Chained Lin-Kernighan for Large Traveling Salesman Problems

David Applegate • William Cook • André Rohe
Algorithms and Optimization Department, AT&T Labs—Research, 180 Park Avenue, P.O. Box 971, Florham Park, New Jersey 07932-0971, USA
Program in Applied and Computational Mathematics, Princeton University, Fine Hall, Washington Road, Princeton, New Jersey 08544-1000, USA
Forschungsinstitut für Diskrete Mathematik, Universität Bonn, Lennéstrasse 2, D-53113 Bonn, Germany
david@research.att.com • bico@math.princeton.edu • rohe@or.uni-bonn.de

We discuss several issues that arise in the implementation of Martin, Otto, and Felten's Chained Lin-Kernighan heuristic for large-scale traveling salesman problems. Computational results are presented for TSPLIB instances ranging in size from 11,849 cities up to 85,900 cities; for each of these instances, solutions within 1% of the optimal value can routinely be found in under one CPU minute on a 300 MHz Pentium II workstation, and solutions within 0.5% of optimal can routinely be found in under ten CPU minutes. We also demonstrate the scalability of the heuristic, presenting results for randomly generated Euclidean instances having up to 25,000,000 cities. For the largest of these random instances, a tour within 1% of an estimate of the optimal value was obtained in under one CPU day on a 64-bit IBM RS6000 workstation.
(*Networks-Graphs; Traveling Salesman; Heuristics*)

1. Introduction

Given the cost of travel between each pair of a finite number of cities, the *traveling salesman problem* (TSP) is to find the cheapest *tour* passing through all of the cities and returning to the point of departure. (We consider the symmetric version of the TSP, where the cost of travel between two cities does not depend on the direction we are traveling.) The simplicity of this model, coupled with its apparent intractability, makes it an ideal platform for exploring new algorithmic ideas, and it has long been a primary subject for the study of both exact and heuristic methods in discrete optimization. Treatments of the TSP can be found in Lawler et al. (1985), Reinelt (1994), Jünger et al. (1995), and Johnson and McGeoch (1997).

In heuristic approaches to the TSP, the goal is to find tours of low cost in reasonable amounts of computing time. One of the most successful methods proposed for this task is the simple and elegant local-search algorithm of Lin and Kernighan (1973). Their procedure is a "tour-improvement" method, in that it takes a given tour and attempts to modify it in order to obtain an alternative tour of lesser cost.

For two decades, Lin-Kernighan was the method of choice whenever high-quality tours were needed. Implementations of the algorithm are described in Bland and Shallcross (1989), Johnson (1990), Mak and Morton (1993), Perttunen (1994), Reinelt (1994), Schäfer (1994), Verhoeven et al. (1995), Johnson and McGeoch (1997), Rohe (1997), Neto (1999), and elsewhere.

An important, and widely adopted, part of Lin and Kernighan's overall tour-finding scheme is the repeated use of the basic Lin-Kernighan algorithm. The idea is simple: as long as computation time is available, by generating a new initial tour and

0899-1499/03/1501/0082$05.00
1526-5528 electronic ISSN

applying Lin-Kernighan, we have a chance of finding a tour that is cheaper than the best tour we have found thus far. This standard practice ended, however, with the publication of the work of Martin et al. (1991, 1992), who argued that repeatedly starting from new tours is an inefficient way to sample the locally optimal solutions produced by Lin-Kernighan. The alternative strategy they propose is to *kick* the Lin-Kernighan tour (that is, to perturb it slightly), and reapply the algorithm. If this effort produces a better tour, we discard the old Lin-Kernighan tour and work with the new one. Otherwise, we continue with the old tour and kick it again.

We refer to the algorithm of Martin et al. as *Chained Lin-Kernighan*, to match the *Chained Local Optimization* concept introduced in Martin and Otto (1996). Chained Lin-Kernighan offers a great performance boost over the original Lin-Kernighan scheme, as demonstrated in the computational study of Johnson (1990). Further results comparing the two approaches can be found in Reinelt (1994), Jünger et al. (1995), Codenotti et al. (1996), Johnson and McGeoch (1997), Hong et al. (1997), and Neto (1999). These papers offer good codes for a range of problem instances, but each of the implementations is impractical for very large examples having 250,000 or more cities. (A recent alternative to Chained Lin-Kernighan was proposed by Helsgaun 2000. His LKH code gives exceptionally good solutions for a wide range of problem sizes, but again it is not practical for very large problem instances.)

In this note, we discuss three issues involved in obtaining efficient implementations of Chained Lin-Kernighan for large instances, namely the breadth of the Lin-Kernighan search, the structure of the kick, and the choice of an initial tour. We report computational results for the TSPLIB set of test instances collected by Reinelt (1991, 1995), as well as results for random geometric instances having up to 25,000,000 cities.

The implementation we use in our study is available for research purposes as part of the *Concorde* TSP code of Applegate et al. (1998, 2001); Concorde uses Chained Lin-Kernighan as part of its exact solution procedure. The Concorde code is available at http://www.math.princeton.edu/tsp/. The research described in this paper contributed to the design of Concorde's Chained Lin-Kernighan implementation. It should be noted that the default settings in the 99.12.15 version of Concorde's Chained Lin-Kernighan differ slightly from those adopted in this paper (the details are provided in Section 7).

2. Chained Lin-Kernighan

Suppose we have an n-city TSP, with $c(i,j)$ representing the cost of travel between city i and city j. Consider a tour $(i_0, \dots, i_{n-1})$. If for some $0 \le p < q < n$, we have

$$c(i_{p-1}, i_p) + c(i_q, i_{q+1}) > c(i_{p-1}, i_q) + c(i_p, i_{q+1})$$

(the subscripts should be taken modulo n), then we can construct a better tour by "flipping" the subsequence $(i_p, \dots, i_q)$, that is, by moving to the tour

$$(i_0, \dots, i_{p-1}, i_q, i_{q-1}, \dots, i_{p+1}, i_p, i_{q+1}, \dots i_{n-1}).$$

The well-known 2-*opt* algorithm repeatedly searches for such tour flaws, and performs the corresponding flip operations to remove them.

Flip operations are also the basic building blocks of Lin and Kernighan's algorithm. Rather than searching for a single flip, however, Lin-Kernighan attempts to build a (possibly quite long) sequence of flips that taken together, one after another, end up at an improved tour. The point is that by allowing some of the intermediate tours to be more costly than the initial tour, Lin-Kernighan can go well beyond the point where 2-opt would terminate. If the Lin-Kernighan search procedure is successful in finding an improved tour, then the sequence of flips is made and a new search is begun. For details of Lin and Kernighan's algorithm, we refer the reader to the literature cited above; our computational study uses the implementation described in Applegate et al. (1999). The Applegate et al. implementation is tailored for Chained Lin-Kernighan, trading off tour quality for speed in several design decisions when compared to the original algorithm of Lin and Kernighan (1973). One of these design issues, the backtracking in the algorithm, is discussed in Section 3; other design issues are treated in Applegate et al. (1999).

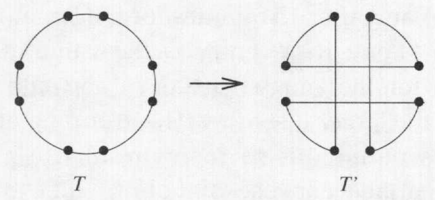

Figure 1 **A Double Bridge**

In addition to the basic Lin-Kernighan algorithm, Chained Lin-Kernighan calls for a method for perturbing a given tour. The mechanism proposed by Martin et al. (1991) consists of a sequence of flips that exchanges four edges in the tour for four other edges. The particular 4-*exchange* they use is illustrated in Figure 1. This kick was discussed in the original Lin and Kernighan (1973) paper in a different context. It serves the purposes of Martin et al. well: it is compact, it can alter the global shape of a tour, and standard implementations of Lin-Kernighan cannot find the set of flips needed to undo the exchanged edges. Martin et al. call this kick a *double bridge*.

The final ingredient in Chained Lin-Kernighan is the starting tour. For small instances this is not really an issue, since the algorithm is powerful enough to overcome just about any tour that it is given. For large instances, however, the choice of the initial tour can have an impact on Chained Lin-Kernighan's performance.

We will treat each of the three components of the algorithm in the discussion below. Our main test bed of problem instances consists of five of the seven TSPLIB instances having over 10,000 cities, together with a randomly generated Euclidean instance (points in the plane, with travel costs determined by the Euclidean distances, rounded to the nearest integer). The TSPLIB instances are geometric examples ranging in size from 11,849 cities up to 85,900 cities; the randomly generated instance has 100,000 cities. (We exclude the TSPLIB instances brd14051 and d15112 to give the problem suite a better balance between uniformly distributed instances and instances that are highly structured.) We also make a series of tests on large random Euclidean instances, ranging in size up to 25,000,000 cities. The random instances have integer coordinates drawn uniformly from the n by n square, where n is the number of cities.

Table 1 **Test Instances**

Name	Cities	Lower Bound	Tour	Gap
rl11849	11,849	923288	923288	OPTIMAL
usa13509	13,509	19982859	19982859	OPTIMAL
d18512	18,512	645198	645244	0.007%
pla33810	33,810	66005185	66050599	0.069%
pla85900	85,900	142307500	142395858	0.062%
r100000	100,000	225736239	225929775	0.086%

With the exception of rl11849 and usa13509, the optimal values for our main test instances are not known; the best upper and lower bounds that have been reported to Reinelt (1995) (as of December 2001) are listed in Table 1 (r100000 is not part of the TSPLIB). Each of the lower bound values given in the table was obtained with the Concorde code of Applegate et al. (1998, 2001). The rl11849, usa13509, and r100000 tours were obtained using Chained Lin-Kernighan together with the branch-width code of Cook and Seymour (1993); the d18512 and pla33810 tours were obtained using the LKH code of Helsgaun (2000) together with the branch-width code; the pla85900 tour was obtained by K. Helsgaun using a variant of LKH.

3. Level of Backtracking

The core of Lin and Kernighan's algorithm is a search technique for locating sequences of flip operations that appear to have a chance of leading to an improved tour. The search proceeds step by step, adding one flip after another to the sequence, with a built-in criterion for determining when the process should be stopped. A number of procedures for selecting flips have been proposed in Lin and Kernighan (1973), Mak and Morton (1993), Reinelt (1994), and Johnson and McGeoch (1997). In our implementation we use both the standard and alternate (second level) search methods described in Lin and Kernighan (1973), as well as the search method of Mak and Morton (1993); we impose a bound of 25 on the maximum length of an allowable flip sequence.

An essential part of the Lin-Kernighan algorithm is the use of backtracking to enhance the flip search procedure; Lin and Kernighan (1973) explore up to five choices for the first flip in the sequence, and for each of these flips they explore up to five choices for the second flip. At further levels of the search, Lin and Kernighan allow only a single choice, but it is natural to consider backtracking at any level of the search. We write (5, 5) to describe Lin and Kernighan's backtracking proposal, and in general we write $(b_1, b_2, \dots, b_p)$ to refer to a bound of b_k choices for flips at each of the $k = 1, \dots, p$ levels, with no backtracking permitted beyond the pth level. (We use the same choices for the main backtracking and for the alternate backtracking described in Lin and Kernighan 1973.) Implementations of Chained Lin-Kernighan must trade off the quality of tours produced by individual calls to Lin-Kernighan versus the speed with which the calls can be made; varying the bounds on backtracking is an effective means for regulating this tradeoff.

In Table 2, we report results for eight choices of backtracking. Each row in the table corresponds to a set of five trials over each of the six instances in our test suite. For each of the thirty trials, we calculate the percentage excess of the cost of the computed tour over the cost of the best tour known for the particular instance (as reported in Table 1), that is,

$$\frac{100 \cdot (\text{computed tour cost} - \text{best tour cost})}{\text{best tour cost}}.$$

The average of the thirty values, calculated at each of three points in the runs, is reported in Table 2; the "Short Run" entry is the value after one minute for the three smaller instances (rl11849, usa13509, d18512) and after three minutes for the three larger instances (pla33810, pla85900, and r100000), the "Medium Run" entry is the value after five minutes for the smaller instances and after 15 minutes for the larger instances, and the "Long Run" entry is the value after thirty minutes for the smaller instances and after 90 minutes for the larger instances. The grouping of problem instances permits us to report results in a compact fashion, at the expense of hiding some differences based on problem structure and size (the grouping does, however, allow one to see trends in the impact of the design choices). Even with 30 trials, there is still significant variance in the overall test results, but the tests do appear to permit the two digits of accuracy we report in the tables.

Table 2 Level of Backtracking

Search	Short Run	Medium Run	Long Run
(2,1)	0.56%	0.36%	0.27%
(2,2)	0.50%	0.33%	0.24%
(5,5)	0.48%	0.29%	0.21%
(4,3,2)	0.47%	0.29%	0.21%
(4,3,3,2)	0.49%	0.28%	0.20%
(4,3,3,2,2)	0.51%	0.29%	0.21%
(12,12)	0.59%	0.35%	0.22%
(4,4,3,3,2,2)	0.64%	0.31%	0.21%
(5,5,4,4,3,3,2,2)	1.35%	0.61%	0.30%

The tests were carried out on a 300 MHz Intel Pentium II workstation with 256 megabytes of memory. The code was compiled with the GNU gcc 2.7.2.1 compiler, using the –O3 option. In these tests we use the two-level list tour data structure described in Chrobak et al. (1990) and in Fredman et al. (1995). The tests use the "250-geometric" kick and the "Quick-Borůvka" initial tour described in Sections 4 and 5.

Lin and Kernighan's (5, 5) rule performs quite well, but for longer runs the results in Table 2 indicate that a slightly wider search is preferable; we adopt the (4, 3, 3, 2)- breadth as the default value in our implementation.

4. Choice of the Kick

In their computations, Martin et al. (1992) generate double bridges at random, but only use as kicks those that involve pairs of edges of relatively small total cost. Johnson (1990) and Johnson and McGeoch (1997), on the other hand, drop this restriction on the edge costs and simply use random double-bridge kicks. An argument in favor of this latter strategy is that cost-restricted kicks tend to be local in nature and might, therefore, cause the algorithm to get stuck in some undesirable global configuration. From another perspective, however, the local nature of the cost-restricted kicks can be seen as an advantage: the calls to the Lin-Kernighan algorithm will be both faster (the costly edges in the random flips tend to lead to long sequences of flips in the Lin-Kernighan searches) and more likely to be successful (since we are not causing havoc in the tour, there is a greater chance that Lin-Kernighan can find a way to correct the perturbation that we have made). As we shall see below, for large instances this second argument is dominant—cost restricted kicks are much more effective than random kicks.

To obtain cost-restricted kicks for large instances, we need an alternative to the process of examining kicks at random used by Martin et al. The difficulty with their procedure is that only a very small fraction of random kicks would be accepted by any reasonably small cost threshold. To overcome this, we propose three direct construction procedures for "local" kicks. In each of our procedures, we employ a method proposed by Rohe (1997) for selecting the first edge of a kick; the idea is to start the double bridge at a city v that appears to be out of place in the tour. We describe this method below.

When we search for a kick we have in hand a current tour T through the cities; for convenience we fix an orientation of T. For a city v, let $next(v)$ denote the city immediately following v in tour T and let $near(v)$ denote the city w that minimizes $c(v, w)$, the cost of travel from v to w. To select the first edge removed by the kick, we consider a small fraction of the cities (selected randomly) as candidates for v and choose the one that maximizes

$$c(v, next(v)) - c(v, near(v)).$$

The first edge of the double bridge will be $(v, next(v))$. To complete the construction, we choose the remaining three edges to be close to v, as we describe below.

Our first selection procedure examines, for some constant α, a random sample of αn cities (where n is the number of cities in the TSP instance). We attempt to build a double bridge using three edges of the form $(w, next(w))$, for cities w that are amongst the six nearest neighbors of v, distinct from $next(v)$, in the random sample. We call the double bridges found by this procedure *close kicks*. Note that as we increase α, the kicks we obtain with this method are increasingly local in nature.

A second, perhaps more natural procedure, is to complete the double bridge from edges of the form $(w, next(w))$, where w is chosen at random amongst the k cities nearest to v. By varying k, we can get very local kicks or kicks similar to those generated purely at random. Notice, however, that these kicks are time-consuming to compute in general instances, since we would be required to examine every city in order to obtain the k nearest cities. In geometric instances, however, we can use *kd*-trees (see Bentley 1992) to examine the nearest sets efficiently. We call the double bridges found in this way *geometric kicks*.

Our third procedure is based on taking three random walks from city v in a prescribed *neighbor graph* that is used in the flip-selection procedure of Lin-Kernighan; the double-bridge edges we consider are of the form $(w_i, next(w_i))$, where w_i is the city reached in walk i, for $i = 1, 2, 3$. By varying the number of steps taken in the walks, we can control the locality of the resulting kicks; the double bridges found with this process are called *random-walk kicks*. In our implementation, the neighbor graph consists of the three least costly edges in each of the four geometric quadrants (for two-dimensional geometric instances, like those in our test suite) around each city. This graph was proposed by Miller and Pekny (1995) in the context of two-matching algorithms, and Johnson and McGeoch (1997) have shown that it is an effective neighbor graph for Chained Lin-Kernighan. (We developed the random-walk kick as an effective way to obtain cost-restricted kicks on instances having several million cities, where the close and geometric kicks become costly to compute.)

In Table 3, we compare random, close, geometric, and random-walk kicks, as well as random kicks where we use Rohe's rule for choosing the initial edge. (The tests use (4, 3, 3, 2)-breadth, as we

Table 3 Kicking Strategy

Kick	Short Run	Medium Run	Long Run
random	0.78%	0.43%	0.23%
random, long first edge	0.77%	0.41%	0.23%
close ($\alpha = 0.003$)	0.54%	0.29%	0.17%
close ($\alpha = 0.01$)	0.49%	0.28%	0.18%
close ($\alpha = 0.03$)	0.52%	0.33%	0.22%
geometric ($k = 100$)	0.53%	0.33%	0.24%
geometric ($k = 250$)	0.50%	0.28%	0.20%
geometric ($k = 1000$)	0.55%	0.29%	0.17%
random-walk (25 steps)	0.59%	0.44%	0.36%
random-walk (50 steps)	0.49%	0.26%	0.17%
random-walk (100 steps)	0.55%	0.28%	0.17%

described in the previous section; the results are again the average over thirty trials, that is, five trials over each of the six instances in our test suite; we use the "Quick Borůvka" initial tour described in the next section.) The results indicate a clear preference for the cost-restricted kicks.

Our reported results are restricted to double-bridge kicks, but there is no strong argument favoring these over other kicking structures. Hong et al. (1997) tested the use of k-exchange kicks for k varying from two up to 50; they report that several values of k work well on their 318-city, 532-city, and 800-city test instances. A quite different kick was studied in Codenotti et al. (1996), involving a perturbation of the x, y-coordinates of the cities. In our implementation, however, double bridges appear to perform at least as well as any alternatives that we have tried; we will comment further on the Codenotti et al. scheme in Section 6 below.

5. The Initial Tour

In this section we discuss the choice of a starting tour for Chained Lin-Kernighan. Although the algorithm behaves very well over a wide range of tours, we will see that its performance can be influenced by the structure of the initial solution. In our tests below, we use 50-step random-walk kicks and our default (4, 3, 3, 2)-breadth in Lin-Kernighan.

In Table 4 we report results for a number of different starting tours, using our standard suite of six test instances. Three of the tours, "Random," "Nearest Neighbor," and "Christofides," are well known in the TSP literature. The fourth starting tour, "Greedy," is produced by a heuristic developed by Bentley (1992) (he calls it "multiple fragment"); it is used as a starting tour in Johnson and McGeoch (1997) and in Codenotti et al. (1996). The remaining two tours, "Quick-Borůvka" and "HK-Christofides," are described below.

Table 4 Initial Tour

Tour	Short Run	Medium Run	Long Run
Random	0.62%	0.31%	0.18%
Nearest Neighbor	0.58%	0.31%	0.18%
Christofides	0.50%	0.27%	0.17%
Greedy	0.55%	0.29%	0.17%
Quick-Borůvka	0.49%	0.26%	0.17%
HK-Christofides	0.44%	0.23%	0.15%

Quick-Borůvka is motivated by the minimum-weight spanning tree algorithm of Borůvka (1926). In Quick-Borůvka, we build a tour edge by edge. The construction begins (for geometric instances) by sorting the cities of the TSP according to their first coordinate. We then process the cities in order, skipping those cities that already meet two edges in the partial tour we are building. To process city x, we add to the partial tour the least costly edge meeting x that is permissible (so we do not consider edges that meet cities having degree two in the partial tour, nor edges that create subtours); this procedure can be implemented efficiently using kd-trees. As a stand-alone heuristic, quick-Borůvka produces tours that are of slightly worse quality than Greedy, but it requires less time to compute and it appears to work well together with Chained Lin-Kernighan.

The final tour in the table is *HK-Christofides*. The standard Christofides heuristic (Christofides 1976) works with a minimum-cost spanning tree, combining it with a matching on the cities having odd degree in the tree; the tour is produced from the union of the tree and matching via a "short-cutting" technique. (A description of the algorithm can be found in Johnson and Papadimitriou 1985. In our implementation, we use the Lin-Kernighan matching heuristic of Rohe 1997 to find the matching, rather than using an exact matching algorithm.) In HK-Christofides, we do not start with the minimum spanning tree, but rather a tree produced by running the iterative algorithm of Held and Karp (1971) (for computing lower bounds for TSP instances). The Held-Karp procedure computes a sequence of spanning trees, using costs that are adjusted at each iteration according to the degree of the nodes in the current tree. If a node has degree less than 2, then the cost of each edge meeting the node is decreased; if a node has degree greater than 2, the cost of each edge meeting the node is increased. The number of trees in the Held-Karp sequence depends on the values of several parameters that must be chosen in the algorithm, but even a relatively short run will usually produce a tree having a far greater number of nodes of degree 2 than does the initial minimum-cost spanning tree; this feature makes it an attractive tree for starting the Christofides heuristic.

The results reported in Table 4 demonstrate that the initial tour can indeed have an impact on the performance of Chained Lin-Kernighan (the table again reports the average over 30 trials, as in the previous sections). Random starting tours and tours produced by the Nearest Neighbor heuristic exhibit the worst behavior. The Christofides, Greedy, and Quick-Borůvka heuristics all provide much better approximations to optimal tours, and this results in good performance in Chained Lin-Kernighan in each of these cases. Finally, with the help of the excellent tour approximation provided by the Held-Karp iterative procedure, Chained Lin-Kernighan with HK-Christofides stands out in our tests as the overall winner in terms of tour quality. It must be noted, however, that the "Short," "Medium," and "Long" time checks include only the time spent in Chained Lin-Kernighan, and not the time needed to compute the starting tours. The time (in seconds) needed to compute the initial tours for the 100,000-city random problem are

Random	NN	Greedy	Q-Borůvka	Christofides	HK-Christofides
0.0	1.3	8.2	2.0	13.6	1621.8

respectively. The large value for the HK-Christofides tour makes in unsuitable for short runs (unless the Held-Karp lower bound is also needed for the given application), but its exceptional performance on long runs indicates that it should be a candidate for the starting tour if very high quality tours are required.

We made an attempt at constructing a fast HK-Christofides tour by performing only a small fixed number of iterations of the Held-Karp procedure (and working only on a sparse graph), but the resulting Chained Lin-Kernighan implementation was not significantly better than with the basic Christofides tour. Based on the results in Table 4, we adopt Quick-Borůvka as our default starting tour—it gives results similar in quality to those obtained using Christofides or Greedy, and it requires less time to compute.

6. Computational Results

We describe a series of tests aimed at demonstrating the range of application of Chained Lin-Kernighan. Throughout this section, we adopt our default (4, 3, 3, 2)-breadth for Lin-Kernighan.

6.1. TSPLIB Instances

We now consider the full set of seven TSPLIB instances having over 10,000 cities. The values of the best known tours and lower bounds are

Name	Cities	Lower Bound	Tour	Gap
brd14051	14,051	469374	469388	0.003%
d15112	15,112	1573084	1573084	OPTIMAL

for the two examples that were not part of our main test suite.

In Table 5 we report benchmark results for our Chained Lin-Kernighan implementation over the TSPLIB instances and the random Euclidean instance r100000; the average number of iterations used at each of the checkpoints is reported in Table 6. Unlike the tables in the previous sections, Table 5 reports the %-excess over the known lower bounds for the instances (reported above and in Table 1), so the results are guaranteed to be within the stated percentage of the optimal values. For each of the eight instances we made five runs, using Quick-Borůvka as the starting tour; the reported values are the average tour qualities achieved at the indicated checkpoints. To obtain good performance over a wide range of running times, we adopt a hybrid kick where we use a 50-step random walk for the first n iterations and switch to a 100-step random walk thereafter; this is important for the quality of the short runs, as can be seen from the results reported earlier in Table 3.

The CPU-time checkpoints in Table 5 are the full times for the runs, including the start-up time to compute the initial tour and to compute the neighbor

Table 5 **Excess over Lower Bounds**

Name	1 Minute	10 Minutes	1 Hour	4 Hours	24 Hours
rl11849	0.51%	0.29%	0.24%	0.22%	0.19%
usa13509	0.45%	0.22%	0.15%	0.13%	0.09%
brd14051	0.49%	0.16%	0.11%	0.09%	0.07%
d15112	0.39%	0.17%	0.11%	0.07%	0.06%
d18512	0.42%	0.18%	0.12%	0.09%	0.07%
pla33810	0.73%	0.41%	0.30%	0.26%	0.23%
pla85900	0.85%	0.34%	0.25%	0.21%	0.16%
r100000	1.63%	0.48%	0.28%	0.22%	0.17%

Table 6 Iterations of Chained Lin-Kernighan

Name	1 Minute	10 Minutes	1 Hour	4 Hours	24 Hours
rl11849	2,916	28,782	163,199	650,199	3,968,180
usa13509	1,968	21,818	118,745	473,399	2,858,160
brd14051	3,228	32,520	185,799	742,199	4,468,970
d15112	2,701	28,837	162,199	647,999	3,903,349
d18512	3,454	36,192	200,271	797,399	4,831,603
pla33810	3,205	37,393	193,862	770,399	4,653,300
pla85900	3,020	38,920	203,828	757,799	4,445,160
r100000	806	18,660	124,176	444,607	2,627,547

graph. The average start-up time (in seconds) for the eight test instances are

rl11849	usa 13509	brd 14051	d15112	d18512	pla 33810	pla 85900	r100000
1.9	2.5	2.2	2.5	3.0	4.3	11.6	23.4

respectively. These rather small values do not have a significant effect on the results for the higher checkpoints, but they do affect the "1 Minute" times. Improved one-minute results could be obtained by using, for example, a fast Delaunay triangulation code to create a neighbor graph (which works quite well for short runs). To illustrate this, we used the "sweep2" code of Fortune (1987, 1994) to compute a Delaunay triangulation for r100000 in 3.8 seconds, and obtained an excess of 1.410% after a total of 1 minute of CPU time. For longer runs, however, our default "3-quadrant" neighbor graph produces more reliable results than the (sparser) Delaunay triangulation.

6.2. Very Large Instances

The largest instance in our standard test suite has 100,000 cities; the implementation, however, can easily handle much larger examples. We demonstrate this by considering a series of randomly generated Euclidean instances, ranging in size from 10,000 cities up to 25,000,000 cities. For each problem size, we consider a set of ten instances (except in the largest two cases, where we consider only single instances), using a 50-step random-walk kick and performing n iterations, where n is the number of cities. The results of the tests are reported in Table 7. These runs were carried out on a 64-bit IBM RS6000, Model 43-P 260 workstation equipped with four gigabytes of memory (this machine is approximately 1.2 times faster than our 300 MHz Pentium II workstation). Due to storage considerations, for the 25,000,000-city instance we use the six nearest neighbors to each city as our neighbor graph, rather than the usual three-quadrant graph; this is the cause for both its faster-than-expected running time and its slightly worse-quality tour. In these tests, we use the splay-tree tour data structure described in Applegate et al. (1990) and in Fredman et al. (1995); this data structure has better asymptotic behavior than the two-level list implementation.

Table 7 Geometric Instances Run on an IBM RS6000, Model 43-P 260

Number of Cities	Trials	Tour Quality Ratio	CPU Time (seconds)
10,000	10	0.7189	260
25,000	10	0.7171	715
100,000	10	0.7156	3,300
250,000	10	0.7152	9,059
1,000,000	10	0.7148	37,232
2,500,000	10	0.7145	94,110
10,000,000	1	0.7143	396,362
25,000,000	1	0.7146	698,628

Each random instance in our test consists of n cities with integer coordinates drawn uniformly from the n by n square; the travel costs are the Euclidean distances rounded to the nearest integer value. The "Tour Quality Ratio" reported in Table 7 is the average cost of the tours found by Chained Lin-Kernighan, divided by $n\sqrt{n}$. This ratio relates to the result of Beardwood et al. (1959), who showed that for random Euclidean instances in the unit square (using the Euclidean distances as the edge costs) the ratio of the optimal tour length to $\sqrt{n}$ converges almost surely to a constant β_{OPT}. (For a discussion of this result, see Karp and Steele 1985. Note that the decrease in the ratios reported in the table as n increases is due to the general rate at which optimal tour lengths converge to β_{OPT}.) Johnson et al. (1996) give an empirical estimate of β_{OPT} using a combination of exact and heuristic TSP algorithms; their conclusion is that $\beta_{OPT} = 0.7124 \pm 0.0002$ (similar values were also obtained by Percus and Martin 1996 and by Cerf et al. 1997). Taking this value as a rough estimate of the optimal tour cost for the 25,000,000-city example, we have that the final tour produced by Chained Lin-Kernighan (after eight CPU days) is approximately 0.3% above optimal.

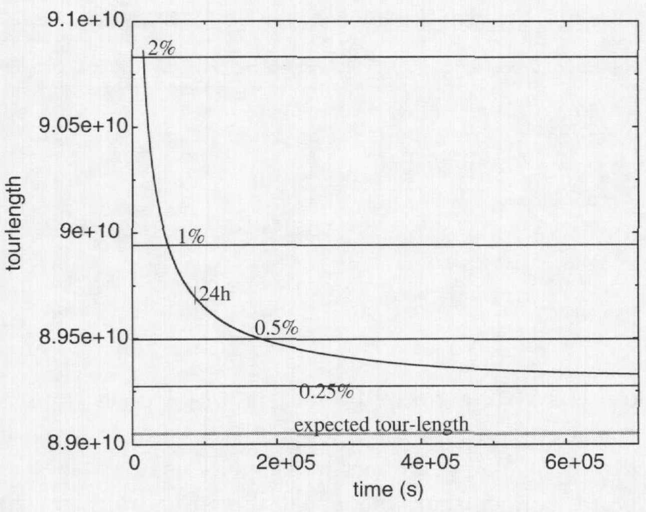

Figure 2 Run on 25,000,000-City Instance

A history of the 25,000,000-city run is given in Figure 2. After 24 CPU hours, the cost of the tour is easily within 1% of the estimate of the optimal value.

6.3. Comparison with Earlier Implementations

Most of the previous work on large TSP instances has dealt with fast heuristics that aim for tours that are within several percent of optimal; a nice collection of results of this type can be found in Bentley (1992), including a study of a 1,000,000-city instance. Our Chained Lin-Kernighan implementation is not meant to compete with these heuristics, but rather to provide a method to obtain much higher quality tours (even for very large instances) when a modest amount of additional computing time is available.

There have been two earlier studies dealing with large-scale implementations of Chained Lin-Kernighan, carried out by Codenotti et al. (1996) and Johnson and McGeoch (1997). We will comment briefly on the relationship between our results and the findings reported in the papers of these two groups.

As we mentioned in Section 4, Codenotti et al. (1996) employ an alternative kicking structure, involving a perturbation of the cities (rather than a perturbation of the tour) designed to cause the current tour to be no longer locally optimal. Their implementation is aimed at large instances and they include data for random Euclidean examples with up to 100,000 cities. Following the suggestion of Johnson et al. (1996), Codenotti et al. record their results as the percentage excess of the cost of the tours over the Held-Karp lower bound for the given instances. Their runs were made on a Silicon Graphics R4000 Indigo, which is a considerably slower machine than our standard 300 MHz Pentium II workstation. To get an approximate scaling factor between the two computers, we benchmarked our implementation on a Sun Microsystems Sparc 10, Model 41, which is of similar vintage to the R4000 Indigo and (for this type of computing) roughly the same speed. Our code is 6.2 times faster on the Pentium II than on the Sparc 10, and we use this as the scaling factor between the Pentium II and R4000 Indigo. On tests of 100,000-city instances, Codenotti et al. obtained tours with average Held-Karp excess of 1.65% (in the best of three reported results) in 10 hours of computer time. To compare our implementation with their results, we made trials on five different 100,000-city instances, stopping our code when it reached a tour having cost no greater than 1.65% over the Held-Karp bound, and recording the total amount of CPU time used (including the start-up time). The average time over the five trials is reported in the first line of Table 8. The approximate speed-up (using the 6.2 scaling factor between the machine times) for our Chained Lin-Kernighan implementation is 36.9x, that is, the new code achieved the target value in 36.9 times less CPU time. The relatively poor performance of the Codenotti et al. heuristic may be due to both the global nature of their kicks (for example, their kick makes it difficult to limit the number of cities that need to be considered as starting points for flip sequences in the Lin-Kernighan searches, as in the "don't look" bits used by Johnson and McGeoch 1997), and their more aggressive use of Lin-Kernighan (they permit the algorithm to work very long in each iteration).

The excellent survey paper of Johnson and McGeoch (1997) contains a section on a "Production-mode Iterated Lin-Kernighan" that includes results

Table 8 Comparison on 100,000-City Random Instances

Target HK-Excess	CPU Time	Speed-Up	Reference Paper
1.65%	157.3	36.9x	Codenotti et al. (1996)
1.31%	298.4	7.8x	Johnson and McGeoch (1997)
1.08%	606.9	11.5x	Johnson and McGeoch (1997)

for instances with up to 100,000 cities. Johnson and McGeoch use the term *Iterated Lin-Kernighan* to refer to the restricted case of Chained Lin-Kernighan where random double-bridge moves are used as kicks, and no probabilistic acceptance of tours is allowed. (See also Johnson 1990.) Their runs were carried out on a Silicon Graphics Challenge L/150 (with a 150 MHz R4400 processor). To get a rough translation between this machine and our Pentium II, we benchmarked our code on a Silicon Graphics Indigo 2 (with a 250 MHz R4400 processor). Our code is 2.6 time faster on the Pentium II than on the Indigo 2; scaling this by 250/150 to adjust for the speed of the R4400 processors, we have an approximate factor of 4.4 between the timings in Johnson and McGeoch (1997) and the results on our 300 MHz Pentium II.

In Table 8 we compare (using the 4.4 factor) our implementation with Johnson and McGeoch's two reported values for 100,000-city random instances; their implementation achieves an excess of 1.31% over the Held-Karp bound after 10,200 seconds and an excess of 1.08% after 30,700 seconds. The values reported in Table 8 are the average times over five test instances; we obtained average speed-ups of 7.8x and 11.5x in the two tests.

7. Conclusions

The reported computational results demonstrate that Chained Lin-Kernighan is suitable for use on even very large test instances.

For large instances, our tests indicate that it is important to consider the use of cost-restricted kicks, rather than the random double-bridge kicks adopted in Johnson (1990) and Johnson and McGeoch (1997); a good choice is a k-step random-walk kick, which is effective, simple to implement, and requires time depending only on the choice of k (independent of the number of cities). For a starting tour, Christofides, Greedy, and Quick-Borůvka all provide good results; if a Held-Karp tree is available then it is also worthwhile to consider using HK-Christofides. Finally, it is important to tune the Lin-Kernighan heuristic to respond to the needs of Chained Lin-Kernighan; in our implementation we adopt the $(4, 3, 3, 2)$-breadth to give a modest-width search in the basic algorithm.

The implementation we adopted in Section 6 differs in two ways from the default settings used in the 99.12.15 release of Concorde's Chained Lin-Kernighan solver. Firstly, the neighbor graph consists of the three least costly edges in each of the four geometric quadrants, whereas Concorde uses only the two least costly edges in each quadrant. This difference can be set in Concorde by using the "-q 3" command-line option. Secondly, we use the hybrid 50-step/100-step random-walk kick, whereas Concorde uses the 250-geometric kick. The 99.12.15 version of Concorde does not include code for random-walk kicks, since this idea was developed after the code was released; we will incorporate random-walk kicks in future releases of Concorde.

Acknowledgments
The authors thank David Johnson and an anonymous referee for helpful remarks on an earlier draft of this paper. William Cook was supported by ONR Grant N00014-01-1-0058.

References

Applegate, D., R. Bixby, V. Chvátal, W. Cook. 1998. On the solution of traveling salesman problems. *Documenta Mathematica Journal der Deutschen Mathematiker-Vereinigung, International Congress of Mathematicians*, 645–656.

Applegate, D., R. Bixby, V. Chvátal, W. Cook. 1999. Finding tours in the TSP. Report Number 99885, Research Institute for Discrete Mathematics, Universität Bonn, Bonn, Germany.

Applegate, D., R. Bixby, V. Chvátal, W. Cook. 2001. TSP cuts which do not conform to the template paradigm. M. Jünger, D. Naddef, eds. *Computational Combinatorial Optimization*. Springer, Heidelberg, Germany, 261–304.

Applegate, D., V. Chvátal, W. Cook. 1990. Lower bounds for the travelling salesman problem. *TSP '90*. Technical report CRPC-TR90547, Center for Research in Parallel Computing, Rice University, Houston, TX.

Beardwood, J., J. H. Halton, J. M. Hammersley. 1959. The shortest path through many points. *Proceedings of the Cambridge Philosophical Society* **55** 299–327.

Bentley, J. L. 1992. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing* **4** 387–411.

Bland, R. G., D. F. Shallcross. 1989. Large traveling salesman problems arising from experiments in X-ray crystallography: A preliminary report on computation. *Operations Research Letters* **8** 125–128.

Borůvka, O. 1926. On a certain minimal problem (in Czech). *Práce Moravské Přírodovědecké Společnosti* **3** 37–58.

Cerf, N. J., J. Boutet de Monvel, O. Bohigas, O. C. Martin, A. G. Percus. 1997. The random link approximation for the Euclidean traveling salesman problem. *Journal de Physique I* **7** 117–136.

Christofides, N. 1976. Worst-case analysis of a new heuristic for the travelling salesman problem. Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.

Chrobak, M., T. Szymacha, A. Krawczyk. 1990. A data structure useful for finding Hamiltonian cycles. *Theoretical Computer Science* **71** 419–424.

Codenotti, B., G. Manzini, L. Margara, G. Resta. 1996. Perturbation: an efficient technique for the solution of very large instances of the Euclidean TSP. *INFORMS Journal on Computing* **8** 125–133.

Cook, W., P. D. Seymour. 1993. An algorithm for the ring-routing problem. Technical Memorandum, Bellcore, Morristown, NJ.

Fortune, S. J. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica* **2** 153–174.

Fortune, S. J. 1994. sweep2. Computer code (in the C programming language). www.netlib.org/voronoi/sweep2.

Fredman, M. L., D. S. Johnson, L. A. McGeoch, G. Ostheimer. 1995. Data structures for traveling salesmen. *Journal of Algorithms* **18** 432–479.

Held, M., R. M. Karp. 1971. The traveling-salesman problem and minimum spanning trees: part II. *Mathematical Programming* **1** 6–25.

Helsgaun, K. 2000. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research* **126** 106–130. The LKH code is available at http://www.dat.ruc.dk/~keld/research/LKH/.

Hong, I., A. B. Kahng, B. Moon. 1997. Improved large-step markov chain variants for the symmetric TSP. *Journal of Heuristics* **3** 63–81.

Johnson, D. S. 1990. Local optimization and the traveling salesman problem. *Proceedings 17th Colloquium of Automata, Languages, and Programming, Lecture Notes in Computer Science*, No. 443. Springer-Verlag, Berlin, Germany, 446–461.

Johnson, D. S., L. A. McGeoch. 1997. The traveling salesman problem: a case study in local optimization. E. H. L. Aarts, J. K. Lenstra, eds. *Local Search in Combinatorial Optimization*. Wiley, New York, 215–310.

Johnson, D. S., L. A. McGeoch, E. E. Rothberg. 1996. Asymptotic experimental analysis for the Held-Karp traveling salesman bound. *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. Association for Computing Machinery, New York, 341–350.

Johnson, D. S., C. H. Papadimitriou. 1985. Performance guarantees for heuristics. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys, eds. *The Traveling Salesman Problem*. Wiley, Chichester, U.K., 145–180.

Jünger, M., G. Reinelt, G. Rinaldi. 1995. The traveling salesman problem. M. Ball, T. Magnanti, C. L. Monma, G. Nemhauser, eds. *Handbook on Operations Research and Management Sciences: Networks*. North-Holland, Amsterdam, The Netherlands, 225–330.

Karp, R. M., J. M. Steele. 1985. Probabilistic analysis of heuristics. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys, eds. *The Traveling Salesman Problem*. Wiley, Chichester, U.K., 181–205.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys. 1985. *The Traveling Salesman Problem*. Wiley, Chichester, U.K.

Lin, S., B. W. Kernighan. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research* **21** 498–516.

Mak, K.-T., A. J. Morton. 1993. A modified Lin-Kernighan traveling-salesman heuristic. *Operations Research Letters* **13** 127–132.

Martin, O. C., S. W. Otto. 1996. Combining simulated annealing with local search heuristics. *Annals of Operations Research* **63** 57–75.

Martin, O., S. W. Otto, E. W. Felten. 1991. Large-step Markov chains for the traveling salesman problem. *Complex Systems* **5** 299–326.

Martin, O., S. W. Otto, E. W. Felten. 1992. Large-step Markov chains for the TSP incorporating local search heuristics. *Operations Research Letters* **11** 219–224.

Miller, D. L., J. F. Pekny. 1995. A staged primal-dual algorithm for perfect b-matching with edge capacities. *ORSA Journal on Computing* **7** 298–320.

Neto, D. 1999. *Efficient Cluster Compensation for Lin-Kernighan Heuristics*. Ph.D. thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada.

Percus, A. G., O. C. Martin. 1996. Finite size and dimensional dependence in the Euclidean traveling salesman problem. *Physical Review Letters* **76** 1188–1191.

Perttunen, J. 1994. On the significance of the initial solution in travelling salesman heuristics. *Journal of the Operational Research Society* **45** 1131–1140.

Reinelt, G. 1991. TSPLIB—A traveling salesman library. *ORSA Journal on Computing* **3** 376–384.

Reinelt, G. 1994. *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag, Berlin, Germany.

Reinelt, G. 1995. TSPLIB 95. Research Report, Institut für Angewandte Mathematik, Universität Heidelberg, Heidelberg, Germany. Updated results available at http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

Rohe, A. 1997. *Parallele Heuristiken für sehr grosse Traveling Salesman Probleme*. Diplomarbeit, Research Institute for Discrete Mathematics, Universität Bonn, Bonn, Germany.

Schäfer, M. 1994. *Effiziente Algorithmen für sehr grosse Traveling Salesman Probleme*. Diplomarbeit, Research Institute for Discrete Mathematics, Universität Bonn, Bonn, Germany.

Verhoeven, M. G. A., E. H. L. Aarts, E. van de Sluis, R. J. M. Vaessens. 1995. Parallel local search and the travelling salesman problem. R. Männer, B. Manderick, eds. *Parallel Problem Solving from Nature 2*. North-Holland, Amsterdam, The Netherlands, 543–552.

Accepted by Jan Karel Lenstra; received August 2000; revised December 2001; accepted February 2002.