Local Optimization and the Traveling Salesman Problem

David S. Johnson
AT&T Bell Laboratories
Room 2D-150
Murray Hill, NJ 07974, USA

Abstract

The Traveling Salesman Problem (TSP) is often cited as the prototypical "hard" combinatorial optimization problem. As such, it would seem to be an ideal candidate for nonstandard algorithmic approaches, such as simulated annealing, and, more recently, genetic algorithms. Both of these approaches can be viewed as variants on the traditional technique called local optimization. This paper surveys the state of the art with respect to the TSP, with emphasis on the performance of traditional local optimization algorithms and their new competitors, and on what insights complexity theory does, or does not, provide.

1. Introduction

In the Traveling Salesman Problem, or "TSP," we are given a sequence of *cities* $c_1, c_2, \ldots, c_N$ and for each pair c_i, c_j of distinct cities a *distance* $d(c_i, c_j)$. Our goal is to find a permutation π of the cities that minimizes the quantity

$$\sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \quad + \quad d(c_{\pi(N)}, c_{\pi(1)}).$$

This quantity is referred to as the *tour length*, since it is the length of the tour a salesman would make if he visited the cities in the order specified by the permutation, returning at the end to the city from which he started. In the *symmetric* TSP, on which we shall concentrate in this paper, the distances satisfy $d(c_i, c_j) = d(c_j, c_i)$ for $1 \leq i, j \leq N$.

The symmetric traveling salesman problem has many applications, from VLSI chip fabrication to X-ray crystallography, and a long history (for which see [31], the definitive source on the problem). Its simplicity, its many applications, and no doubt its distinctive name have all helped to establish the TSP as a proving ground of choice for new algorithmic ideas. Important early work on Lagrangean relaxation, branch and bound algorithms, cutting plane techniques, and local optimization all took place within this setting. In addition, the TSP was one of the first problems to which the new theory of NP-completeness was applied in the early 1970's, and has since then been regularly used as the prototypical example of an NP-hard combinatorial optimization problem.

Given that the problem is NP-hard, and hence polynomial-time algorithms for finding

optimal tours are unlikely to exist, much attention has been addressed to the question of efficient *approximation algorithms*, fast algorithms that attempt only to find *near*-optimal tours. To date, the best such algorithms in practice have been based on (or derived from) a general technique known as *local optimization*, in which a given solution is iteratively improved by making local changes. This paper will survey results, both theoretical and empirical, about this technique as it applies to the TSP. We shall be particularly interested in evaluating recent variants on local optimization, such as simulated annealing and genetic algorithms. For brevity's sake, we will concentrate on the question of how well the various approaches perform, rather than on the precise details of the best current implementations. Note, however, that the quoted performances often depend crucially on those details, and for a full picture the reader should consult the cited references.

We begin with two sections intended to put local optimization algorithms in perspective. In Section 2 we discuss the current state of the art for algorithms designed to find optimal solutions and verify their optimality. In Section 3 we discuss *tour construction heuristics* that simply build a tour and stop. These are the heuristics about which most is known theoretically, and their empirical performance can serve as a benchmark.

Section 4 covers the most successful of the standard local optimization algorithms for the TSP, the 2-Opt, 3-Opt, and Lin-Kernighan algorithms. We also discuss what the new theory of "PLS-completeness" has to say about the running times of these algorithms. Sections 5 and 6 cover simulated annealing and genetic algorithms, respectively. The latter also discusses a highly successful new "nested local optimization" algorithm that, in this context, might be called "parthenogenetic." Section 7 concludes with a brief discussion of further alternative approaches to the TSP, as well as directions for further research.

Many of the empirical results reported here are the product of joint work involving the author, Jon Bentley, Lyle McGeoch, and Ed Rothberg (in various combinations). More detailed reports are under preparation and will appear elsewhere [2,3,19,20,22].

2. Finding Optimal Tours

Not only is the TSP hard, it remains hard under a wide variety of restrictions [11,31], so that, assuming $P \neq NP$, there are very few practical domains in which one can hope to have optimization algorithms that run in worst-case polynomial time. Nevertheless, it now appears that "in practice" much larger problems can be solved optimally than one might have expected, calling into question the TSP's reputation as the "prototypically hard" combinatorial optimization problem.

Using branch-and-bound techniques together with sophisticated new algorithms for generating cutting planes, two groups of researchers have reported remarkable successes. Padberg and Rinaldi [40,41] report regularly solving 100-city instances in 15-30 minutes of VAX-780 time, with no branching required, and have even solved the famous 318-city problem of [33] in time 3.4 hours without branching. On a 10 times faster Cyber-205 computer, and *with* branching, they have solved real-world instances of size 532 (roughly 6 hours), 1002 (7.3 hours), and even 2392 (27.3 hours). On an IBM 3090/600, with better linear programming code and an improved algorithm, the time for the 2392-city instance is reduced to 2.6 hours, although the 532-city instance still takes 5.5 hours, indicating that the mere size of a problem is not the determining factor for running time [41].

Independently, Grötschel and Holland [13] have solved a 666-city instance based on

the actual locations of major cities throughout the world in roughly 9 hours on an IBM 3081D, as well as a 1000-city random distance matrix instance in just 0.5 hours. This latter type of instance (inter-city distances are uncorrelated random numbers chosen uniformly in the interval (0,1]) seems to be particularly easy on average. Stone [49] has implemented a relatively simple branch-and-bound scheme whose running time empirically seems only to be roughly quadratic in N, although since the space required is also quadratic, he has not been able to run it on instances of size exceeding 800.

These successes by no means imply that *all* instances in this size range will be as easy (and note that "easy" here is a relative term, given that the better times all involved large and expensive computers and large and sophisticated programs). Nevertheless, it does appear that instances as small as 100 cities must now be considered to be well within the state of the global optimization art, and instances must be considerably larger than this for us to be sure that heuristic approaches are really called for.

3. Tour Construction Heuristics

Fortunately for heuristic designers, there are many applications where such "larger" instances abound. Circuit board drilling applications with up to 17,000 cities are mentioned in [34], X-ray crystallography instances with up to 14,000 cities are mentioned in [4], and instances arising in VLSI fabrication have been reported with as many as 1.2 million cities [28]. Moreover, 5 hours on a multi-million dollar computer for an optimal solution may not be cost-effective if one can get within a few percent in seconds on a PC. Thus there remains a need for heuristics.

The TSP heuristics currently most often covered in the computer science curriculum are what we might call *tour construction heuristics*, as they gradually build up a tour out of shorter paths or cycles. Here are several of the most popular.

Nearest Neighbor. Starting from an arbitrarily chosen initial city, repeatedly choose for the next city the unvisited city closest to the current one. Once all cities have been chosen, close the tour by returning to the initial city.

Greedy Algorithm. Go through the city pairs in order by non-decreasing distance, adding the corresponding edge to the tour whenever doing so will neither create a vertex of degree exceeding two nor a cycle with less than N cities.

Nearest Insertion. Starting from a degenerate tour consisting of the two closest cities, repeatedly choose the non-tour city with the minimum distance to its nearest neighbor among the tour cities, and insert it in between the two consecutive tour cities for which such an insertion causes the minimum increase in total tour length.

Farthest Insertion. Starting from a degenerate tour consisting of the two cities with maximum inter-city distance, repeatedly choose the non-tour city with the *maximum* distance to its nearest neighbor among the tour cities, and insert it as in Nearest Insertion.

Double Minimum Spanning Tree (Double MST). Construct a graph consisting of two copies of a minimum spanning tree for the cities. This graph must have an Euler cycle; construct one. Derive a tour by traversing the cycle and taking the cities in the order in which they are first encountered along it.

Christofides. Construct a graph consisting of a copy of a minimum spanning tree and a minimum length matching of the odd degree vertices in that tree. Then proceed as

in the Double MST algorithm. See [8].

Strip. (This and the following algorithms assume that the cities correspond to points in the plane under some standard metric. For simplicity, we also assume that the points are confined to the unit square.) Divide the unit square into $\sqrt{N}$ equal-width strips, and then starting at the leftmost strip, proceed strip by strip, visiting the cities in order of height within each strip, alternatingly descending and ascending. Connect the last city in the rightmost strip to the first city in the leftmost strip.

Spacefilling Curve. Visit the cities in the order that they occur along a spacefilling curve for the unit square; for details, see [45].

Karp's Partitioning Algorithm. Recursively partition the cities by horizontal and vertical cuts through median cities until no more than B cities are in any set of the partition (B a parameter). Using dynamic programming, optimally solve the subproblems generated by the sets of cities in the partition, patching the solutions together by means of the shared medians between sets. For details, see [23].

Litke's Recursive Clustering Algorithm. Recursively replace clusters of size B by single representative cities until less than B remain. Solve this small problem optimally, then expand the clusters one by one, optimally sequencing the expanded set between the two neighbors of their representative in the current tour. For details, see [34].

The running times for Nearest Neighbor, Nearest Insertion, Farthest Insertion, and Double MST are all $O(N^2)$, Greedy runs in time $O(N^2 \log N)$, and Christofides runs in time $O(N^3)$. The last four algorithms are designed to take advantage of the geometry to which they are restricted, and are consequently substantially faster: Strip and Spacefilling Curve run in time $O(N \log N)$, and running times for Karp's Partitioning algorithm and Litke's Recursive Clustering Algorithm are $O(N)$ for any fixed B. The more general algorithms (with the exception of Christofides) can also be implemented to take advantage of geometric structure, however (using *k-d* trees and more sophisticated data structures [1,2,3]), so that all run empirically in time $O(N \log N)$ or perhaps $O(N \log^2 N)$ for such instances. If one settles for a greedy rather than a minimum matching, one can also obtain an approximation to Christofides with this type of running time.

Table 1 illustrates the actual running times for efficient implementations of these algorithms on instances consisting of random points within the unit square under the Euclidean metric, for N increasing by powers of $\sqrt{10}$ from 100 to 100,000, on a single processor of a Sequent Balance computer. Such processors are relatively slow in modern terms, having roughly the same speed as a VAX-750 without a floating point accelerator. (For nominal running times on a SUN-4 or VAX-8550, divide by 10; for time on a MIPS processor, divide by 40.) Results in Table 1 are averages over 10, 5, 5, 5, 3, 1, 1 instances of each size, but were reasonably consistent from run to run, and indicative of times encountered for more "real-world" geometric instances (see [2,3]).

Let us turn now to the quality of the tours that these heuristics construct. Much is known of a theoretical nature. For instances not obeying the triangle inequality, no polynomial time heuristic can guarantee that the lengths of the tours it constructs will be bounded by a constant times the optimal tour length [47]. Indeed, for random distance matrix instances, Nearest Neighbor's *expected* tour length is $\Theta(\log N)$ times the optimal length (and the other heuristics appear to fare no better).

Assuming the triangle inequality holds (as it does for Euclidean instances), the

	NUMBER OF CITIES (POINTS UNIFORM IN THE UNIT SQUARE)						
ALGORITHM	10^2	$10^{2.5}$	10^3	$10^{3.5}$	10^4	$10^{4.5}$	10^5
Spacefill	0.2	0.8	2.7	8.7	28.4	92.3	302.3
Strip	0.3	1.0	3.5	11.9	39.7	133.6	445.7
Nearest Neighb.	0.6	2.2	9.8	27.1	110.2	316.8	1031.2
Karp	1.5	2.7	42.9	51.4	94.9	1332.6	1621.0
Greedy	1.8	6.2	24.4	78.6	252.8	933.3	2930.3
Cluster	2.0	7.0	24.3	80.2	259.6	872.9	2851.4
Double MST	2.4	8.8	32.1	105.7	356.8	1772.0*	9168.0*
Approx.Christo.	3.1	11.5	45.8	139.6	483.4	1702.6	8151.0*
Nearest Insert	3.4*	12.2*	44.3*	163.8*	543.6*	2177.5*	10869.3*
Farthest Insert	3.4*	13.8*	56.0*	229.6*	805.5*	3240.8*	14925.3*
Christofides	7.1	151.6	4053.1	108000.0*			

Table 1. Running times in seconds on a Sequent Balance Computer for Tour Construction Heuristics. Asterisks indicate times on a VAX-8550 computer, multiplied by 10 so as to be roughly comparable.

algorithms perform much better. Now Nearest Neighbor (and Greedy) are in the *worst case* $\Theta(\log N)$ times the optimal [46], Double MST and Nearest Insertion are at most *twice* optimal [46], and Christofides is at most 1.5 times optimal [8]. For points random in the unit square, the *expected* ratios to optimal for Strip and Spacefilling Curve are bounded, with the former slightly better and both in the vicinity of 1.3 [45]. As to Karp's heuristic, for any ε there is a B such the heuristic's expected tour length is no more than $1+\varepsilon$ times optimal (although unfortunately the running time is exponential in $1/\varepsilon$).

	NUMBER OF CITIES (POINTS UNIFORM IN THE UNIT SQUARE)						
ALGORITHM	10^2	$10^{2.5}$	10^3	$10^{3.5}$	10^4	$10^{4.5}$	10^5
Karp	49.5	61.2	56.1	67.1	76.0	56.8	64.0
Double MST	36.9	37.7	39.0	39.3	39.5	39.7	39.9
Spacefill	25.4	28.0	31.2	33.4	34.2	34.8	34.9
Strip	22.8	30.4	30.3	30.7	30.3	30.5	30.2
Nearest Insert.	23.0	24.4	26.5	26.4	26.5	27.4	27.1
Nearest Neighb.	29.3	26.7	25.1	25.7	24.2	24.2	23.8
Cluster	19.3	20.4	23.9	21.0	22.8	22.9	23.2
Approx.Christo.	11.6	16.3	19.0	18.5	19.1	19.3	19.4
Greedy	22.1	18.6	16.9	16.8	16.5	15.5	14.7
Farthest Insert.	8.3	10.4	12.5	12.6	13.3	13.2	13.6
Christofides	8.8	8.9	9.9	9.8			

Table 2. Average percentage excess over the Held-Karp lower bound on optimal tour length for Tour Construction Heuristics

Table 2 indicates how the heuristics behaved in practice for the instances of Table 1. Entries give the average percentage excess above the Held-Karp lower bound on the optimal tour length [14,15], the latter being computed using the techniques described in

[22]. For instances of this type, the Held-Karp lower bound appears to be within 0.8% of the true optimum [19,22]. Note the lack of a general correlation between speed and quality of solution, or between worst-case guarantees and average performance. Also note that relative performance at $N = 100$ is not always a good indicator of performance as N gets large. For each heuristic, the percentage excess appears to be approaching a limit which is in many cases quite different from the value at $N = 100$. The performance of Karp's algorithm is less consistent, depending as it does on how well the bound B divides the number of cities N. Our experiments set $B = 10$; the percentage excess for $N = 100$ could have been improved to roughly 33% by increasing B to 14, but this would have increased the running time to 130 seconds, i.e. 18 times slower than Christofides.

Experiments reported in [19] indicate that the results here for larger values of N are roughly indicative of how well the algorithms perform on large real-world geometric instances. The one exception is the Strip algorithm, which typically performs much more poorly on real-world instances than on uniform random ones.

4. Local Optimization

If one is not satisfied with a tour constructed by one of the above heuristics, one may attempt to improve it by local perturbations. A fundamental concept here is that of a *neighborhood structure*, a relation (not necessarily symmetric) which contains the pair of tours (T,T') if and only if T' is a legitimate perturbation of T (in which case we call T' a *neighbor* of T). For instance, in the (symmetric) "2-swap" neighborhood structure, two tours are judged neighbors if and only if they differ in the positions of precisely two cities.

A local optimization algorithm based on such a structure consists of two subroutines: (A) an algorithm that, given an instance I, constructs an initial tour T_0, and (B) an algorithm that, given I and any tour T, will determine if there is neighboring tour of better cost, and if so will return one such tour T'. If no such tour exists, T is called "locally optimal." (Note that a locally optimal solution need not be globally optimal, or even *close* to globally optimal. The hope, however, is that it will tend to be fairly good.) These subroutines may be either randomized or deterministic. A local optimization algorithm uses these subroutines as illustrated in Figure 1.

1. Call subroutine A on input I and obtain an initial solution $T = T_0$.
2. Do the following until T is declared locally optimal:
    2.1. Call subroutine B on inputs I and T.
    2.2. If B returns a better solution T', set $T = T'$.
3. Return T.

Figure 1. Local Optimization for the TSP

The most famous local optimization algorithms for the TSP are the "2-Opt," "3-Opt," and "Lin-Kernighan" algorithms [32,33]. (The simple 2-Swap neighborhood structure mentioned above turns out not to be very effective.) All three have been in heavy use for many years, with Lin-Kernighan often spoken of as the "champion" among TSP heuristics. The corresponding neighborhood structures are as follows:

2-Opt. Two tours are neighbors if one can be obtained from the other by deleting two edges, reversing one of the resulting two paths, and reconnecting.

3-Opt. Two tours are neighbors if one can be obtained from the other by deleting *three* edges and putting the three resulting paths together in a new way, possibly reversing one or more of them.

Lin-Kernighan. Here instead of simply going one step further to "4-Opt," one goes to a much more complex, asymmetric neighborhood structure (and one that is too complicated to describe here in detail). In summary, for tour T' to be a neighbor of tour T, it must be shorter and there must be a way to obtain it by breaking three edges of T, rearranging, and then performing a series of additional 2-opt moves constructed by a greedy, bounded-width search. See [33] for details.

Naive worst-case running time bounds for the corresponding subroutine B's are $O(N^2)$, $O(N^3)$, and $O(N^5)$, although fortunately there are combinatorial observations, special-purpose data structures, and algorithmic modifications that can speed all three up substantially in practice (for details, see [2,3]). Unfortunately, even if one knows the running time of subroutines A and B, one cannot bound the running time of the overall algorithm unless one can bound the number of times the loop involving statements 2.1 and 2.2 is performed. For the three algorithms in question here, no nontrivial bounds are known.

Indeed, for the case of Lin-Kernighan, no polynomial bound on iterations exists. This follows from the fact, recently proved in [42], that the Lin-Kernighan neighborhood structure is complete for the class PLS of all polynomial-time local search neighborhood structures (for *all* combinatorial optimization problems), as defined in [21]. For a "PLS-complete" neighborhood structure, local optima can be found in polynomial time only if they can be found that quickly for *all* neighborhood structures in PLS. This latter hypothesis is currently deemed unlikely (although perhaps not as unlikely as P = NP). Consequently, finding local optima for Lin-Kernighan neighborhood structure is presumed to be difficult, at least in the worst case. Note that this observation applies to all methods for finding local optima, not simply the iterative local optimization procedure of Figure 1. For that approach, something stronger can be said as a corollary of the type of reduction used in [42] to prove PLS-completeness: There exist starting tours for which the algorithm is *required* to take an exponential number of steps. It is not at present known whether 2-Opt and 3-Opt are also PLS-complete, although Krentel has recently shown that there exists a finite $K > 3$ such that K-Opt is PLS-complete, and Lueker [35] has constructed instances and starting tours that force 2-Opt to take exponential time.

Theory also tells us something about the quality of solutions that 2-Opt, 3-Opt, and Lin-Kernighan can guarantee. For general instances (without triangle inequality), they cannot guarantee tours that are bounded by any constant multiple of the optimal tour length (assuming $P \neq NP$), even if they *do* take an exponential number of steps. (No local optimization algorithm with polynomial-time subroutines A and B can [43].) Indeed, for any fixed K, instances can be constructed with arbitrarily bad tours that are locally optimal with respect to the K-Opt neighborhood structure [44]. If the triangle inequality holds, one can still construct instances with tours of essentially twice the optimal length that are locally optimal in this sense [46]. (Of course for such instances any local optimization algorithm will inherit the performance guarantee provided by its starting-tour subroutine, and so can avoid arbitrarily bad behavior.)

This being the worst-case theory, let us once again look at what happens in practice. We again first consider random Euclidean instances (the same ones used in Tables 1 and 2, restricted to the values of N that were multiples of 10). For such instances, the number of iterations for 2-Opt and 3-Opt is almost surely polynomial, although the proven bound is

still large [24]. In our implementations we used random starting tours for Lin-Kernighan, and Nearest Neighbor starting tours for 2-Opt and 3-Opt, with the choice of starting city randomized. (Purely random starting tours would greatly increase the running times for 2-Opt and 3-Opt, and lead to poorer final solutions, although they have no such adverse effect on Lin-Kernighan.) Note that with these choices of starting tours we do not "inherit" any constant-multiple performance guarantees for Euclidean instances.

	RUNNING TIME IN SECONDS				PERCENTAGE EXCESS			
N:	10^2	10^3	10^4	10^5	10^2	10^3	10^4	10^5
Greedy	0.2	1.6	18.8	225	22.1	16.9	16.5	14.7
F.Insert	0.3	5.6	80.6	1493	8.3	12.5	13.3	13.6
Christo.	2.8	992.9			8.8	9.9		
2-Opt	0.4	5.0	55.0	746	6.2	6.4	6.4	6.5
3-Opt	0.5	6.6	71.5	980	2.6	3.5	3.6	3.6
Lin-Kern	1.1	16.7	340.2	17776	1.5	2.1	2.2	2.2

Table 3. Running times on a VAX-8550 and percentage excess over the Held-Karp lower bound for Local Optimization algorithms and selected Tour Construction heuristics

Table 3 summarizes the running times and percentage excesses over the Held-Karp lower bounds for our implementations of the three local optimization algorithms. These experiments were performed on a VAX-8550 (roughly 10 times faster than the Sequent processor used for Table 1), and we repeated the experiments for the 3 best tour construction heuristics on this machine, so as to facilitate comparisons. Since the local optimization algorithms all use randomization to generate their starting tours, the results for them were averaged (over 5 runs for each of the 100 and 1,000 city instances, 3 runs each for the 10,000 and 100,000 city instances). The variances were not substantial.

Note that in this range, 2-Opt and 3-Opt have only slightly worse than linear running time growth rates. Bentley [1] has observed by detailed profiling, however, that there is a hidden $\Omega(N^{1.74})$ component to the running time due to the growth rate of the length of the path segments that need to be reversed. This term is just beginning to become significant at $N = 100{,}000$, but is dominant by $N = 1{,}000{,}000$, where 2-Opt and 3-Opt take roughly 3 and 4 hours respectively. The number of calls to "subroutine B" tends to be around $N/4$ throughout this range, with 3-Opt typically making slightly more calls.

The Lin-Kernighan algorithm, which consistently produces results that are within 2.2% of the lower bound, is subquadratic in this range, and for $N \leq 10{,}000$ is not inordinately slower than its competitors. (At $N = 100{,}000$, its 5-hour running time is still quite feasible, although substantially slower than that for 2-Opt and 3-Opt.) Its increased running time complexity is due to the fact that its subroutine B is considerably more time-consuming than those for 2-Opt and 3-Opt; the *number* of calls to the subroutine remains roughly $N/4$ and normally is *less* than the number for 3-Opt.

It should be pointed out that one of the "algorithmic modifications" embodied in our implementations makes it possible that we may occasionally terminate with tours that are not locally optimal. Turning this feature off, however, does not significantly increase the number of subroutine calls (or improve the quality of solution). Leaving it on reduces running times by factors of from 2 to 5.

As with the Tour Construction heuristics, the behavior of 2-Opt, 3-Opt, and Lin-Kernighan on these Euclidean instances is a reasonable predictor for their behavior on real-world instances, including ones arising in local VLSI applications having up to 85,900 cities. The running time for Lin-Kernighan, however, can increase substantially (by factors of 5 or more) when the wrong kind of clustering is present in the instance. (Again this is due to added time spent in individual calls to subroutine B rather than to a substantial increase in the number of such calls.) The quality of the Lin-Kernighan solutions does not seriously degrade for such instances, however. For example, on each of the "real-world" instances mentioned in Section 2 for which optimal solutions are known, Lin-Kernighan averages less than 1.7% above optimal.

A general conclusion would seem to be that Lin-Kernighan would be the algorithm of choice until its running time makes it infeasible. Practitioners, however, might well be willing to settle for the much faster results obtainable by 3-Opt, even when Lin-Kernighan would be feasible, given the substantially greater programming complexity of the latter. Indeed, there is a whole range of trade-offs, and in many applications, the results obtainable by a naive implementation of Nearest Neighbor might well suffice.

There is one class of instances, however, where only Lin-Kernighan seems to be able to construct reasonable tours. This is the random symmetric distance matrix (fortunately not a source of many "real-world" instances, but interesting nonetheless). In contrast to the successes reported for optimization algorithms with this type of instance, heuristics seem to have unlimited difficulties. The percentage excess for 2-Opt is already averaging roughly 47% by $N = 100$ and continues to grow, reaching 170% by $N = 10{,}000$. For 3-Opt the corresponding figures are better, but still horrible: 11% and 81%. Even Lin-Kernighan does not seem to be able to hold a constant ratio, although its increase is only from 1.5% to 5.8%. The possibility that the Held-Karp lower bound might itself be partially to blame here cannot be ruled out, but there are reasons to believe that it is an even better estimator of the optimal tour length for random instances of this type than for geometric ones; the expected values both for it and for the optimal tour length appear to be asymptotic to constants near 2.041 [22,29].

5. Simulated Annealing

If one is not satisfied with the tours constructed by 3-Opt or Lin-Kernighan, and has large amounts of computing time available to seek improvements, several options are open. One that has received much publicity in relation to the TSP is *Simulated Annealing*. This is the randomized variant on local optimization that is illustrated in Figure 2. Note that this procedure allows occasional moves that make the solution cost worse (i.e., moves that go "uphill," and thus may allow us to escape from local optima). The acceptability of uphill moves is affected by a control parameter t called the *temperature*.

Typically the temperature is gradually reduced from a high value, at which most uphill moves are accepted, to a low one at which few if any such moves are accepted. One standard scheme of theoretical interest is "logarithmic cooling," in which the temperature on the kth trial is set to $C/(\log k)$ for some fixed constant C. This cooling schedule will normally guarantee convergence to an optimal solution [37] (although the convergence time is likely to exceed the time needed to find an optimal solution by exhaustive search [47]). A more commonly used scheme is *geometric cooling* [18], in which the temperature is held steady for some prespecified constant L number of trials

1. Call a subroutine to obtain an initial solution $s = s_0$ and temperature $t = t_0$.
2. Do the following until it's time to quit:
    2.1. Choose a random neighbor s' of s.
    2.2. If $c(s') < c(s)$ set $s = s'$.
    2.3. Otherwise,
        2.3.1. Set $\Delta = c(s') - c(s)$.
        2.3.2. Choose random number r uniformly from $[0,1]$.
        2.3.3. If $r \leq e^{-\Delta/t}$, set $s = s'$.
    2.4. Call a subroutine to determine a new temperature t.
3. Return s (or best solution seen so far, if different).

Figure 2. Simulated Annealing

(passes through the 2.1-2.4 loop), and is then multiplied by some prespecified reduction factor, for instance 0.95. This provides no asymptotic guarantees, but seems to work well in practice. Even geometric cooling, however, leads to substantially greater running times (for acceptable results) than needed by the corresponding local optimization scheme. The hope is that the extra time will help buy better results.

Many researchers have implemented versions of simulated annealing for the TSP [6], beginning with the two papers [7,26] that first (and independently) suggested applying the technique to combinatorial optimization. Based on the results and comparisons reported for random geometric instances by Kirkpatrick [25] and Lam and Delosme [30], at least two of these implementations appear to have succeeded in finding better tours on average than does our Lin-Kernighan implementation. Both implementations are based on modifications of the 2-Opt neighborhood structure. Kirkpatrick expands the neighborhoods to include restricted 3-Opt moves (the smallest of the three subpaths in the 3-Opt move must have length 10 or less). Lam and Delosme use only 2-Opt moves, but adaptively truncate the neighborhoods to include only the better moves, thus reducing the number of neighbors of a tour from $\Omega(N^2)$ to a constant, which allows for much faster running times. Furthermore, whereas Kirkpatrick uses geometric cooling or some simple variant of it, Lam and Delosme use an adaptive cooling schedule designed to drop the temperature as rapidly as possible, consistent with a statistical notion of "equilibrium" at a given temperature. Running times at 400 cities for both approaches, as expected, appear to be substantially greater than those for our implementation of Lin-Kernighan, perhaps by a factor of 70 or more for [25], and of at least 20 for [30].

For the sake of direct comparisons, we have constructed our own Simulated Annealing implementation for the TSP [19]. We use an adaptively truncated version of the augmented 2-Opt neighborhood of Kirkpatrick [25], this time allowing 3-opt moves with shortest path lengths of up to 20. We use geometric cooling, starting with a Nearest Neighbor tour at a temperature that is roughly half the average length of an edge in that tour (higher starting temperatures took more time but yielded roughly the same average final tour length). In our experiments the cooling ratio was fixed and the overall annealing time was adjusted by changing the number of moves tried at each temperature, a typical value being $20N$ (for details, see [19]). As with the implementations of [25,30], our implementation is capable of beating Lin-Kernighan on average. Picking a reasonably-sized random

Euclidean instance as a test case, we found tours that averaged 0.4% shorter than did those for Lin-Kernighan on a 1000-city random geometric instance. The running time differential was again large, however, Annealing taking 200 times as long as Lin-Kernighan (runs that were merely 100 times as long did not suffice to obtain better tours than Lin-Kernighan).

The fact that Simulated Annealing can, given enough time, find better solutions than do 3-Opt and Lin-Kernighan, does not necessarily mean it is to be preferred. In particular, note that our implementations of 2-Opt, 3-Opt, and Lin-Kernighan all make use of randomized subroutine A's for constructing their initial tours. Thus different runs can give different results, and one would expect to obtain substantially better than average tours by performing a sequence of independent runs and taking the best solution found. Thus a more appropriate comparison would be Annealing versus an equivalent amount of time spent on multiple runs of Lin-Kernighan or 3-Opt.

Our implementations of the latter are designed with this in mind, since they start by constructing a nearest neighbor data structure that need only be built once and then can be used for free by all subsequent runs. The time to do this construction is in fact the dominant running time component in a single run of 2-Opt or 3-Opt. If one ignores the masking effect of this preprocessing time, the underlying times for 2-Opt, 3-Opt, and Lin-Kernighan differ more substantially than is suggested by Table 3. See Table 4 for a breakdown of the running times with preprocessing separated out as a separate item. (Ignore for now the final line labeled "Iterated LK," which will not be discussed until Section 6.)

	RUNNING TIME IN SECONDS			
N:	10^2	10^3	10^4	10^5
Preprocessing	0.3	4.3	45.6	483
2-Opt	0.1	0.7	9.4	263
3-Opt	0.2	2.3	25.9	497
Lin-Kern	0.8	12.4	294.6	17293
Iterated LK	0.5	5.0	100.6	

Table 4. Running times on a VAX-8550 with preprocessing time separated out

On the 1000-city instance just discussed, we performed 10,000 independent runs of Lin-Kernighan, 1000 of 3-Opt, and 100 of Annealing, so that we could estimate how well the multiple-run approach might be expected to perform. (Note that there were only 1000 different possible runs for 3-Opt, since in our implementation the only randomization used is Nearest Neighbor's choice of the first city when generating the starting tour. We stepped through the 1000 choices, rather than randomizing.) Results are summarized in Table 5, which lists the running time (on a Sequent processor rather than the faster VAX-8550 of Tables 3 and 4) and estimated quality of solution for increasing numbers of iterations of Annealing, 3-Opt, and Lin-Kernighan. Note that the best 3-Opt run was still worse than the average runs for Lin-Kernighan and Annealing. The best of 20 runs of Lin-Kernighan, however, is better than the average Annealing run at 1/13th of the time. The predicted best for 2000 Lin Kernighan runs was comparable to the predicted best of 50 Annealing runs at 1/6th the time. Annealing appears to be gaining, but the running time needed for it to catch up appears to be enormous (and, as we shall see in Section 6, there are better ways to perform multiple runs of Lin-Kernighan).

	3-OPT		LIN-KERNIGHAN		ANNEALING	
RUNS	EXCESS	MINUTES	EXCESS	MINUTES	EXCESS	MINUTES
1	3.69	1.6	2.18	4.0	1.71	791
2	3.46	2.1	2.02	6.9	1.56	1582
5	3.21	3.6	1.85	15.7	1.42	3952
10	3.07	6.1	1.75	30.3	1.33	7903
20	2.96	11.1	1.66	59.5	1.27	15806
50	2.84	26.1	1.56	147.1	1.22	39513
100	2.76	51.0	1.50	293.1		
200	2.67	100.9	1.43	585.1		
500	2.55	250.7	1.34	1461.0		
1000	2.43	500.3	1.29	2921.0		
2000	2.43	999.4	1.23	5831.0		
5000	2.43	2496.9	1.18	14600.0		

Table 5. For multiple runs of 3-Opt, Lin-Kernighan, and Simulated Annealing, the estimated percentage excess over the Held-Karp lower bound of the best tour found and the overall running times on a Sequent processor

The results depicted in Table 5 are typical for both our random and real-world geometric instances. The conclusion is that for such instances Annealing can only be competitive with multiple-run Lin-Kernighan if extremely large running times are allowed. For random distance matrix instances, Annealing, like 2-Opt and 3-Opt, is much worse: On a 400-city random distance matrix, where Lin-Kernighan averages 3% above the Held-Karp lower bound, our annealing implementation needed 1000 times as much time as a single Lin-Kernighan run just to get within 20% (the excess reported by Kirkpatrick and Toulouse in [27]).

6. Genetic Algorithms

The multiple-run experiments reported above involved *independent* runs. No use was made in later runs of the results of earlier runs. One might reasonably wonder whether this is the best way to proceed, or whether there might be some way to capitalize on those earlier results. The much-vaunted "genetic algorithm" offers one path to such a capitalization. One performs some fixed number of independent runs (possibly in parallel), and then derives new starting solutions based on a transfer of information between (a "mating" of) the solutions found. This process can then be repeated for many "generations" until progress stops being made, with the "population" at each mating step consisting of the newly constructed tours and a selection of the best ones from previous generations.

This idea was first proposed for the TSP by Brady [5], who mated two tours by looking for a pair of subpaths, one in each tour, that contain precisely the same set of cities. The longer of the two paths is then replaced in its tour by the shorter. The local optimization algorithm used was 2-Opt. The results (for a 64-city geometric instance) were unfortunately no better than performing multiple independent runs of 2-Opt for the same total time. (More total runs are possible under the latter approach, since no time need be spent searching for mating possibilities.)

Recently, a substantially more effective implementation of the genetic scheme has been devised by Mühlenbein et al. [38,39]. Here the mating strategy is the more powerful

one of taking a path in one tour (of length between 10 and $N/2$), and reordering the second tour so that it contains that path while maintaining all other cities in the same cyclic order. For instance, if the path *abcde* is inserted into the tour *faghiebjkdcl* in this fashion, the result would be *fabcdeghijkl*. Similarly, the local optimization algorithm is more powerful, being a restricted neighborhood version of 3-Opt (like Kirkpatrick's, but with the shortest path having length at most 3). Care has also been taken to optimize the rules by which mating pairs are chosen.

The most recent reports on this approach [38] are for the 532-city instance optimally solved in [40], and are impressive (in the appropriate context). The optimal solution for this problem is 27,686, and Mühlenbein was able to find a solution of length 27,702 in under 3 hours on a network of 64 transputers. This is not only less than 0.06% above optimal, it is better than the best tour we were able to find in 20,000 independent runs of Lin-Kernighan (27,705), which took roughly 530 hours of Sequent processor time.

One might question whether such a large expenditure of time is worth the small improvement one obtains over simply running Lin-Kernighan once (which averages 0.98% over optimal in under two minutes) or 100 times (0.25% in under 3 hours), especially when the true optimal can be found and verified using branch-and-cut in 5 hours if one is able to obtain the appropriate code and supercomputer [40,41]. Still, it is interesting to consider the limits of what can be obtained by heuristics. In this vein, let us turn to a variant on the above approach that has yielded even better results for the 532-city instance.

In [36], Martin et al. have proposed what one might call a "parthenogenetic" algorithm. The "population" consists of a single tour, and instead of mating we have random mutations. The current tour is subjected to a random 4-Opt move, biased so as to create short new edges where possible, and the resulting tour is used as a starting tour for 3-Opt. The approach outlined in [36] also has an element of simulated annealing, as the resulting tour may or may not become the new "current tour," depending on a process like that of steps 2.2-2.3 in Figure 2. In 250 hours on a Sun-3, a tour of length 27,693 was found by this approach, and with an additional 300 hours and some additional tricks, this was converted to an optimal tour of length 27,686.

Inspired by these results, we were curious whether similar successes could be obtained by a more-straightforward parthenogenetic approach based on Lin-Kernighan, as described in Figure 3. Note that this algorithm can be viewed as nesting a local optimization algorithm within a randomized local optimization algorithm, and as such might form a more general paradigm (although we shall not pursue that aspect further here).

1. Generate a random tour T.
2. Do the following for some prespecified number M of iterations:
    2.1. Perform an (unbiased) random 4-Opt move on T, obtaining T'.
    2.2. Run Lin-Kernighan on T', obtaining T''.
    2.3. If $length(T'') \leq length(T)$, set $T = T''$.
3. Return T.

Figure 3. Iterated Lin-Kernighan

A first observation is that, when run inside this loop, Lin-Kernighan is typically confronted with a tour that is very nearly locally-optimal to begin with, and consequently its

time per iteration decreases substantially from that required by independent runs. This effect is quantified in the last line of Table 4. A second observation is that Iterated Lin-Kernighan (Iterated LK) more than holds its own in the high cycle-count sweepstakes.

For the 532-city instance of [40,41], if we set $M = 500$ the running time was roughly 8 hours on the Sequent (45 minutes on the VAX-8550, 12 minutes on a MIPS processor), and an optimal tour (length 27,686) was output on 6 of 20 runs. (Here "run" refers to the overall sequence of M iterations.) The average final tour length was 27,699. For the 318-city instance of [33], optimal tours were found on 12 of 20 runs with $M = 200$ (under 2 Sequent hours per run). Indeed, Iterated LK has found optimal solutions for all the solved instances we have been able to obtain, including the 2392-city instance of [40,41], although its running time increases substantially for the larger instances, and its success rate goes down. For the 2392-city instance, we needed $M = 1000$ to get optimal tours on 2 of 20 runs, and each run took roughly 55 Sequent hours.

Reducing the time to more moderate levels still yields worthwhile results, however. With $M = 50$, we averaged less than 0.65% above optimal in 4 hours of Sequent time (24 minutes on the VAX-8550). This is a substantial improvement on the average Lin-Kernighan solution (1.54% above optimal), and on the best one could expect performing independent runs for 4 hours (1.15% above). On the 1000-city random geometric instance of Table 5, Iterated LK with $M = 100$ takes just two Sequent hours and averages 1.07% over the Held-Karp lower bound. This is better than the best solution found doing the 10,000 independent Lin-Kernighan runs, which collectively took 486 hours. For more on Iterated LK and our uses of it, see [19].

7. Concluding Remarks

An inescapable general conclusion, in light of the results presented here for approximation algorithms and those reported in [13,40,41] for optimization algorithms, is that the TSP is in practice much less formidable than its reputation would suggest. More specifically, the triumvirate of 3-Opt, Lin-Kernighan, and Iterated LK together establish an impressive (time/tour quality) trade-off curve and represent stiff competition for any new TSP heuristics. We have not had time here to discuss such additional approaches as tabu search [12], threshold accepting [9], neural nets [16], or the elastic band algorithm of [10], but all appear to be slower than single-run Lin-Kernighan, and we have seen no reports indicating that any of them provides better tours. (Indeed, the neural net approach of [16] cannot even outperform 2-Opt at 30 cities [17].)

There is, however, room for improvement within the triumvirate. For example, our current implementations store tours in arrays, and hence reversing a subpath must take time proportional to its length. We are now investigating whether alternative representations might become more cost-effective as instances grow to 100,000 cities and above [20].

There are also possibilities for speeding up Iterated LK, perhaps by using information from earlier tours to restrict the selection of the random 4-Opt move in Step 2.1. Currently Iterated LK can come within 0.8% of the Held-Karp lower bound on random Euclidean instances of sizes from 100 to 10,000, but the number of iterations needed to do this seems to be growing linearly, with most iterations yielding poorer tours and hence no progress. Even on a MIPS processor, 10,000 iterations on a 10,000-city instance takes 75 hours, so making individual iterations more effective is a first priority.

REFERENCES

1. J. L. BENTLEY, "Experiments on traveling salesman heuristics," in *Proc. 1st Ann. ACM-SIAM Symp. on Discrete Algorithms*, SIAM, Philadelphia, PA, 1990, 91-99.
2. J. L. BENTLEY, "Fast algorithms for geometric traveling salesman problems," in preparation.
3. J. L. BENTLEY, D. S. JOHNSON, L. A. MCGEOCH, AND E. E. ROTHBERG, "Near-optimal solutions to very large traveling salesman problems," in preparation.
4. R. G. BLAND AND D. F. SHALLCROSS, "Large traveling salesman problems arising from experiments in X-ray crystallography: A preliminary report on computation," *Operations Res. Lett.* **8** (1989), 125-128.
5. R. M. BRADY, "Optimization strategies gleaned from biological evolution," *Nature* **317** (October 31, 1985), 804-806.
6. N. E. COLLINS, R. W. EGLESE, AND B. L. GOLDEN, "Simulated Annealing: An Annotated Bibliography," *Amer. J. Math. & Mgmt. Sci.* **8** (1988), 205-307.
7. V. CERNY, "A Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm," *J. Optimization Theory and Appl.* **45** (1985), 41-51.
8. N. CHRISTOFIDES, "Worst-case analysis of a new heuristic for the travelling salesman problem," Report No. 388, GSIA, Carnegie-Mellon University, Pittsburgh, PA, 1976.
9. G. DUECK, "Threshold Accepting: A general purpose optimization algorithm appearing superior to simulated annealing," TR88.10.011, Heidelberg Scientific Center, IBM Germany, 1988.
10. R. DURBIN AND D. WILLSHAW, "An analogue approach to the travelling salesman problem using an elastic net method," *Nature* **326** (April 16, 1987), 689-691.
11. M. R. GAREY, AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
12. F. GLOVER, "Tabu search – Part I," *ORSA J. Comput.* **1** (1989), 190-206.
13. M. GRÖTSCHEL AND O. HOLLAND, "Solution of large-scale symmetric travelling salesman problems," Report No. 73, Institut für Mathematik, Universität Augsburg, 1988.
14. M. HELD AND R. M. KARP, "The traveling-salesman problem and minimum spanning trees," *Operations Res.* **18** (1970), 1138-1162.
15. M. HELD AND R. M. KARP, "The traveling-salesman problem and minimum spanning trees: Part II," *Math. Programming* **1** (1971), 6-25.
16. J. J. HOPFIELD AND D. W. TANK, "'Neural' computation of decisions in optimization problems," *Biol. Cybern* **52** (1985), 141-152.
17. D. S. JOHNSON, "More approaches to the travelling salesman guide," *Nature* **330** (December 10, 1987), 525.
18. D. S. JOHNSON, C. R. ARAGON, L. A. MCGEOCH, AND C. SCHEVON, "Optimization by Simulated Annealing: An Experimental Evaluation, Part I (Graph Partitioning)," *Operations Res.* **37** (1989), 865-892.
19. D. S. JOHNSON, C. R. ARAGON, L. A. MCGEOCH, AND C. SCHEVON, "Optimization by Simulated Annealing: An Experimental Evaluation, Part III (The Traveling Salesman Problem)," in preparation.
20. D. S. JOHNSON, L. A. MCGEOCH, AND G. OSTHEIMER, "Data structures for traveling salesmen," in preparation.
21. D. S. JOHNSON, C. H. PAPADIMITRIOU, AND M. YANNAKAKIS, "How easy is local search?," *J. Comput. System Sci.* **37** (1988), 79-100.
22. D. S. JOHNSON AND E. E. ROTHBERG, "Asymptotic experimental analysis of the Held-Karp lower bound for the traveling salesman problem," in preparation.
23. R. M. KARP, "Probabilistic analysis of partitioning algorithms for the traveling-salesman in the plane," *Math. Oper. Res.* **2** (1977), 209-224.
24. W. KERN, "A probabilistic analysis of the switching algorithm for the Euclidean TSP," *Math. Programming* **44** (1989), 213-219.

25. S. KIRKPATRICK, "Optimization by simulated annealing: Quantitative studies," *J. Stat. Physics* **34** (1984), 976-986.
26. S. KIRKPATRICK, C. D. GELATT, JR, AND M. P. VECCHI, "Optimization by Simulated Annealing," *Science* **220** (13 May 1983), 671-680.
27. S. KIRKPATRICK AND G. TOULOUSE, "Configuration space and the travelling salesman problem," *J. Physique* **46** (1985), 1277-1292.
28. B. KORTE, "Applications of combinatorial optimization," talk at the 13th International Mathematical Programming Symposium, Tokyo, 1988.
29. W. KRAUTH AND M MÉZARD, "The cavity method and the travelling-salesman problem," *Europhys. Lett.* **8** (1989), 213-218.
30. J. LAM AND J.-M. DELOSME, "An efficient simulated annealing schedule: implementation and evaluation," manuscript (1988).
31. E. L. LAWLER, J. K. LENSTRA, A. H. G. RINNOOY KAN, AND D. B. SHMOYS, *The Traveling Salesman Problem*, John Wiley & Sons, Chichester, 1985.
32. S. LIN, "Computer solutions of the traveling salesman problem," *Bell Syst. Tech. J.* **44** (1965), 2245-2269.
33. S. LIN AND B. W. KERNIGHAN, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," *Operations Res.* **21** (1973), 498-516.
34. J. D. LITKE, "An improved solution to the traveling salesman problem with thousands of nodes," *Comm. ACM* **27** (1984), 1227-1236.
35. G. LUEKER, manuscript, Princeton University, 1976.
36. O. MARTIN, S. W. OTTO, AND E. W. FELTEN, "Large-step Markov chains for the traveling salesman problem," manuscript (1989).
37. D. MITRA, F. ROMEO, AND A. SANGIOVANNI-VINCENTELLI, "Convergence and finite-time behavior of simulated annealing," *J. Advan. Appl. Prob.* **18** (1986), 747-771.
38. H. MÜHLENBEIN, "The dynamics of evolution and learning – Towards genetic neural networks," manuscript (1989).
39. H. MÜHLENBEIN, M. GORGES-SCHLEUTER, AND O. KRÄMER, "Evolution algorithms in combinatorial optimization," *Parallel Comput.* **7** (1988), 65-85.
40. M. PADBERG AND G. RINALDI, "Optimization of a 532-city symmetric traveling salesman problem by branch and cut," *Operations Res. Lett.* **6** (1987), 1-7.
41. M. PADBERG AND G. RINALDI, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," Report R. 247, Istituto di Analisi dei Sistimi ed Informatica del CNR, Rome, 1988.
42. C. H. PAPADIMITRIOU, "The complexity of the Lin-Kernighan heuristic for the traveling salesman problem," manuscript (1990).
43. C. H. PAPADIMITRIOU AND K. STEIGLITZ, "On the complexity of local search for the traveling salesman problem," *SIAM J. Comput.* **6** (1977), 76-83.
44. C. H. PAPADIMITRIOU AND K. STEIGLITZ, "Some examples of difficult traveling salesman problems," *Operations Res.* **26** (1978), 434-443.
45. L. K. PLATZMAN AND J. J. BARTHOLDI, III, "Spacefilling curves and the planar travelling salesman problem," *J. Assoc. Comput. Mach.* **36** (1989), 719-737.
46. D. J. ROSENKRANTZ, R. E. STEARNS, AND P. M. LEWIS, II, "An analysis of several heuristics for the traveling salesman problem," *SIAM J. Comput.* **6** (1977), 563-581.
47. S. SAHNI AND T. GONZALEZ, "P-complete approximation problems," *J. Assoc. Comput. Mach.* **23** (1976), 555-565.
48. G. H. SASAKI AND B. HAJEK, "The time complexity of maximum matching by simulated annealing," *J. Assoc. Comput. Mach.* **35** (1988), 387-403.
49. H. S. STONE, private communication (1989).