one of the most well known *combinatorial optimization problems*. It can be described as a salesman's problem of finding the cost-optimal route to a given number of cities (customers) such that each of these cities is visited exactly once. The problem can be represented by a full connected graph  $G = (V, E)$  and a function  $d$  denoting the distance between two vertices  $v_i, v_j \in V$ . For *symmetric TSPs* (STSP)  $d_{i,j} = d_{j,i}$  holds always for all  $i$  and  $j$ . For *asymmetric TSPs* (ATSP)  $d_{i,j} \neq d_{j,i}$  holds for at least one pair  $(v_i, v_j)$ . The optimal solution is a *Hamiltonian cycle* where the sum of the distance values is minimal. Such a cycle is a *permutation*  $\pi$  on the vertices  $V$  that minimizes the cost function  $C(\pi)$ .

$$C(\pi) = \sum_{i=1}^{n-1} d_{\pi(i), \pi(i+1)} + d_{\pi(n), \pi(1)} \quad (1)$$

For an instance with  $n$  cities ( $n = |V|$ ), there are  $\frac{(n-1)!}{2}$  different tours. Although having a simple setup, the TSP is a *NP-hard* problem [9, 15].

## 1.1 Algorithms

*Heuristic algorithms* perform a non-complete search in the solution space and do not guarantee to find an optimal solution. Their main advantage is that they can find a good sub-optimal solution in much shorter time than exact algorithms. Heuristic algorithms performing a *local search* usually exploit neighborhood relations between nodes. One example is the *k-opt neighborhood* [14, 19], where two tours are called neighbors if one tour can be transformed to the other by exchanging  $k$  edges. Although a larger  $k$  gives better tours, for most applications  $k$  is limited to  $k \leq 3$  due to exponentially rising computation costs.

Lin and Kernighan [20] approached the problem of finding a tradeoff between tour quality and computation cost by introducing an algorithm (LK) where  $k$  is kept variable. In each iteration the algorithm performs a sequence of exchange moves (thus increasing  $k$ ) while considering whether or not a possible move could lead to a better tour. Although this depth search returns only better tours (if available) it might consider internally worse tours as intermediate steps.

To improve the results of the LK algorithms, early implementations restarted the algorithms with new initial tours iteratively. This approach was superseded by the Chained LK algorithm (CLK) introduced by Martin, Otto and Felten [21], which follows a simulated annealing (SA) pattern. Instead of restarting with a new tour, the chained variant perturbs a LK-optimized tour by applying a 4-exchange move (*double-bridge move*, DBM) to escape from local optima.

## 1.2 Motivation

Due to improved algorithms and faster computers the size of instances that could be solved has grown steadily since the beginning of TSP research. The first instance solved to optimality was a 42/49-cities problem [11] by Dantzig *et al.* in 1954. The latest achievement was the solution for *sw24978* in May 2004. The required amount of

computation power was supplied by a cluster of 96 dual processor machines (2.8 GHz) requiring over 80 CPU years to prove optimality with an exact algorithm. In contrast, for heuristic algorithms distributed computation is not common yet. This might be due to the fact that heuristic algorithms require less computation time. But to solve large TSP instances with today's heuristic algorithms it is inevitable from our point of view to distribute computation. Our approach is presented in this paper.

The classical approach for distributed computing is to setup a client-server system, where the central server organizes the workflow, while the clients are not aware of other clients and just perform their computations. The main drawback here is the server as the single point of failure. As every communication is routed through this server its throughput bounds the size of the network and thus the system is not scalable. This problem is addressed by Peer-to-Peer (P2P) networks. In these networks each node is both server and client for other nodes and ideally all nodes are symmetric. Unlike traditional client-server systems, P2P networks are designed for a dynamic environment where nodes can join and leave at any time, communication is asynchronous and no global information is known. For file sharing, this technology has been applied e.g. in FreeNet [8]. Distributed computing is a rather new branch in P2P systems. A prominent project in this area is DREAM [22], which is an distributed environment to simulate evolutionary computation experiments where distributed resource machine (DRM) nodes communicate using epidemic algorithms [12].

In this paper we present a distributed algorithm for solving Traveling Salesman Problems. This algorithm utilizes an existing Chained Lin-Kernighan algorithm from Applegate *et al.* [2] and embeds it into an evolutionary algorithm that is running distributed over several nodes in a network. With this approach our algorithm finds both better tours given a computation time limit and it converges faster towards an optimal solution compared to the original Chained Lin-Kernighan algorithm.

The paper is organized as follows. The rest of the introduction presents several related studies. Section 2 introduces the architecture of the distributed algorithm and important algorithmic features. Section 3 describes the testbed and available and used parameters. Section 4 presents the results of the simulation runs in detail. Finally, in section 5 conclusions are drawn and perspectives for future work are proposed.

### 1.3 Distributed Algorithms for large TSPs

Bachem and Wottawa proposed in 1992 two attempts to parallelize heuristics for the TSP on transputers [6]. They used a job-level parallelization running the LK algorithm on different processors, where each run starts with a ran-

dom generated tour broadcasting locally optimized tours to the other nodes. In a speed-up technique called *partial reduction*, edges that occurred previously on good tours were protected in subsequent LK iterations resulting in a runtime reduction of about 10 – 50% while keeping the tour quality constant. Another algorithm by Bachem and Wottawa is based on clustering the original instance. For the generation of clusters for Euclidean TSPs, the authors divide the node set recursively into two halves. Although having problems with suboptimal clustering, the resulting tours were only 2 – 5% worse than LK tours.

The Asparagus96 system by Gorges-Schleuter [16] is an asynchronous parallel genetic algorithm. In each generation each individual selects a mate from its neighborhood and performs a MPX2 (*maximal preservative crossover*) crossover. The resulting tour which may be damaged is mutated by a double-bridge move and repaired by a 2-repair step (3-repair step for ATSPs). If the offspring has a shorter tour length than its ancestor, the ancestor is superseded by the offspring. For parallelization the algorithm simulates several populations in parallel, where the best individual of a population can be chosen as mate for a individual in another population. The average tour quality for instance f13795 over 10 runs results in a tour quality of 0.34% above the optimum.

A multilevel approach embedding a Chained Lin-Kernighan algorithm has been proposed by Walshaw [24]. He uses a multilevel scheme where a problem instance is iteratively coarsened by matching and merging cities to reduce the problem size. After constructing a tour in the reduced instance, the instance is stepwise uncoarsened again. In each uncoarsening step the current tour is refined by the  $C^N$ LK algorithm. The resulting tours are better compared to the plain CLK algorithm given the same computation time. Furthermore, a given tour quality is reached quicker with the multilevel system. In a testbed of 81 instances the multilevel setup  $MLC^{N/10}$ LK (number of kicks for CLK is one tenth of the number of cities) archives on average slightly better tours and is still 4 times faster than CLK.

## 2 Description of the used Algorithms

### 2.1 Details on Chained Lin-Kernighan

In this subsection some details regarding the Chained Lin-Kernighan algorithm and its implementation by Applegate *et al.* are presented.

The Quick-Borůvka algorithm [5] constructs a TSP tour by sorting the vertices in geometric instances by their coordinates to determine the order of processing. The algorithm iterates at most twice until a valid tour has been constructed. In each iteration each city which has not yet two adjacent edges in the partial tour is processed. For a city

that is processed an adjacent edge with the following properties is selected, that has a minimum weight, does not lead to a subtour and is not adjacent to a city that has already two other adjacent edges. This selected edge is inserted into the partial tour. In [5] results from a comparison of CLK runs based on tour constructed by the HK-Christofides heuristics [7] or Quick-Borůvka, respectively, show that the latter heuristics results in better tours, although the first requires more time for tour construction.

As part of any Chained Lin-Kernighan algorithm a tour that was optimized by the Lin-Kernighan algorithm (LK) is perturbed. Martin, Otto and Felten proposed in [21] a double-bridge move (DBM) to “kick” the intermediate tour. The double-bridge move is a 4-exchange move which is a cheap move as it does not require to flip the order of the cities in any subtour. In the DBM, four edges are deleted from the tour and four new edges are added. Depending on the selection of the 4 relevant cities this move can have a strong impact on the tour. For the selection of the 4 cities Applegate *et al.* propose four kicking strategies. The Random kicking strategy selects the relevant cities at random. This strategy degenerates the tour, but might help to leave a local optimum. For the Geometric kicking strategy the relevant cities are chosen from the  $k$  nearest neighbors of a selected city  $v$ . Small  $k$  causes the kick to be local, whereas for large  $k$  the kick becomes similar to the Random kick. In the Close strategy, a subset of the cities of size  $\beta n$  ( $\beta$  is a parameter,  $n$  is the number of cities) is chosen. From this subset the six cities nearest to a chosen city  $v$  (first relevant city) are used to choose the three other relevant cities. Larger  $\beta$  make it more likely that cities close to  $v$  are part of the subset and thus used for a local kick. The Random-walk strategy starts from the first relevant city  $v$  three independent random walks of a given length on the original graph. The end points are the missing relevant cities.

## 2.2 System Architecture

The system is a structured network of computing nodes. During an initial setup phase the nodes connect to a dedicated bootstrapping node (referred to as “hub”) that constructs the structure by supplying neighborhood lists to each node. For the simulations here, eight nodes were arranged in a hypercube topology.

Each node in the distributed system consists of an evolutionary algorithm (EA) that uses a network module to communicate with other nodes and a Chained LK (CLK) module for solving problem instances locally. The software is written in Java, except for the CLK module, which was taken from the Concorde package [2, 4] (031219). This CLK implementation is well-known in the TSP community and has been used by other researchers (e.g. Walshaw in [24]), too. The CLK module was wrapped by a JNI inter-

face to use it from within the Java code.

As the whole communication is based on TCP/IP, the nodes are supplied with hostname and port number of the software hub during startup. Each node contacts the hub and requests the list of its neighbors. The hub determines the node’s position within the hypercube and builds the node’s neighbor list. As this list is based on nodes that are already known to the hub, the first nodes will receive a sparse list of neighbors. To build a connected hypercube, a node contacts each neighbor after receiving the list. If the contacted node did not know the contacting node before, the contacting node is added to the contacted node’s neighbor list. As the nodes communicate directly by TCP connections and hence constitute a peer-to-peer network, the hub is the only central component in the network and is only used during initialization of the network. For systems with a small network size, this approach appears to be feasible.

The network module includes functions for sending and receiving tours from other nodes. The CLK module offers functions for Quick-Borůvka based tour construction and local search optimization of already perturbed tours to the EA module. For most parameters of the CLK algorithm the default values were used, except for the kicking type strategy, which was supplied by the EA module.

## 2.3 Algorithm

Simplified, the algorithm is structured as follows. In each iteration the node’s tour is perturbed by one or several random double-bridge moves. This perturbed tour is optimized by the Chained Lin-Kernighan algorithm. Thereafter, the new tour is compared with all tours received meanwhile from other nodes. The best tour is stored as the node’s new tour. If this tour was the result of the local CLK function it is broadcasted to all neighboring nodes. The pseudo code for this algorithm is shown in Figure 1.

The termination criterion as represented by the function TERMINATIONDETECTED can be (1) the discovery of the known optimal tour length (if known) by the local CLK function, (2) a notification message that another cluster node has found an optimal solution or (3) a predefined limit of time or number of CLK calls that has been reached. Due to different running times on the nodes at the end of a simulation more and more nodes might become inactive. Thereby the network topology degenerates and the neighborhood of each nodes decreases. As there is no global control, the best result of this simulation has to be collected from the local output of each node independently from the simulation itself.

Nodes perturbate their current best tour before optimizing it by using the CLK algorithm to leave their current local optimum, but the strength of the perturbation has to be chosen carefully. A perturbation that is too weak might not help

```

function DISTRIBUTEDALGORITHM
   $s_{prev} := \text{INITIALTOUR}$ ;
   $s_{best} := \text{CHAINEDLINKERNIGHAN}(s_{prev})$ ;
  while not TERMINATIONDETECTED do
     $s := \text{CHAINEDLINKERNIGHAN}(\text{PERTURBATE}(s_{best}))$ ;
     $S_{received} := \text{ALLRECEIVEDTOURS}$ ;
     $s_{best} := \text{SELECTBESTTOUR}(S_{received} \cup \{s\} \cup \{s_{prev}\})$ ;
    if  $\text{LENGTH}(s_{best}) = \text{LENGTH}(s_{prev})$  then
       $\text{NumNoImprovements} ++$ ;
    else if  $s_{best} = s$  then
       $\text{BROADCASTTONEIGHBORS}(s_{best})$ ;
    end if
     $s_{prev} := s_{best}$ ;
  end while

function PERTURBATE( $s$ )
  if  $\text{NumNoImprovements} > c_r$  then
     $\text{RESETCOUNTERS}$ ;
    return  $\text{INITIALTOUR}$ ;
  else
     $\text{NumPerturbations} := \lfloor \frac{\text{NumNoImprovements}}{c_v} \rfloor + 1$ ;
    return  $\text{VARIATE TOUR}(s, \text{NumPerturbations})$ ;
  end if

```

**Figure 1. Pseudo code for the distributed EA and for the perturbation step.**

to leave the current local optimum, but a too strong perturbation might damage the tour too heavily causing a loss of quality. So, as a compromise, the used strategy begins with a weak perturbation and increases its strength if no better tours are found. If subsequent strength increase does not help, the current tour is discarded and a new initial tour will be constructed.

Whenever the CLK function does not find a better tour than the previous best tour, a counter ( $\text{NumNoImprovements}$ ) is increased. This counter gets resetted when a better tour has been found or received from another node. Its value is used to determine the number of random double-bridge moves applied to a tour (formula see Figure 1), controlled by a parameter  $c_v$ . A higher number of perturbation moves leads to stronger tour perturbation. Eventually, the perturbation moves will modify the tour, so that it will leave the current local optimum. If multiple perturbation moves do not help to change the tour significantly, the current tour will be discarded and a new tour will be constructed. This event occurs if the number of iterations without improvements reaches the value of a parameter  $c_r$ .

### 3 Experimental Setup

For our analysis a set of instances from various sources has been selected. The instance sizes range from 1000 to 85900 cities and are the same as used in other research projects. The numbers in the names denotes the number of cities in the instances.

- From Reinelt’s TSPLIB [23] the following instances were taken: f11577, f13795, pr2392, pcb3038, fn14461, usa13509, pla33810 and pla85900.
- From the 8th DIMACS challenge [1] two random instances were used, where 1000 cities are arranged in a square using Euclidean distances. For instance E1k.1, the cities are randomly uniform distributed, while in instance C1k.1 cities are normally distributed around one of 10 cluster centers.
- From the collection of national TSPs [3]: fi10639 and sw24978. An optimal solution for sw24978 has been recently found (March 2003, approved in May 2004).

The cluster used for this analysis consisted of eight identical computer nodes with one 3.0 GHz SMT processor (Pentium 4) and 512 MB RAM each running Linux (Kernel 2.6). The nodes were connected in a switched Ethernet with 1 Gbps.

#### 3.1 Runs & Parameters

Each simulation setup was performed 10 times. The number of runs was limited due to time constraints. For further analysis average values were calculated.

The program linkern that is part of the concorde package has been used as CLK engine. For the first part of the analysis, no modifications were made on the source code. The resulting values were used for comparison with later results from the distributed algorithm.

Each instance of the testbed was solved with each of the four kicking strategies. The number of kicks (termination criterion of the algorithm) was set to a very high value to make time bounds the only termination criterion. The time limit was set to  $10^4$  CPU seconds for instances with less than  $10^4$  cities and  $10^5$  CPU seconds otherwise. For instances with known optimum, this optimum was set as an additional termination criterion.

For the distributed algorithm, the number of CLK calls has been set to unlimited to make time bounds the only termination criterion here, too. The time limit was set to  $10^3$  CPU seconds per node for instances with less than  $10^4$  cities and  $10^4$  CPU seconds per node otherwise, which is a tenth of the values for the original CLK. For instances with known optimum, this optimum was set as an additional termination criterion. The parameters were set as  $c_v = 64$  and  $c_r = 256$ .

Some parameters have been changed in different simulations to observe effects of different values. Primary simulations were performed using a hypercube topology with 8 nodes (therefore the time bound reduction by factor 10). Additionally, setups with only 1 node were performed to check the influence of parallelization. The kicking type

was set to one of the four valid types (Random, Geometric, Close and Random-walk). As Random-walk performed best during initial simulations, simulations for larger instances were primarily done with this kicking strategy.

## 4 Experimental Results

The number of messages broadcasted between nodes corresponds to the number of improvements found by a node. In case of all 30 runs of instance `sw24978` using an eight node setup, 2546 times a node found a better tour and sent it to the other nodes. On average, 84.9 broadcasts were initiated per run, so each node sent about 11 messages in a  $10^4$  seconds period. Due to rapid improvements at the beginning of each run, most messages are broadcasted within this phase. On average, the first 10 messages of a run were sent by nodes that had consumed less than 692.3 CPU seconds at the time of sending. Although at the start of each run better tours are found more often, the overall communication overhead is neglectable.

### 4.1 Chained Lin-Kernighan

As noted above, the Chained Lin-Kernighan program `linkern` from the Concorde package (ABCC-CLK, for short CLK) was used to solve all TSP instances from the testbed. On each instance all four kicking strategies were applied.

For instances with a size above 3000 cities, CLK could not find an optimal solution in any run. For smaller instances, the Random kicking strategy had the most successful runs (23/40). Table 3 shows in columns marked with “CLK” the number of successful runs for a given instance and kicking strategy. For the smaller instances (`pcb3038` and smaller), the Geometric kicking performs worst regarding both approximation performance and average tour quality after reaching the time limit ( $10^4$  CPU seconds). Larger instances (`f13795` and larger) perform worst with the Random kicking strategy, the other strategies perform equally well with an advantage for the Random-walk strategy. In contrast, for instance `pla33810` the Random kicking strategy performs best, while the Random-Walk strategy is only the second best choice. For graphical examples see Figure 2. Table 4 shows the proximity of the average tours to the optimum (or Held-Karp lower bound) for different instances and kicking strategies after a given periode of time.

### 4.2 Distributed Chained Lin-Kernighan

By default, all instances were solved by the distributed algorithm using the Random-walk kicking strategy, which is the default kicking strategy in `linkern`, too. For most

smaller instances (less than  $10^4$  cities) all four kicking strategies were applied.

The best results were achieved with a distributed algorithm variant running on 8 nodes and using the double-bridge move for perturbation as described before. In the following discussion, this will be the default setup. For comparison, several instances were analyzed using a distributed algorithm that was (1) restricted on one single node, (2) running without DBMs or (3) running with both restrictions.

As shown in Table 3 the distributed algorithm finds the optimal solution for most instances up to `fn14461` in at least one run, for many instances in all 10 runs. In cases where not all runs were successful within 1000 CPU seconds, the results were already close to the optimum. Compared to the successfulness of CLK runs, the distributed algorithm has only in one case (`f11577` with Random kicking) fewer successful runs. The distributed algorithm can handle instances (e.g. `f13795`) very well (39/40 runs successful), which the standard CLK fails to solve every time within its time bound.

The approximation towards the optimum is faster with the distributed algorithm (DistCLK) compared with the original algorithm. As shown exemplarily in Figure 2 for instances `f11577` and `sw24978` (see [13] for plots of the other instances) the distributed version is clearly better than CLK for most instances. For the instance `f11577`, CLK gets stuck after about 150 seconds in local optima (9 runs in 22395, 1 run in 22256) that it cannot leave within the time bound (Figure 2(c)). The distributed variant, however, finds the optimum in 9 out of 10 runs in less than 300 CPU seconds per node. Instance `pr2392` was quite easy to solve for DistCLK, as all 10 runs find an optimal tour after at most 260 CPU seconds per node. In contrast, CLK found an optimal tour in only 4 out of 10, which require 3853 CPU seconds on average. For instance `pcb3038`, the average tour length over 10 CLK runs after  $10^4$  CPU seconds was about 0.06% above the optimum. DistCLK performed much better, reaching this tour quality already after 20 CPU seconds per node on average. Additionally, 9 out of 10 runs found the optimum after at most 1723 seconds. On instance `f13795` CLK got stuck in local optima again after 111 seconds on average. The distributed version solved this problem to optimality after at most 569 CPU seconds per node. For instance `fn14461`, CLK could have found better tours if given more time, as the 10 runs had found 9 different quality levels after reaching the time bound. For instance `usa13509`, neither of the algorithms found optimal tours. After  $10^4$  CPU seconds the average tour quality was only 0.021% above the optimum compared to CLK having a final tour quality of 0.090%. A similar behavior can be observed with instance `sw24978` (see Figure 2(d)). The distributed algorithm has a final average tour quality of 0.050% above the optimum (best is 0.033%), whereas CLK has a final av-

average tour quality of 0.088% (best is 0.064%). The final average tour length of the CLK algorithm is already reached after 2377.6 CPU seconds per node by the distributed algorithm. For instance `pla33810`, the average tour quality after the corresponding time limit of  $10^4$  seconds per CPU is 0.149% above the Held-Karp bound for the distributed algorithm. The CLK algorithm has a final average tour quality of 0.221% above this bound. This tour quality is reached by the distributed algorithm already after 670 CPU seconds per node on average. For the distributed variant with 8 nodes, about 14.9 times less CPU time is required to find this tour length.

#### 4.2.1 Variator Strength and Restarts

The variation and restarting strategy described in Section 2.3 might behave differently depending on random processes for the same instance in different runs. The following two example runs were selected out of ten simulation runs with instance `fi10639` with 8 nodes and the Random-Walk kicking strategy.

For run *A* only a weak perturbation was enough to enable the CLK algorithm to find a better tour. During the first 4952 CPU seconds 51 improving tours were found by the nodes. As after about 6600 seconds no new improvements were made, within a small time frame all eight nodes increased *NumPerturbations* to 2. Before requiring any further increase, a better tour was found (7858 seconds) by a node. As this tour was broadcasted in the net and improving the local best tours, the local *NumNoImprovements* variables were resetted, too. After about 9500 seconds *NumPerturbations* increased again as no new tour was found meanwhile. Short after the increase a better tour was found, which was improved only once before the time bound was reached. The final tour's length was 520627 which is 0.047% above the Held-Karp bound.

Run *B* showed that strong perturbations are necessary in some cases. For the first 3396 CPU seconds 45 improving tours were found by the nodes. Variable *NumPerturbations* was increased sequentially: After about 5020 seconds to level 2, after about 6700 seconds to level 3 and after 8370 seconds to level 4. A better tour was found by a node after 9337 seconds preventing a further increase of *NumPerturbations*. This tour was improved four more times resulting in a final tour of length 520584 (0.039% above Held-Karp bound).

The tour qualities of all the runs with the same parameters were between 520563 (0.035%) and 521002 (0.119%).

#### 4.2.2 Effects of Parallelization

To compare the effects of parallelization the subset of the test instances were run in setups with both 1 and 8 nodes, while keeping other setup parameters constant (e.g. the

Distance to Optimum	CPU time per node [sec]			Speed-up Factor
	ABCC-CLK	1 node	8 nodes	
Instance <code>pr2392</code>				
0.10%	8510.7	246.2	10.7	23.01
0.05%	–	421.1	24.2	17.40
0.00%	–	937.1	262.2	3.57
Instance <code>fi10639</code> (HK Bound instead of optimum)				
0.12%	3912.6	1183.4	188.8	6.27
0.10%	15183.3	2671.7	350.6	7.62
0.08%	–	6960.5	723.0	9.63

**Table 1. Speed-up with instances `pr2392`, `fi3795` and `fi10639`. Average over 10 runs each.**

Random-walk kicking strategy). Between two local CLK search steps the already described variable strength double-bridge move perturbation was performed. In case of the 8 node variant the locally improved tours were exchanged between neighboring nodes. Simulation results show, that the distributed algorithm can scale well with the number of nodes. For the following discussion, “speed-up factor” is the relation between the original CLK algorithm and the distributed algorithm regarding the total CPU time summed over all CPU nodes. For a setup with eight nodes, a speed-up factor of more than 8 means that the distributed algorithm required less total CPU time to reach a given result compared to the original CLK algorithm. This case occurs due to synergetic effects resulting from cooperation of nodes in the distributed algorithm.

In Table 1 a comparison between the original Chained LK algorithm and the distributed algorithm running on 1 or 8 nodes, respectively, is shown. For instance `pr2392`, the variant with 8 nodes is at the beginning more than twice as fast as expected from parallelization. It reaches a tour quality level of 0.1% above the optimum after 10.7 CPU seconds per node compared to 246.2 seconds for the distributed algorithm's single node variant (speed-up factor 23.01) and the original CLK algorithm with 8510.7 CPU seconds. For the quality level of 0.05% above the optimum the 8 node variant is still two times faster than the single node variant (speed-up factor 17.4), in respect to CPU seconds. Here, ABCC-CLK does not reach this level as well as the optimum within the given  $10^5$  second time limit. The parallel variant with 8 nodes requires about a quarter of the time of the single node variant (speed-up factor 3.57). This behavior depends on three runs in the parallel variant, that need between 110 and 260 seconds, while the other seven runs require less than 43 seconds to find the optimum. The medians over the optimum finding times for both variants are 71.2 seconds versus 596.5 seconds (factor 8.38) which suits the expectations from parallelization. As for instance `fi10639` (Figure 3(b)) no optimal solution is known, the Held-Karp bound was used to measure tour qualities. The first distance level of 0.12% above the Held-Karp bound for

this instance was reached after 1183 CPU seconds in the one node variant, compared to the eight node variant requiring 189 seconds. This speed-up of 6.27 gets improved subsequently. The tour quality of 0.10% is reached in average after 2672 seconds versus 351 seconds (speed-up factor 7.62). Finally, the quality level of 0.08% required a computation time of 6961 seconds for the single node variant and 723 seconds for the parallel variant resulting in a speed-up factor of 9.63. As shown above parallelization works for this distributed algorithm when comparing single versus multiple node variants.

### 4.3 Comparison with Related Work

For comparison with other TSP solvers, the running times of selected instances have been normalized to a 500 MHz Alpha processor as standardized for the 8th DIMACS Implementation Challenge for the TSP [18, 1]. The computational data for the following comparison with other TSP solver below has been taken from the same source. This normalization is generated by running a TSP benchmark tool comparing running times with those from the Alpha machine. The normalization factor differs for various instance sizes and ranges from 1.96 to 3.68 for the instances used in Table 2. The computation time presented for the distributed algorithm (columns marked with “DistCLK” in Table 2) are the average CPU times, multiplied by 8 (as results came from a system with 8 nodes) and scaled by the normalization factors.

LKH by Helsgaun [17] is an advanced Lin-Kernighan algorithm, as it uses sequential 5-exchange steps operating on neighborhoods restricted to 5 using an  $\alpha$ -nearness. The  $\alpha$ -values are calculated based on one-trees with modified weight matrices ( $\pi$ -values). LKH is known for good tour qualities, but requires long running times. For most of the compared instances, the distributed algorithm has on average better tours after the first iteration compared to the final tour lengths of Helsgaun’s LK. This is due to the fact, that the current tour length of the distributed algorithm is the best single node tour length (generated by the underlying CLK algorithm) within the network. But for this initial tour quality, the distributed algorithm requires significantly more time than LKH to reach its final tour quality level for smaller instances (up to *usa13509*). For the two larger instances, however, less time is required. The ratio between the computation times for both algorithms shifts towards the distributed algorithm for increasing instances size growing from 0.13 for instance *fn14461* and 0.50 (*usa13509*) to 2.87 (*p1a33810*) and 4.46 for instance *p1a85900*.

Walshaw presented a multi-level approach to solve TSP problems that embeds the Chained Lin-Kernighan algorithm, too [24]. For the data used here, the number of iterations of the CLK algorithm was set to  $10N$  ( $N$  is the num-

ber of instance cities). Walshaw’s Multi-level ( $MLC^N LK$ ) approach’s final tour qualities are worse compared to the tour qualities of the first iteration within the distributed algorithm, except for one case, but  $MLC^N LK$  requires significantly less time compared to the distributed algorithm for its first iteration. For one comparable case (instance *f13795*),  $MLC^N LK$  requires only 26 normalized CPU seconds to find a tour that’s quality is 0.54% above the optimum, whereas the distributed variant requires 938 seconds.

Cook and Seymour improve in their tour merging algorithm [10] results from independent runs of an underlying TSP solver (such as CLK or LKH) by merging the edges into a new graph and finding tours in this new graph. In an example using instance *r15934* and LKH, the union of 10 suboptimal tours contained the optimal tour which is found by Cook and Seymour’s algorithm within a very short period of time (compared to the time required for the LKH runs). For the data used here, the average over 5 runs of tour-merging using a branch decomposition with 10 CLK tours (12 quadrant neighbors, Don’t-Look-Bits,  $N$  iterations and Geometric kicking strategy) was used. To gain its good tour quality TM-CLK (Tour Merging using CLK) requires more time than the two heuristics above, but is still significantly faster than the distributed algorithm. E.g. for instance *pr2392*, TM-CLK requires only 93 seconds to find an optimal tour, contrary to the distributed algorithm that requires 7465. Currently, there is no data available for the other instances of this testbed, so the distributed algorithm may perform better for larger instances.

## 5 Conclusion

The proposed distributed algorithm improves the quality and performance of the original CLK algorithm in different ways. It allows to run the algorithm on multiple machines with significant speed-up. Communication costs are small compared to computational costs and therefore have no influence on the performance.

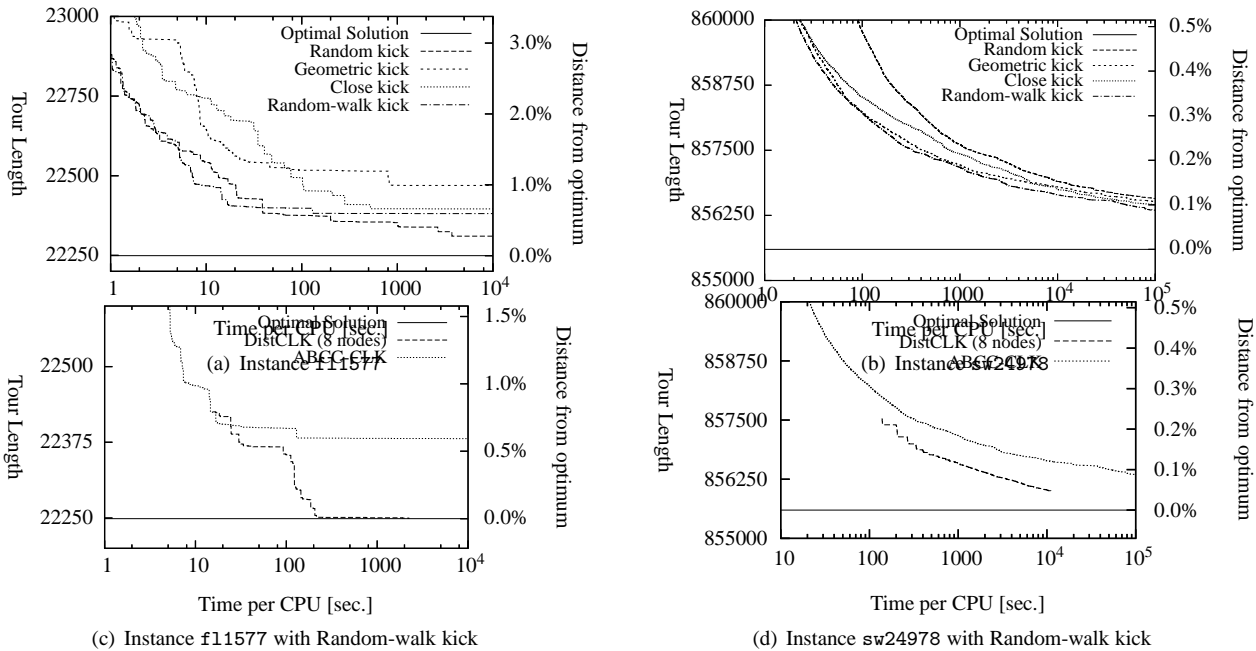
Distributed computation is common for exact algorithms. Using the approach proposed here, heuristic algorithms can profit from large computer clusters, too. By exchanging tour between nodes, nodes with worse tours can leave their neighborhood to enter more promising areas of the search space. This strategy alone might degenerate as all nodes got stuck in a local optimum. To increase the effectiveness of the distributed algorithm, a perturbation move with variable strength was introduced.

The comparison with other heuristic TSP solvers indicate that the distributed variant is best suited for large instances. Due to the fact that 8 machines were running in parallel the *absolute* time to find a good solution makes the distributed algorithm competitive to existing heuristics for real-world applications.

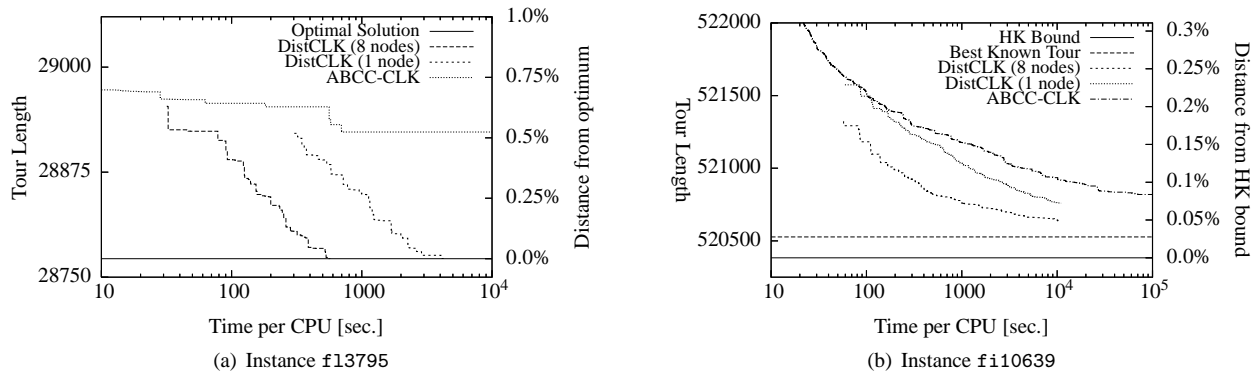
Instance	Helsgaun LK			Walshaw Multi-Level CLK			Cook&Seymour Tour Merging		
	Distance	LKH	DistCLK	Distance	MLC <sup>N</sup> LK	DistCLK	Distance	TM-CLK	DistCLK
pr2392	0.24%	34.87	< 205.37	0.52%	8.29	< 205.37	0.00%	92.50	7465.24
f13795	6.73%	74.06	< 914.73	0.54%	26.03	937.62	0.06%	509.69	16402.12
fn14461	0.07%	129.23	978.12	0.20%	22.38	< 584.41			
usa13509	0.21%	1133.81	< 2272.18	0.19%	148.49	< 2272.18			
pla33810	0.96%	7982.09	< 2785.89	1.08%	294.81	< 2785.89			
pla85900	1.25%	48173.84	< 9350.55	0.75%	1092.51	< 9350.55			

For cells marked with “<”, the distributed algorithm’s intermediate average results included only better tours, so the given value is the point of time when an average value was available for the first time.

**Table 2. Normalized computation time compared with other algorithms. “Distance” is the distance to the optimum (or Held-Karp Lower Bound for instances pla33810 and pla85900).**



**Figure 2. Upper figures: Relation between tour length and CPU time for the Chained Lin-Kernighan algorithm from Applegate et al. using different DBM kicking strategies. Lower figures: Relation between tour length and CPU time for the Distributed Chained Lin-Kernighan algorithm (DistCLK) compared with the results from the original CLK (ABCC-CLK).**



**Figure 3. Effects of parallelization running the distributed algorithms on a different number of nodes and optional perturbation for instances f13795 and fi10639.**

## References

- [1] 8th DIMACS Implementation Challenge: The Traveling Salesman Problem. <http://www.research.att.com/~dsj/chtsp/>.
- [2] Concorde TSP Solver. <http://www.tsp.gatech.edu/concorde.html>.
- [3] National Traveling Salesman Problems. <http://www.tsp.gatech.edu/world/countries.html>.
- [4] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding Cuts in the TSP (a Preliminary Report). Technical report, Rutgers University, Piscataway NJ, 1995.
- [5] D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. Technical Report 99887, Forschungsinstitut für Diskrete Mathematik, Universität Bonn, 1999.
- [6] A. Bachem and M. Wottawa. Parallelisierung von Heuristiken für große Traveling-Salesman-Probleme. In M. Baumann and R. Grebe, editors, *Transputer-Anwender-Treffen, Informatik Aktuell*, pages 204–213. Springer, 1993.
- [7] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. In J. F. Traub, editor, *Symposium on New Directions and Recent Results in Algorithms and Complexity*, page 441, Orlando, Florida, 1976. Academic Press.
- [8] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science*, 2009:46, 2001.
- [9] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM Press, 1971.
- [10] W. Cook and P. Seymour. Tour Merging via Branch-Decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.
- [11] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2:393–410, 1954.
- [12] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Masoulie. From Epidemics to Distributed Computing. *COMPUTER: IEEE Computer*, 2004.
- [13] T. Fischer and P. Merz. Embedding a Chained Lin-Kernighan Algorithm into a Distributed Algorithm. Interner Bericht 331/04, University of Kaiserslautern, Kaiserslautern, Germany, Aug. 2004.
- [14] M. M. Flood. The Travelling Salesman Problem. *Operations Research*, 4:61–75, 1956.
- [15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.
- [16] M. Gorges-Schleuter. Asparagos96 and the Traveling Salesman Problem. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, Indianapolis, USA*, pages 171–174. IEEE Press, 1997.
- [17] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.
- [18] D. S. Johnson and L. A. McGeoch. Experimental Analysis of Heuristics for the STSP. In Gutin and Punnen, editors, *The Traveling Salesman Problem and its Variations*. Kluwer Academic Publishers, 2002.
- [19] S. Lin. Computer Solutions Of The Traveling Salesman Problem. *Bell System Technical Journal*, 44:2245–2269, 1965.
- [20] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.
- [21] O. Martin, S. W. Otto, and E. W. Felten. Large-Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, 5:299–326, 1991.
- [22] B. Paechter, T. Bäck, M. Schoenauer, M. Sebag, A. E. Eiben, J. J. Merelo, and T. C. Fogarty. A distributed resource evolutionary algorithm machine (DREAM). In *Proceedings of the Congress on Evolutionary Computation 2000 (CEC 2000)*, pages 951–958. IEEE, IEEE Press, 2000.
- [23] G. Reinelt. TSPLIB – A Traveling Salesman Problem Library. *ORSA Journal On Computing*, 3:376–384, 1991.
- [24] C. Walshaw. A Multilevel Approach to the Travelling Salesman Problem. Mathematics research report 00/im/63, Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, Aug. 2000.

Instance	Kicking strategy							
	Random		Geometric		Close		Random-Walk	
	CLK	DistCLK	CLK	DistCLK	CLK	DistCLK	CLK	DistCLK
C1k.1	6/10	<b>10/10</b>	10/10	10/10	4/10	<b>10/10</b>	9/10	<b>10/10</b>
E1k.1	3/10	<b>10/10</b>	0/10	<b>10/10</b>	0/10	<b>10/10</b>	3/10	<b>10/10</b>
f11577	<b>5/10</b>	3/10	0/10	<b>9/10</b>	0/10	<b>8/10</b>	0/10	<b>8/10</b>
pr2392	9/10	<b>10/10</b>	0/10	<b>10/10</b>	2/10	<b>10/10</b>	4/10	<b>10/10</b>
pcb3038	0/10	<b>5/10</b>	0/10	<b>4/10</b>	0/10	<b>5/10</b>	0/10	<b>7/10</b>
f13795	0/10	<b>10/10</b>	0/10	<b>9/10</b>	0/10	<b>9/10</b>	0/10	<b>10/10</b>
fn14461	0/10	0/10	0/10	0/10	0/10	<b>1/10</b>	0/10	<b>1/10</b>

**Table 3. Number of CLK runs that found the optimum within a given time bound. For CLK, the limit was set to 10000 seconds and to 1000 seconds for the distributed variant with 8 nodes solving in parallel. Larger instances were omitted as both algorithms did not find optimal solutions for them.**

Instance	Kicking strategy							
	Random		Geometric		Close		Random-Walk	
	100 sec	10 <sup>4</sup> sec	100 sec	10 <sup>4</sup> sec	100 sec	10 <sup>4</sup> sec	100 sec	10 <sup>4</sup> sec
C1k.1	0.013%	0.007%	0.013%	OPT	0.031%	0.020%	0.005%	0.002%
E1k.1	0.035%	0.020%	0.068%	0.043%	0.035%	0.028%	0.024%	0.016%
f11577	0.569%	0.275%	1.206%	0.992%	1.102%	0.661%	0.670%	0.594%
pr2392	0.275%	0.050%	0.361%	0.283%	0.237%	0.105%	0.237%	0.093%
pcb3038	0.150%	0.070%	0.156%	0.081%	0.175%	0.077%	0.103%	0.060%
f13795	0.567%	0.567%	0.801%	0.579%	0.884%	0.581%	0.643%	0.524%
fn14461	0.121%	0.048%	0.089%	0.054%	0.093%	0.041%	0.098%	0.041%
fi10639	0.318%	0.160%	0.245%	0.138%	0.287%	0.144%	0.217%	0.106%
usa13509	0.268%	0.130%	0.234%	0.127%	0.229%	0.129%	0.204%	0.112%
sw24978	0.488%	0.153%	0.308%	0.140%	0.342%	0.136%	0.307%	0.122%
pla33810	0.508%	0.168%	0.563%	0.358%	0.592%	0.372%	0.519%	0.287%
pla85900	0.544%	0.209%	0.442%	0.232%	0.419%	0.231%	0.334%	0.160%

**Table 4. Distance of the average tour length compared to known optimum (Held-Karp bound for instances fi10639, pla33810 and pla85900) for CLK-ABCC after 100 and 10000 CPU seconds, respectively. Compare to Table 5.**

Instance	Kicking strategy							
	Random		Geometric		Close		Random-Walk	
	10 sec	10 <sup>3</sup> sec	10 sec	10 <sup>3</sup> sec	10 sec	10 <sup>3</sup> sec	10 sec	10 <sup>3</sup> sec
C1k.1	OPT	OPT					OPT	OPT
E1k.1	0.018%	OPT					◇	OPT
f11577	◇	0.022%	1.244%	0.002%	0.771%	0.004%	◇	0.006%
pr2392	◇	OPT	0.306%	OPT	0.227%	OPT	0.152%	OPT
pcb3038	◇	0.007%	0.136%	0.010%	0.127%	0.005%	◇	0.004%
f13795	◇	OPT	◇	0.014	◇	0.019%	◇	OPT
fn14461	◇	0.025%	◇	0.015%	◇	0.008%	◇	0.013%
fi10639	◇	0.113%	◇	0.086%	◇	0.072%	◇	0.116%
usa13509							◇	0.062%
sw24978	◇	0.171%	◇	0.116%			◇	0.116%
pla33810	◇	0.217%					◇	0.126%
pla85900							◇	0.182%

**Table 5. Distance of the average tour length compared to known optimum (Held-Karp bound for instances fi10639, pla33810 and pla85900) for DistCLK after 10 and 1000 CPU seconds per node, respectively. Compare to Table 4.**