Chained Lin-Kernighan for large traveling salesman problems

David Applegate
AT&T Labs – Research

William Cook*
Computational and Applied Mathematics
Rice University

André Rohe
Forschungsinstitut für Diskrete Mathematik
Universität Bonn

July 27, 2000

Abstract

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 be found in under 1 CPU minute on a 300 Mhz Pentium II workstation, and solutions within 0.5% of optimal can be found in under 10 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 can be obtained in under 1 CPU day on a 64-bit IBM RS6000 workstation.

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. 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, Lenstra, Rinnooy Kan, and Shmoys (1985), Reinelt (1994), Jünger, Reinelt, and Rinaldi (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 lessor 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),

*Supported by ONR Grant N00014-98-1-0014.

Verhoeven, Aarts, van de Sluis, and Vaessens (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 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, Otto, and Felten (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 Martin, Otto, and Felten's algorithm 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, Reinelt, and Rinaldi (1995), Codenotti, Manzini, Margara, and Resta (1996), Johnson and McGeoch (1997), Hong, Kahng, and Moon (1997), and Neto (1999). These papers offer very good implementations for problem instances ranging in size from several hundred cities up to several thousand cities.

In this note, we discuss a number of issues involved in obtaining efficient implementations of Chained Lin-Kernighan for instances having 10,000 or more cities, including 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 is available for research purposes as part of the TSP code of Applegate, Bixby, Chvátal, and Cook (1998, 1999). A version of the code (written in the C programming language) can be be obtained over the internet at the page:

`http://www.keck.caam.rice.edu/tsp/`

1 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 $(i_0, \ldots, i_{n-1})$. 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 $(i_p, \ldots, i_q)$, that is, by moving to the tour

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

The well-known *2-opt* algorithm repeatedly searchs 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, Bixby, Chvátal, and Cook (1999).

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, Otto, and Felten (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

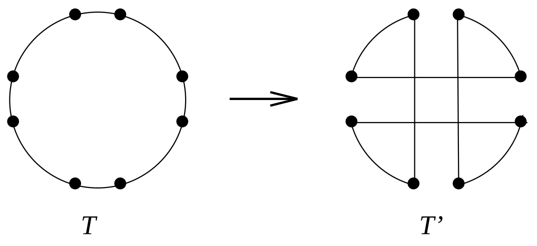

Figure 1: A double-bridge

discussed in the original Lin and Kernighan (1973) paper in a different context. It serves Martin, Otto, and Felten's purposes 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, Otto, and Felten 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 a great 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.

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 we are aware of are reported in Table 1. Each of the lower bound values given in the table was obtained with the TSP

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	645255	0.009%
pla33810	33,810	66005185	66059941	0.083%
pla85900	85,900	142307500	142409553	0.072%
r100000	100,000	225736239	225929775	0.086%

code of Applegate, Bixby, Chvátal, and Cook (1998, 1999); the tour values were obtained with the branch-width code of Cook and Seymour (1993).

2 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 (1992), 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 (1992); 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 5 choices for the first flip in the sequence, and for each of these flips they explore up to 5 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 $(b_1, b_2, \ldots, b_p)$ to refer to a bound of b_k choices for flips at each of the $k = 1, \ldots, p$ levels, with no backtracking permitted beyond the pth level. Implementations of Chained Lin-Kernighan must tradeoff 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 5 trials over each of the 6 instances in our test suite. For each of the 30 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,

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

The average of the 30 values, calculated at each of three points in the runs, is reported in Table 2; the "Short Run" entry is the value after 1 minute for the 3 smaller instances (rl11849, usa13509, d18512) and after 3 minutes for the 3 larger instances (pla33810, pla85900, and r100000), the "Medium Run" entry is the value after 5 minutes for the smaller instances and after 15 minutes for the larger instances, and the "Long Run" entry is the value after 30 minutes for the smaller instances and after 90 minutes for the larger instances.

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 complier, using the –O3 option. In these tests we use the 2-level list tour data structure described in Chrobak, Szymacha, and Krawczyk (1990) and in Fredman, Johnson, McGeoch, and Ostheimer (1995).

Table 2: Level of Backtracking

Search	Short Run	Medium Run	Long Run
(2,1)	0.553%	0.359%	0.269%
(2,2)	0.501%	0.325%	0.238%
(5,5)	0.480%	0.286%	0.204%
(4,3,2)	0.463%	0.285%	0.201%
(4,3,3,2)	0.490%	0.280%	0.195%
(4,3,3,2,2)	0.505%	0.284%	0.203%
(4,4,3,3,2,2)	0.637%	0.304%	0.203%
(5,5,4,4,3,3,2,2)	1.343%	0.609%	0.294%

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.

3 Choice of the kick

In their computations, Martin, Otto, and Felten (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 later 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 Martin, Otto, and Felten's process of examining kicks at random. 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 of the kick, we consider a small fraction of the cities 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 β, a random sample of βn 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 6 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 β, 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 efficiently examine the nearest sets. 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 3 least costly edges in each of the four geometric quadrants (for 2-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 2-matching algorithms, and Johnson and McGeoch (1997) have shown that it is an effective neighbor graph for Chained Lin-Kernighan.

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 described in the previous section.) The results indicate a clear preference for the cost-restricted kicks. Notice that for each of our local kicks, the wider variants are preferable for long runs, while the intermediate-width kicks are preferable for short and medium runs.

Our reported results are restricted to double-bridge kicks, but there is no strong argument favoring these over other kicking structures. Hong, Kahng, and Moon (1997) tested the use of k-exchange kicks for k varying from 2 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, Manzini, Margara, and Resta (1996), involving a perturbation of the x, y-coordinates of the cities. In our implementation, however, double-bridges appear

Table 3: Kicking Strategy

Kick	Short Run	Medium Run	Long Run
random	0.774%	0.423%	0.229%
random, long first edge	0.767%	0.407%	0.231%
close ($\beta = 0.003$)	0.535%	0.281%	0.173%
close ($\beta = 0.01$)	0.481%	0.273%	0.183%
close ($\beta = 0.03$)	0.519%	0.325%	0.223%
geometric ($k = 100$)	0.521%	0.329%	0.237%
geometric ($k = 250$)	0.490%	0.280%	0.195%
geometric ($k = 1000$)	0.549%	0.287%	0.171%
random-walk (25 steps)	0.584%	0.435%	0.358%
random-walk (50 steps)	0.483%	0.258%	0.170%
random-walk (100 steps)	0.544%	0.276%	0.168%

to perform at least as well as any alternatives that we have tried; we will comment further on the Codenotti, Manzini, Margara, and Resta scheme in Section 5 below.

4 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 greatly 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 6 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, Manzini, Margara, and Resta (1996). The remaining two tours, "Quick-Borůvka" and "HK-Christofides", are described below.

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 2 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 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.

Table 4: Initial Tour

Tour	Short Run	Medium Run	Long Run
Random	0.613%	0.307%	0.177%
Nearest Neighbor	0.575%	0.309%	0.180%
Christofides	0.487%	0.267%	0.167%
Greedy	0.547%	0.284%	0.169%
Quick-Borůvka	0.483%	0.258%	0.170%
HK-Christofides	0.431%	0.230%	0.147%

The results reported in Table 4 demonstrate that the initial tour can indeed have an impact on the performance of Chained Lin-Kernighan. 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 will further evaluate Quick-Borůvka versus HK-Christofides in Section 5.

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.

5 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.

TSPLIB instances

We now consider the full set of 7 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	469393	0.004%
d15112	15,112	1573037	1573095	0.004%

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 8 instances we made 5 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.

Table 5: Excess over Lower Bounds

Name	1 Minute	10 Minutes	1 Hour	4 Hours	24 Hours
rl11849	0.510%	0.288%	0.238%	0.218%	0.190%
usa13509	0.448%	0.223%	0.147%	0.128%	0.092%
brd14051	0.487%	0.164%	0.111%	0.091%	0.069%
d15112	0.394%	0.169%	0.110%	0.072%	0.067%
d18512	0.423%	0.180%	0.123%	0.094%	0.069%
pla33810	0.732%	0.407%	0.296%	0.260%	0.227%
pla85900	0.847%	0.339%	0.247%	0.209%	0.161%
r100000	1.630%	0.480%	0.276%	0.217%	0.172%

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 graph. The average start-up time (in seconds) for the 8 test instances are

rl11849	usa13509	brd14051	d15112	d18512	pla33810	pla85900	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 impact the "1 Minute" times. Improved 1-minute results

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	810,559
usa13509	1,968	21,818	118,745	473,399	571,987
brd14051	3,228	32,520	185,799	742,199	901,671
d15112	2,701	28,837	162,199	647,999	794,213
d18512	3,454	36,192	200,271	797,399	955,276
pla33810	3,205	37,393	193,862	770,399	913,239
pla85900	3,020	38,920	203,828	757,799	858,647
r100000	806	18,660	124,176	444,607	518,802

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.

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 10 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 4 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 6 nearest neighbors to each city as our neighbor graph, rather than the usual 3-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, Chvátal, and Cook (1990) and in Fredman, Johnson, McGeoch, and Ostheimer (1995); this data structure has better asymptotic behavior than the 2-level list implementation.

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, Halton, and Hammersley (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).) Johnson, McGeoch, and Rothberg (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

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

were also obtained by Percus and Martin (1996) and by Cerf, Boutet de Monvel, Bohigas, Martin, and Percus (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 8 CPU days) is approximately 0.3% above optimal.

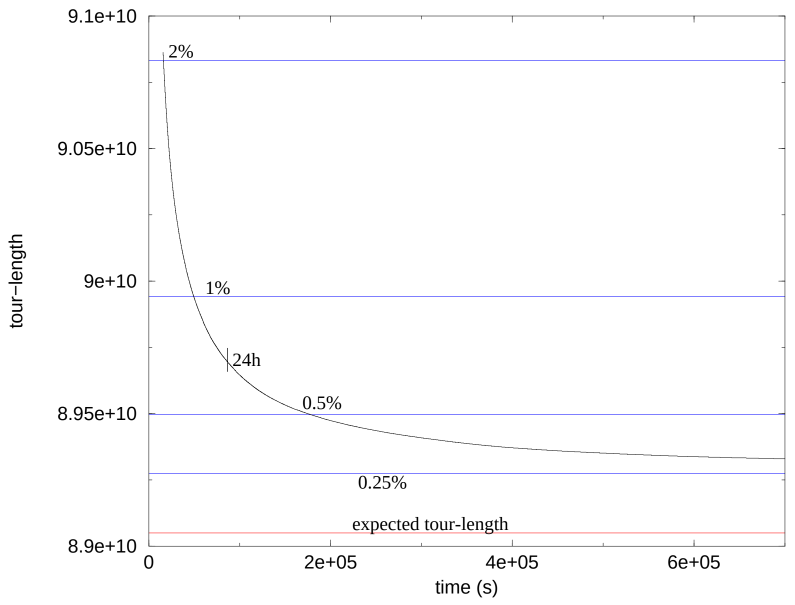

Figure 2: Run on 25,000,000-City Instance

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.

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, Manzini, Margara, and Resta (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 3, Codenotti, Manzini, Margara, and Resta (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, McGeoch, and Rothberg (1996), Codenotti, Manzini, Margara, and Resta 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 as 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, Manzini, Margara, and Resta obtained tours with average Held-Karp excess of 1.65% (in the best of 3 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 5 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, Manzini, Margara, and Resta heuristic may be due to both the global nature of their kicks, and their more aggressive use of Lin-Kernighan (they permit the algorithm to work very long in each iteration).

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, Manzini, Margara, and Resta (1996)
1.31%	298.4	7.8x	Johnson and McGeoch (1997)
1.08%	606.9	11.5x	Johnson and McGeoch (1997)

The excellent survey paper of Johnson and McGeoch (1997) contains a section on a "Production-mode Iterated Lin-Kernighan" that includes results 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 5 test instances. The approximate speed-up of a 11.5x over their very good code provides further evidence of the effectiveness of local kicks versus random kicks on large test instances.

Using 1,000 minutes

In the preceding sections, we took "Long Run" to mean 30 minutes of CPU time for the smaller instances and 90 minutes for the larger instances. Although this is indeed a large amount of computing time, in many cases (particularly given the rapid increases in available computing power) it may be feasible to devote much more CPU time to the tour-finding routine. One of the strengths of Chained Lin-Kernighan is that it can effectively use additional computing power to greatly improve the quality of the tours that are produced. This feature distinguishs it from many other TSP heuristics that have been proposed thus far in the literature, but it raises the issue of how to best use Chained Lin-Kernighan when very large amounts of CPU time are available. To discuss this, we consider the case when 1,000 minutes can be devoted to the tour-finding procedure.

We begin by considering again the HK-Christofides tour. A 1,000-minute limit gives ample time to run the Held-Karp procedure, even if we are not interesting in the lower bound it provides. To compare HK-Christofides with Quick-Borůvka, we made 5 trials on each of the 3 test instances usa13509, pla85900, and r100000. Each trial consists of a 1,000-minute run on our 300 MHz Intel Pentium-II workstation. In our tests, we use the 100-step random walk kick, since this choice gives the best results at the "Long Run" checkpoints reported in Table 3. The percentage excess of the cost of the computed tours over the cost of the best known tours (as reported in Table 1) is

Quick-Borůvka	HK-Christofides
0.076%	0.090%

averaged over the 15 trials. On average Quick-Borůvka performs better than HK-Christofides, with Quick-Borůvka behaving much better on the structured instance (pla85900), and HK-Christofides slightly better on the two uniform-like instances (usa13509 and r100000).

In our implementation of Chained Lin-Kernighan, we adopt the strategy of Johnson (1990) and simply discard any tour produced by Lin-Kernighan that is more costly than the current best tour produced during the run of the iterated algorithm. Martin, Otto, and Felten (1991, 1992) actually propose a more general, simulated-annealing-like approach that works with the new Lin-Kernighan tour with a certain probability that depends on the difference in the costs between it and the old Lin-Kernighan tour. In very long runs (several days of CPU time) we found that this idea was sometimes useful in improving tours that had caused our usual implementation to stall. In our 1,000-minute tests, however, we were not able to find a choice for the "annealing-schedule" that resulted in tours of significantly better quality than with our standard implementation.

Another possibility for improving the 1,000-minute performance is to reconsider the selection of the width of the kicks. The results reported in Table 3 indicate that wider kicks perform relatively better as the time limit is increased. In an attempt to take advantage of this, we tried a variation where we increased the width of the kicks as the search continued (using 50-step random-walk kicks for the first $n/10$ iterations, switching to 100-step kicks for up to the nth iteration, then switching to close kicks with decreasing values of β), but we were not able to obtain better performance over the group of test instances.

Finally, given the random nature of the algorithm (and the stalling that begins to appear towards the end of runs), it is not clear that the best use of 1,000 minutes is to perform a single run of Chained Lin-Kernighan. An alternative is to make several independent runs of the algorithm, and select the best tour from those that are produced. We tested this idea, obtaining the average tour qualities

2×500 Minutes	4×250 Minutes	10×100 Minutes
0.103%	0.122%	0.149%

over the three test instances (usa13509, pla85900, and r100000). For this time range, it appears to be better to simply use one long run of the algorithm, but it is certainly the case that for some (much longer) time limits the use of multiple runs should be considered.

6 Conclusions

The reported computational results demonstrate that Chained Lin-Kernighan is suitable for use on even very large test instances. These results support the findings of Johnson and McGeoch (1997), indicating that variants of Chained Lin-Kernighan offer very stiff competition for other TSP heuristics over a wide range of available computing times.

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.

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*, pp. 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.

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

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.

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.

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", Bellcore Technical Memorandum, December.

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) available via anonymous ftp from `netlib.att.com`.

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.

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", in *Proceedings 17th Colloquium of Automata, Languages, and Programming*, Lecture Notes in Computer Science, Volume 443, Springer-Verlag, Berlin, pp. 446–461.

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

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

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

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

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

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

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.

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

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.

Reinelt, G. 1995. "TSPLIB 95", Research Report, Institut für Angewandte Mathematik, Universität Heidelberg.

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

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

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