Large-Step Markov Chains for the Traveling Salesman Problem *

Olivier Martin
Department of Physics, CCNY, New York, NY, 10031, USA

Steve W. Otto
Department of Computer Science and Engineering,
Oregon Graduate Institute of Science and Techhnology,
19600 NW von Neumann Dr, Beaverton, OR, 97006-1999, USA
otto@cse.ogi.edu

Edward W. Felten
Department of Computer Science,
University of Washington, Seattle, WA, 98195, USA

January 29, 1991

Abstract

We introduce a new class of Markov chain Monte Carlo search procedures, leading to more powerful optimization methods than simulated annealing. The main idea is to embed deterministic local search techniques into stochastic algorithms. The Monte Carlo explores only local optima, and it is able to make large, global changes, even at low temperatures, thus overcoming large barriers in configuration space. We test these procedures in the case of the Traveling Salesman Problem. The embedded local searches we use are 3-opt and Lin-Kernighan. The large change or step consists of a special kind of 4-change followed by local-opt minimization. We test this algorithm on a number of instances. The power of the method is illustrated by solving to optimality some large problems such as the LIN318, the AT&T532, and the RAT783 problems. For even larger instances with randomly distributed cities, the Markov chain procedure improves 3-opt by over 1.6%, and Lin-Kernighan by 1.3%, leading to a new best heuristic.

*This manuscript was published in *Complex Systems*, v. 5:3, pg. 299, 1991

1 Introduction

The Traveling Salesman Problem (TSP) is probably the most well-known member of the wider field of Combinatorial Optimization (CO) problems. These are optimization problems where the set of feasible solutions (trial solutions which satisfy the constraints of the problem but are not necessarily optimal) is a finite, though usually very large set. The number of feasible solutions grows as some combinatoric factor such as $N!$ where N characterizes the size of the problem. One technique for solving these problems is exhaustive search of all feasible solutions. This method, however, has time complexity typically growing as $N!$ and so is not a viable technique for problems of interesting size.

One might ask whether there are much faster techniques than exhaustive search. Among optimization problems in general, the TSP is a member of the set NP-complete. This is a class of difficult optimization problems whose time complexity is probably exponential [1]: even the most clever algorithms suffer from this time growth. The members of NP-complete are related so that if a polynomial time algorithm were found for one problem, polynomial time algorithms would exist for all members of NP-complete. All CO problems can be formulated as optimizing an objective function (e.g., the length) subject to constraints (e.g., legal tours). It has often been the case that progress on the TSP has led to progress on other CO problems and on more general optimization problems. In this way, the TSP is a playground for the study of NP-complete problems. Though the present work concentrates on the TSP, a number of our ideas are general and apply to all optimization problems.

The most significant issues occur as one tries to find extremely good or exact solutions to the TSP. Many algorithms exist which are fast and find feasible solutions which are within a few percent of the optimum length. In this paper we present algorithms which will usually find exact solutions to substantial instances of the TSP, for example, up to $N \sim 1000$.

In a general instance of the TSP one is given N "cities" and a matrix d_{ij} giving the distance or cost function for going from city i to j. Without loss of generality, the distances can be assumed positive. A "tour" consists of a list of N cities, $tour[i]$, where each city appears once and only once. In the TSP, the problem is to find the tour with the minimum "length," where length is defined to be the sum of the lengths along each step of the tour,

$$length = \sum_{k=0}^{N-1} d_{tour[k],tour[k+1]},$$

and $tour[N]$ is identified with $tour[0]$ to make it periodic. Most common instances of the TSP have a symmetric d_{ij} matrix; we will hereafter focus on this case, which also is in NP-complete.

2 Overview of Algorithms for the TSP

Before presenting the details of our work, we discuss the main methods employed for the TSP. This helps to show where this work fits in and also provides some needed background.

There are a number of exact methods (i.e., which are guaranteed to find the exact optimum in a bounded number of steps) for solving the TSP. One family consists of the Branch and Bound algorithm of Held and Karp [2, 3] and its derivatives [4]. These algorithms attempt to prove that sets of links belong or do not belong to the optimal tour, using bounds from, for example, minimal spanning trees. There exist transformations on the distance matrix which leave the relative ranking of all tours unaffected but which change the spanning tree sub-problems. One then maximizes over these transformations, obtaining the tightest possible spanning tree bound, causing the branch and bound tree to prune most rapidly. Though the pruning is dramatic, branch and bound is still an exponential (in N) algorithm.

To date, the most effective exact methods are the cutting-plane or facet-finding algorithms [5, 6]. These use an integer linear programing formulation of the TSP. Roughly speaking, various constraints are added to a linear programming problem until the solution found is a legal tour. The performance of these methods are strongly dependent on the kinds of constraints that are added and they are still to some extent an art form. In the last ten years, these exact methods have been pursued so vigorously that it is now possible to exactly solve problems with several hundred cities [6, 7]. The state of the art algorithms are quite complex, with codes on the order of 9000 lines.

There are also many approximate or heuristic algorithms. These obtain good solutions in a (relatively) small amount of time but do not guarantee that the optimal solution will be found.

There is a class of heuristic algorithms which simply directly construct tours by some rule. The simplest of these is the trivial "greedy" algorithm which goes as follows. Start with some (randomly selected) city. Now take as the first link of the tour the step from this city to its closest neighbor. From the second city, step to the nearest city which still has not yet appeared in the tour. Continue in this fashion until no cities remain. The final step is from the last city to the first city. The tours which greedy produces look reasonable for the most part, except for a few long links which come from the end of the process, when few cities remain and it is difficult to find a close-by, untaken city.

Once greedy or something like it has given one a vaguely reasonable tour, the idea naturally presents itself to look for ways to improve a given tour. This leads to the class of "local search" algorithms. These methods sequentially construct a chain of tours: usually the ith tour is constructed from the $(i-1)$th tour by changing some number of links. Local search algorithms demand that the tour strictly improve as one goes from one tour to the next — that is, the tours are constructed so as to decrease the length at each step. The most effective such algorithms are those of Lin [8] and Lin and Kernighan [9]. Lin starts with the idea of a k-change: take the current tour and remove k different links from it. Now re-connect the dangling sections in a new way so as to again achieve a legal tour. A tour is considered to be "k-opt" if no k-change exists which decreases the length of the tour. Lin's algorithm begins with a random tour and applies 2 and 3-changes until one reaches a 3-opt (and also 2-opt) tour. He found that the 3-opt heuristic was quite powerful: for a problem of moderate size ($N = 48$), 3-opt from a random start had a non-negligible probability ($\sim 5\%$) of hitting the exact optimum. Therefore by taking many random starts, he was almost certain to find the exact optimum for problems of this size. Lin also tried higher k-changes but decided that they were not worthwhile, though it should be realized that this conclusion depends on the speed of the k-opt algorithm for $k > 3$. If a fast algorithm can be found for $k > 3$, it may very well be worthwhile to go to 4-opt or beyond.

Lin and Kernighan improved on these ideas by both speeding up the 3-opt process and also by including some of the higher-order k-changes. In their algorithm, the order of a change is not pre-determined, rather k is increased until a stopping criterion is satisfied. Thus many kinds of k-changes and all 3-changes are included. In practice, there are many ways to choose the stopping criteria, and the best codes are rather involved. The Lin and Kernighan method is a powerful heuristic and is considered to be the benchmark against which all other heuristics are compared. Surprisingly, there have not been significant improvements in performance of local search algorithms since the work of Lin and Kernighan which goes back to 1973.

Local search algorithms tend to get trapped in local minima of the objective function. They proceed downhill for a while, making much progress, but then stop. In order to make more progress, many links would have to be simultaneously changed in a single k-change, for some large value of k. Another class of algorithms is possible in which one relaxes the strict downhill restriction of the chain of tours and actually lets the tour length (occasionally) increase. In this way, one can hope that the tour will climb out from the current local minimum and cross over a barrier to a better solution. We call this class of algorithms "iterative sampling." This class includes simulated annealing and genetic algorithms.

In simulated annealing [10, 11], the uphill moves are accomplished by introducing a "temperature" and updating the system according to the Metropolis rule. A trial move is made for instance, by applying a 2 or 3-change to the current tour; if this gives a downhill change, it is always accepted, while an uphill move is accepted with conditional probability $e^{-\Delta L/T}$. ΔL is the change in the length due to the trial move and T is the temperature, a free parameter which controls the typical size of ΔL. One thus constructs what is called a "Markov chain" of tours. Markov chains are distinguished from more general types of chains by the requirement that the ith tour is constructed strictly from the $(i-1)$th tour (and not, for instance, from both the $(i-1)$ and $(i-2)$ tours).

If the trial moves satisfy a certain symmetry property (we will return to this later) then a tour of length L will appear with probability proportional to $e^{-L/T}$. The exact optimum is the single most likely configuration to appear, but this is counter-acted by the fact that there is such a large number of tours even slightly above the optimum. This means that with simulated annealing the system will almost always be in a sub-optimal tour. To fight this, one attempts to drive the system towards the true optimum by slowly lowering the temperature T, and this is termed annealing. If the annealing is done "sufficiently" slowly, one is guaranteed to find the true optimum if one waits long enough, but this is almost impossible to achieve in practice. See Bentley and Johnson for an an extensive comparison of the above heuristics [12].

Another type of iterative sampling algorithm is the class of so-called "genetic algorithms" [13, 14, 15]. Here one starts with an ensemble of tours which "compete:" the best tours replicate and the worst tours are eliminated. To create new kinds of tours, one applies "mutations" such as random k-changes and "cross-overs" where two or more tours are in some way put together to create a new tour. This approach has not yet been systematically explored and probably can be significantly improved.

A very different approach has generated much interest recently — the neural network approach of Hopfield and Tank [16]. In this method, the constraint of "legal" tours is not strictly enforced during the computation. (Note that this also occurs in cutting-plane algorithms.) In practice, the method has not yet been successful at solving problems of size 40 or greater [17].

In Section 3, we introduce a class of Markov chains in which each step is produced by a "kick" followed by a local search optimization. The local search method is described in section 4 and a number of other program optimizations are given in section 5. Section 6 presents our results of local search timings and the application of the entire method to a number of solved Euclidean TSPs. Some background material on the density of tours of the TSP is given in Appendix A, and Appendix B discusses some properties of Markov chains.

3 Large-Step Markov Chains

The algorithms of Lin and Lin-Kernighan are powerful because they consider many possible changes to a tour. This means that the 'local-opt' criterion is rather stringent, and only a very small subset of all possible tours are generated. Furthermore, the length of such tours are typically near the optimum. In this section we show how to combine this good feature of the local search method with Markov chains so as to produce a more powerful type of Monte Carlo procedure than the standard simulated annealing method.

Throughout this paper we will concentrate on TSP's using a two dimensional Euclidean metric, that is, the d_{ij} elements correspond to distances in a plane, but our methodology does not depend on this. The density of "states" (i.e., tours) away from the optimum of a TSP instance increases rapidly as a function of length. As discussed in Appendix A, the density of states (i.e., tours) near the optimum is a rapidly rising function because distant parts of the tour can be modified almost independently. This independence then leads to a combinatoric factor in the number of tours away from the optimum.

If an algorithm samples all these states it will not find the optimal solution for problems of significant size: the density of states strongly biases the system away from the optimum, and the odds of actually hitting the optimum become negligible. The first thing to do to make the TSP more manageable is to thin the set of tours to be considered. This is what the local search algorithms do.

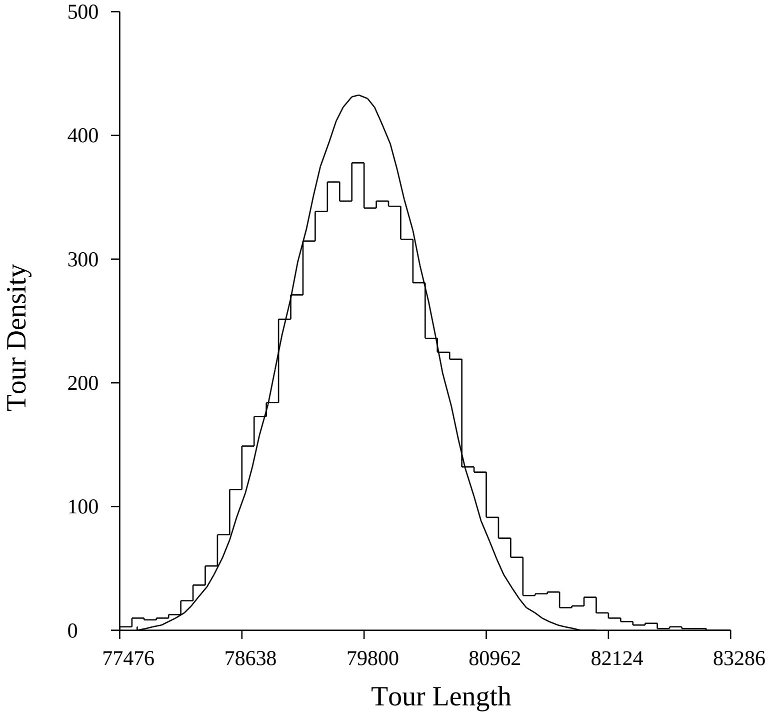

Figure 1: Binned density of 3-opt tours for a random-scatter $N = 100$ Euclidean TSP in two dimensions. The curve is the prediction of the model described in Appendix A. The histogram contains 6138 distinct 3-opt tours — these were found by running the large-step Monte Carlo at high T for a long time.

Figure 1 shows the (binned) density of 3-opt tours for a particular instance of a 100 city TSP (the cities were randomly scattered in a square and the curve is from the model given in Appendix A). The striking feature here is that the distribution falls off very quickly below a few percent from the minimum length. In fact, even in this range, the density of 3-opt tours is much smaller than the density of all tours. One can say the Lin and Lin-Kernighan algorithms are effective because they dramatically reduce the size of the search space. Empirically, it appears that the set of all the 3-opt tours are sampled relatively flatly: each 3-opt tour appears with similar probability if one begins with random starts. In particular, there is no strong bias towards the optimum. This is not surprising, since there is nothing built into the algorithm which would produce this bias, apart from the fact that 3-opt tours are fairly close to the optimum length. For small N, the set of 3-opt tours is manageable and the algorithms can actually find the optimum by repeated trials, which almost amounts to enumeration.

However, for large N, the set of 3-opt (and more generally, locally optimal) tours itself becomes too large to enumerate. To improve the efficiency of the algorithm, we need to bias the *sampling* of locally optimal tours towards tours of shorter length. By using a Markov chain it is possible to sample the set in a more intelligent way. The idea is to construct locally optimal tours from previous ones, not from random starts. Such an algorithm superficially resembles simulated annealing (there is an accept/reject step and an analogue of temperature) but the important difference is that one restricts the tours in the chain to be locally optimal.

Let us now schematically present our algorithm. To be specific, we will consider the local opt procedure to be 3-opt, but the methodology applies to any local opt, in particular Lin-Kernighan. Suppose the current tour is 3-opt. Figure 2 is a schematic representation of the objective function versus tours; the 3-opt tours are at local minima of this function. The goal is to construct a Monte Carlo step (a step in the Markov chain from the ith tour to the $(i+1)$th tour) which goes directly from local minimum to local minimum, biased towards shorter lengths. We accomplish this in the following way. Starting at the current 3-opt tour (labeled *Start* in Figure 2), we give the tour a large "kick", taking it to *Intermediate*. We will describe in more detail later what we use for the kick — for now it can be thought of as a randomly selected k-change for some $k > 3$. We now apply an efficient 3-opt procedure to *Intermediate*. This brings us to a new local minimum, labeled *Trial* in figure 2. So far we have stepped from one 3-opt tour to another. We do not merely accept this new tour, however, since we wish to bias towards short lengths: we apply an accept/reject procedure to *Trial*. If the length has decreased, *Trial* is accepted and becomes the new current tour. If the length has increased, it is accepted with conditional probability $e^{-\Delta L/T}$; if the acceptance test fails, the tour is returned to *Start*. This forms one step of the Markov chain. This procedure is repeated many times, exploring local-opt tours in a biased way. In particular, one expects to sample more often the local-opt tours of shortest lengths than with algorithms which sample the locally optimal tours randomly.

This algorithm gives rise to a large-step Markov Chain because after the kick

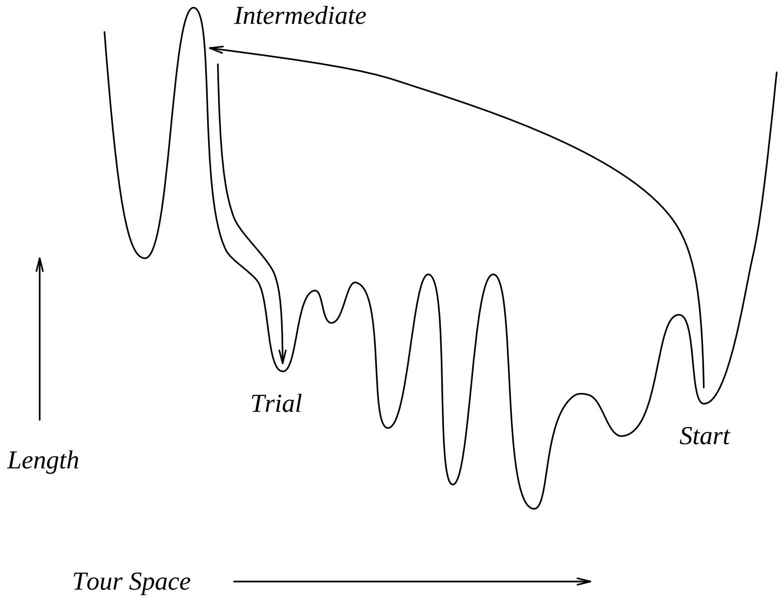

Figure 2: Schematic representation of the objective function and of the tour modification procedure used in the large-step Markov chain.

and local-opt are applied, typically many links have been changed. If we take as the metric of the TSP the number of links by which two tours differ, we may say that *Trial* is often quite "far" from *Start*. Large-step Markov Chains are powerful because they can reduce the auto-correlation time of the Markov chain and the search space is explored rapidly. Appendix B discusses these points in greater depth. It is important to realize that finding a practical large-step Markov Chain is not a simple matter. After all, we could have taken as the large step simply a k-change for some large value of k. The performance, however, would be terrible since a randomly selected k-change would just take *Start* to a random location in the space of all tours; *Trial* would then almost always be rejected. Furthermore, it is essential to employ a very good tour improvement method so as to bring us not only to a new tour, but to a new, high quality tour. Our approach is reminiscent of the "spin wave" moves suggested by Ceperly and Kalos for physics Monte Carlos [18]. For an efficient algorithm, the choice of the large step must be specifically tailored to the problem at hand, which is why we have chosen a specific kind of "kick" which is well suited for the TSP.

Another way of thinking of the advantage of large-step versus small-step Monte Carlo (e.g., simulated annealing) is the following. In going from one local-opt solution to another by some number of link changes, a barrier (i.e., a longer tour) is encountered. In a small step Monte Carlo, the intermediate tour with larger length must first be accepted in order to proceed. If one wants to sample very near the optimal length, this forces the "temperature" to be low, leading to a very low acceptance of such intermediate steps: the algorithm thus gets stuck for exponentially long times in valleys of the objective function. The large steps allow one to climb over some of the barriers and have the accept/reject test only *after* having returned to a valley. Thus large steps should be constructed so that barriers are easily jumped over. In effect, the objective function landscape has been smoothed and many of the ridges have been eliminated. This shows up quite dramatically during our runs: even at very low T, near the optimum, the large-step Monte Carlo continues to explore new tours. It is very effective at avoiding trapping.

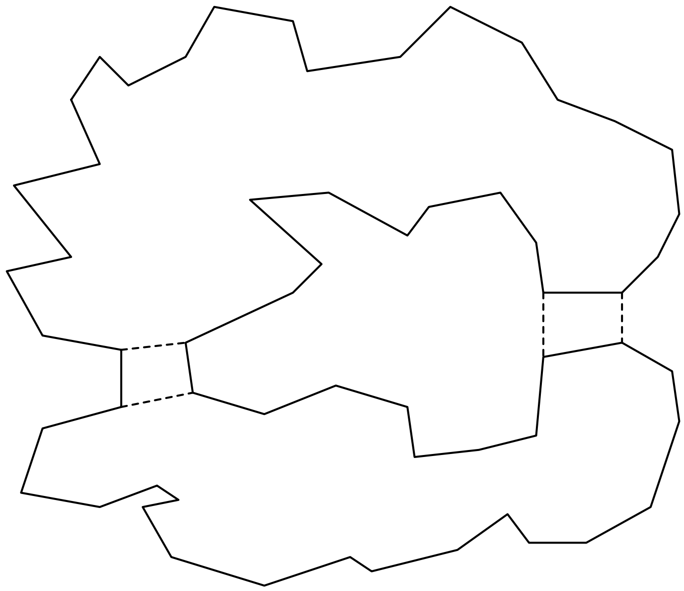

Figure 3: Example of a double-bridge kick (shown in dashed lines). The bridges rearrange the connectivity of the tour on large scales.

For the "kick" we have chosen the particular type of 4-change drawn in Figure 3. It consists of a pair of improper 2-changes. Each improper 2-change is a "bridge," i.e., it takes a legal, connected tour into two disconnected parts. The combination of both bridges, of course, must be chosen so as to produce a legal final tour. The motivation for this type of kick is evident from the figure — it allows a peninsula to hop from one place in the tour to another without much of an increase in the tour length. Obviously this is just one choice for the kick but we have found it to be an effective way of getting the Monte Carlo to rapidly explore the space of local-opt tours. The double-bridge kick enables large-scale changes in the current tour to take place. The double bridges can be generated randomly, or with some bias towards allowing nearby peninsulas to hop as in Figure 3. The important point is that the double-bridge move is the simplest move which cannot be built from the composition of a local sequence of 2 and 3-changes.

The ideas of this section are quite general. For any optimization problem for which powerful local search methods or other heuristics are known, one can incorporate these into large-step Monte Carlos which generate only interesting feasible solutions and also bias the sampling of these solutions towards the true optimum. We are aware of one previous work which uses large step Markov chains [19]. In that reference, the authors present a method for finding the ground states of proteins, but they did not consider it to be a general optimization method. In addition, they had rather limited success with their method, perhaps because their large step was not very suited to their problem. Note also that, contrary to their claim, their algorithm does not satisfy detailed balance, so the Boltzmann distribution is not obtained.

Lack of Detailed Balance

In the case of standard simulated annealing, the trial moves which one feeds to the accept/reject part of the Metropolis procedure appear with symmetric probabilities. That is, if $T_{A \to B}$ is the probability of the trial move $A \to B$ (before one applies the accept/reject test), then $T_{A \to B} = T_{B \to A}$. This, combined with the form of the Metropolis accept/reject procedure leads to a total transition probability which has the property of "detailed balance." This ensures that the system asymptotically reaches thermal equilibrium (the ensemble reaches a steady state) and that the states are populated with probability proportional to $e^{-L/T}$ (the Boltzmann distribution). The algorithm is thus closely analogous to a physical system in thermal equilibrium.

This is not the case for the large-step Monte Carlo. $T_{Start \to Trial}$ is the probability that the move $Start \to Trial$ in figure 2 is attempted. There is no reason for the inverse trial move, $Trial \to Start$, to appear with the same probability. Thus the algorithm does not satisfy detailed balance and there is no direct physical (statistical mechanical) analogue of the large-step Monte Carlo. The sampling of the tours is biased towards the optimum, but not necessarily by the Boltzmann factor $e^{-L/T}$.

4 Fast Local Searches

Fundamental to the large-step Monte Carlo is an efficient local search procedure. The Lin-Kernighan algorithm takes a tour to 3-opt and beyond because it includes some of the higher-order k-changes. It is very fast: checking that a tour is L-K opt takes on the order of N operations. This is to be contrasted with the k-opt algorithms introduced by Lin which require $O(N^k)$ steps. More recently, an $O(N)$ check-out time approximation to 3-local-opt was presented

[20] as a heuristic way of getting close to 3-opt quality tours. The purpose of this section is to show that 3-opt can, with no approximation, be realized so that check-out time is $O(N)$ rather than $O(N^3)$ as is commonly used by practitioners in the field [4]. The running time for our fast 3-opt is similar to Lin-Kernighan restricted to 2 and 3 changes giving rise to 3-opt tours. A possible advantage of the algorithm below is that the method generalizes to all "connected" graphs for any k.

We begin with the case of 2-opt which consists of improving a tour until its length cannot be decreased by any 2-changes. If a 2-change leads to a decrease of the tour length, we implement the exchange and this requires inverting and re-writing part of the tour. This single 2-change costs an amount of computation time, $d(N)$, which depends on the quality of the current tour. If the current tour is very bad, for instance, a random tour, $d(N)$ is proportional to N. For "good" tours, $d(N)$ can be much less, proportional to N^α with $\alpha < 1$. Let us suppose from now on that the tour is "reasonable," for example it was obtained by "kicking" a 3-opt tour as is the case in our large-step Monte Carlo.

To find a 2-change which decreases the tour length, we must consider all pairs of links for a possible exchange. Naïvely, this requires $O(N^2)$ steps, but in fact it is silly to consider pairs of links which are very far apart in the physical space of the problem. Figure 4 shows such an example. Intuitively, it is clear that the 2-change obtained by cutting the two marked links will increase rather than decrease the length.

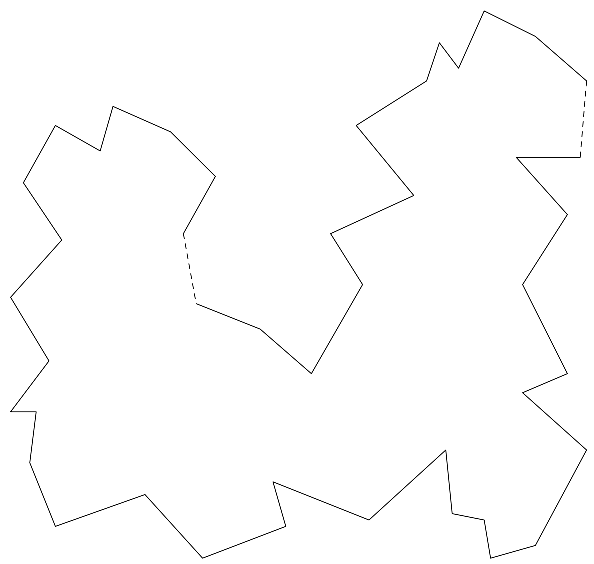

Figure 4: A candidate pair (dashed lines) for a 2-change. Clearly, this 2-change cannot decrease the tour length.

By making this idea precise one arrives at a fast 2-opt algorithm. To do this, we need additional data structures which specify which cities are close to any given city. For simplicity, consider using N^2 storage space to do this; for large problems, one can store say the 20 nearest neighbors only, or to be completely rigorous, a tree data structure can be constructed. Let the cities be labeled by an index which goes from 0 to $N-1$. The tour is represented by a mapping from tour entries to city labels. This is simply the *tour* array:

```
tour[j] = i;     /* city i is the jth city in the tour */
```

The *neighborhood* array is defined by:

```
nbhd[i][j] = k;  /* city k is the jth closest city to city i */
```

This array is found by sorting each row of the distance matrix. This is done once and for all at the beginning of a run. To efficiently keep track of where the cities are in the current tour, we introduce the *which_slot* data structure defined by:

```
cities[i].which_slot = j;  /* city i is currently the jth city in the tour */
```

This structure must be updated as the tour changes; updating *which_slot* is a similar amount of work to updating the tour itself. Finally, there are two more necessary data items. *min_link* is defined as the minimum size any link can take. This is easily found once at the beginning of a run by simply finding the minimum value of d_{ij} over all i and j. *max_link* is defined to be the value of the largest link in the *current* tour. This quantity must be dynamically updated (this can be done incrementally and with few operations).

```
for (n_1 =0;n_1 <N;++n_1) {              /* loop through tour slots */
    m_1 = ( n_1 - 1 + N ) % N;
    for (j_2 =0;j_2 <N-1;++j_2) {
        c_2 = nbhd[n_1][j_2];
        n_2 = cities[c_2].which_slot;  /* n_2 goes out from city tour[n_1] */
        m_2 = ( n_2 - 1 + N ) % N;
        if ( d[tour[n_1]][tour[n_2]]+min_link >
     d[tour[m_1]][tour[n_1]]+max_link ) {
            break;                     /* out of j_2 loop; go to next n_1 */
        }
        /* try the move */
        /* if move accepted, break out of j_2 loop and go to next n_1 */
    }
}
```

Table 1: The pseudo-code for fast 2-opt.

C-style pseudo-code for the efficient 2-opt is shown in Table 1. The meaning of n_1, m_1, n_2, and m_2 is shown in figure 5. The crucial step is the **if** statement. The quantity on the left hand side of the expression within the **if** forms a lower bound on the new length; the quantity on the right hand side is an upper bound on the old length. Moreover, the right side is a constant within the enclosing loop while the left hand side is monotonically increasing because of the way n_2 is constructed. Therefore, when these bounds pass each other we can stop considering n_1 as a possible starting point for a 2-change and can go on to the next n_1.

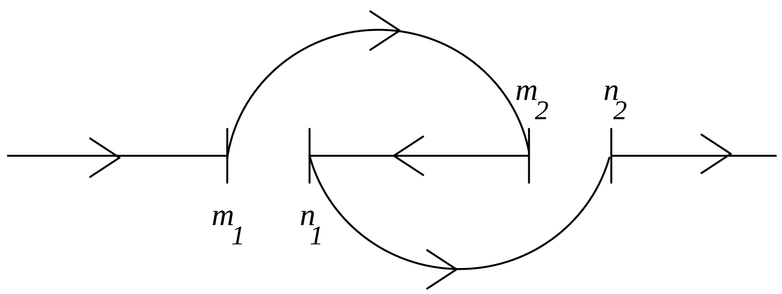

Figure 5: Labels used in the pseudo-code for a 2-change. The two links, (m_1, n_1) and (m_2, n_2) are exchanged for two other links.

The time complexity of this 2-opt algorithm depends on the quality of the current tour through $d(N)$, an *a priori* unknown function. At this point, we can only discuss the N dependence of the "check-out" period [8] which is the time it takes to verify that a tour is 2-opt. (We referred to this time at the begining of this section.) The check-out time is $f(N)N$, where the N comes from the outer loop of the algorithm: all slots are tried at least once. The function $f(N)$ represents the average number of cities within a sphere of radius *(max_link-min_link)* of city $tour[n_1]$. As N grows, one expects (at least for random scatter TSPs) that this is a very slowly growing function of N. The simplest way to see this is by considering the "scaled" TSP problem, where the cities are randomly scattered in a square of size $\sqrt{N} \times \sqrt{N}$ and the length of a good tour is proportional to N. In this case, the typical size of each link in a good tour is of order 1. As N grows, one can have a city density fluctuation which causes *max_link* to grow slowly as a function of N. If *max_link* were a constant, the number of cities within a sphere of radius *max_link-min_link* would be a constant and $f(N)$ would be constant. This is not always the case, however, so $f(N)$ may be slowly growing. All this depends on the fact that during the 2-opt iteration the tour is "reasonable" — i.e., that *(max_link-min_link)* is small. This is true during the large-step Monte Carlo and later we will give timings.

The above discussion is easily extended to 3-opt; naïvely, the check-out time takes $O(N^3)$ steps, but for a reasonable tour it is silly to consider cases where the three links are far apart. If we label the three links to be broken as (m_1, n_1), (m_2, n_2) and (m_3, n_3), there are two topologically different ways to reconnect the points to make a legal tour. Table 2 gives the pseudo-code for the case depicted in figure 6. As in 2-opt, there are branches which give early exits from

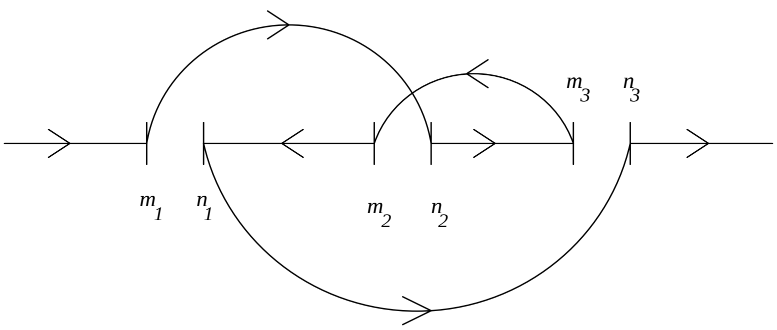

Figure 6: Labels used in the pseudo-code for a 3-change. The three links, (m_1, n_1), (m_2, n_2), and (m_3, n_3) are exchanged for three other links.

the loops. There are several of these corresponding to the tighter bounds which can be found as more of the potential 3-change is constructed. The meaning of the various indices is shown in figure 6. The time complexity for check-out can again be written as $f(N)N$, where now $f(N)$ counts the average number of cities within a sphere of radius $2 \cdot$ *(max_link-min_link)* of city $tour[m_1]$. Again, $f(N)$ has a weak dependence on N and can be measured. The time complexity for the overall 3-opt procedure will be discussed in section 6 (see figure 8).

Though we used the Euclidean planar TSP to motivate this algorithm, nothing actually depends on it. The triangle inequality is not needed; the algorithm given above works for *any* symmetric d_{ij}. It is fast whenever d_{ij} is such that any given city only has relatively few near neighbors (e.g., not all N cities equidistant!). Then $f(N)$ is nearly constant: one can implement 2 and 3-opt so that the check-out time is $O(N)$.

The extension to k-opt with $k > 3$ is not straightforward. In the language of Lin and Kernighan, for $k > 3$, there exist 'non-sequential' or 'disconnected' k-changes. See figure 2 of their paper. Indeed, at $k = 4$, a new type of edge exchange appears. It consists of a 4-change made out of two improper 2-changes. These are nothing but the double-bridge moves which we use for the kick in the large-step Monte Carlo. For such 4-changes, there is no locality bound which constrains the two bridges to be near each other. This is a feature which many k-changes share for $k > 3$. To understand this, consider constructing a k-change sequentially. First, break one link, creating two free cities, 1 and 2, and a dangling link on each. Choose a new city, 3, and connect it to one of the dangling links. Now break one of the old links at city 3 so that there are again a total of two dangling links and two free cities. Continue in this manner. At every stage of this construction, there are two dangling links attached to cities involved in the k-change. At some time, the two dangling links are joined together. If this occurs at the last step, we call the resulting k-change sequential or “connected;” otherwise, it is called “disconnected.” Note that all legal 2 and

```
for (n_1=0;n_1<N;++n_1) {            /* loop through tour slots */
    m_1 = ( n_1 - 1 + N ) % N;
    for (j_2 =0;j_2 <N-1;++j_2) {
        c_2 = nbhd[m_1][j_2];
        n_2 = cities[c_2].which_slot;    /* n_2 goes out from city tour[m_1] */
        m_2 = ( n_2 - 1 + N ) % N;
        if ( d[tour[m_1]][tour[n_2]]+2*min_link
                > d[tour[m_1]][tour[n_1]]+2*max_link ) {
            break;                     /* out of j_2 loop; go to next n_1 */
        }
        if ( d[tour[m_1]][tour[n_2]]+2*min_link
                > d[tour[m_1]][tour[n_1]]+d[tour[m_2]][tour[n_2]]+max_link ) {
            continue;                /* to next j_2 value */
        }
        for (j_3 =0;j_3 <N-1;++j_3) {
            c_3 = nbhd[n_1][j_3];
            n_3 = cities[c_3].which_slot; /* n_3 goes out from city tour[n_1]*/
            if ( d[tour[m_1]][tour[n_2]]+d[tour[n_1]][tour[n_3]]+min_link >
                  d[tour[m_1]][tour[n_1]]+d[tour[m_2]][tour[n_2]]+max_link ) {
                break;  /* out of j_3 loop; go to next j_2 */
            }
            /* try the move */
            /* if move accepted, break out of j_3, j_2 loops, go to next n_1 */
        }
    }
}
```

Table 2: The pseudo-code for fast 3-opt.

3 changes are connected.

The fast algorithm explained for 2 and 3-opt can be extended to connected-k-opt. At each step in the construction of a connected k-change, the new city can be chosen in a neighborhood of the city which it reconnects to. One can put a bound on the size of this neighborhood in the same way as we did for 2 and 3-changes. This leads to an algorithm for which check-out time grows as $f(N)N$, and the tours generated are optimal under connected k-changes.

It is much more difficult to deal with the disconnected k-changes. There seems to be no $O(N)$ algorithm for such k-changes. One role played by these k-changes is to change the large-scale connectivity of the tour. Perhaps instead of doing these changes explicitly as in a local search algorithm, it is more efficient to sample them stochastically. This is what our large-step Monte Carlo does. The choice of the double-bridge as the kick was made for this reason. Thus our algorithm can be viewed as a simple implementation of a large-step Monte Carlo where kicks consisting of disconnected k-changes (which encourage global connectivity changes, the double-bridge being the simplest such move) are followed by a local search sub-algorithm such as 2, 3-opt, k-connected-opt, or L-K.

Let us mention here how the Lin-Kernighan algorithm also achieves a check-out time proportional to $f(N)N$ (this f need not be the same as the one above). Note that if we restrict the Lin-Kernighan algorithm to k-changes for $k \leq 3$, then all possible 2 and 3 changes are considered. The Lin-Kernighan algorithm can be $O(N)$ for check-out time because it considers a class k-changes which are connected. Using the notation of [9], suppose there exists a k-change for which the total gain is positive:

$$\sum_{i=0}^{k} g_i > 0,$$

where g_i is the gain achieved in the ith exchange. Lin and Kernighan show that there always exists a cyclic permutation of the indices such that the partial sums of the g's are also positive. Thus one can impose this as a constraint on the search. This leads to a bound when choosing each new exchange, having a similar effect to our locality bounds.

5 More Tricks

In this section we present some additional optimizations to the large-step Monte Carlo algorithm which allow it to run fast and better explore the space of local-opt tours. We describe the optimizations in the framework of 3-opt, but extensions to other local searches are straightforward.

The Hash Table

Lets suppose that one has an instance with N not too large and that one wants to be confident that one has found the optimal solution. This means that as the Monte Carlo proceeds (at low temperatures), 3-opt tours very near the optimum should be visited many times. Rather than verify again and again that a tour is 3-opt, one can use the well known device of a hash table [21] to store locally opt tours which have previously been visited. If the lookup can be made fast, the hash table can be continually queried as to whether or not the current tour is a member of the table (i.e., has this tour been seen before and is it 3-opt?). If the answer is yes, an early exit from the loops of the 3-opt algorithm can be taken.

A hashing function which maps tours to locations in the hash table should scatter similar tours. That is, two tours which differ even by only a few links should map to different locations in the table. Other useful properties for the hash function are invariance under cyclic permutations of the tour (it doesn't matter which of the N cities is considered to be the "first" city in the tour) and mirror symmetry (invariance under reversal of the tour orientation). To accomplish this, we construct a table of N^2 random integers, r_{ij}, where we associate each r_{ij} with the link l_{ij}, which is one of the N^2 links which may appear in the tour. To properly represent the links, we symmetrize the random number table, so that $r_{ij} = r_{ji}$ (we are working with a symmetric d_{ij}). For a given tour, a useful hash function is given by the following:

$$H = r_{tour[0],tour[1]} \wedge r_{tour[1],tour[2]} \wedge \cdots \wedge r_{tour[N-1],tour[0]}.$$

$\wedge$ means bitwise XOR. We use a hash table of 2^{16} entries and the lowest 16 bits of H form the index into this data structure. The function H has the aforementioned properties and does a good job of scattering tours uniformly across the hash table. At each place in the hash table we store the full (all the bits) hash value, H, plus the length of the tour. If a tour is known to be 3-opt, an entry is made in the table at the corresponding index. When the table is queried, to get a "match," both the H and the tour length must match. If the query says there is a match, the current tour has been seen before and is 3-opt. Both H and the tour length can be computed incrementally as the current tour changes. This means that H and the tour length are always available and the query into the hash table is a fast operation. In our program, we can query the hash table every time the tour changes at all and if the answer is "match" we immediately exit the 3-opt loops. The hash table forms a repository for all the known 3-opt tours for this instance of a TSP and is valuable in speeding up the search.

As defined, this procedure does have a finite probability of making a mistake (matching H's and tour lengths for different tours). We have ignored this since it is extremely unlikely and also since it doesn't lead to a "hard" mistake — it would cause one to miss a 3-opt tour. If this is thought to be a problem, however, the remedy is simple — take more bits for H (we use a 32 bit H) or add some other piece of information which describes the tour.

Another problem is collisions. Since entries are written into the table but are never taken out, the table can become full over a long run for a large problem. When a collision occurs (we want to write to the table, but that entry is already taken by another tour) one could just give up and not write the entry in. This would not lead to errors, it merely reduces the effectiveness of the hash table. However, one can use secondary chaining to avoid collisions. In this technique, a hash table entry may actually be the head of a linked list of entries, and on both queries and writes, the linked list is scanned, if necessary.

Finally, the performance of the hash table optimization depends on the order in which the cities are traversed in the 3-opt loops. The point is, once the "kick" has been made to the tour, the 3-opt subroutine should first concentrate in the area near the bridges. In this way, all the necessary 3-changes may be done early, the hash table may rapidly find a match, and an extremely early exit from the 3-opt loops can often be accomplished. This possible optimization is somewhat related to the one to be described below, but we have not explored the idea of heuristically changing the looping order so as to increase the chances of early hash table exits.

Outer–Loop Optimization, Sub-Linear 3-opt

Each step of the large-step Monte Carlo begins with a 3-opt tour. This tour is then modified by making a double-bridge move, and it is then 3-opted. The intuition behind the "outer–loop optimization" is that if a link is sufficiently far away from either of the two bridges then it will not be involved in the 2 or 3-changes and so it need not be considered as a possible starting point for any 2 or 3-changes. We can therefore restrict the starting points of the outermost loop of the opt algorithm so as not to run over *all* the links, but just those which are close to the "action." In general, of course, the bridges rapidly get changed; what we really need to do is keep a list of the cities which are attached to links which have moved since the last 3-opt. Call this data structure *change_list*. The "consideration list" or *con_list* will be those cities which are sufficiently close to some member of *change_list*. The outermost loop of the opt algorithm will then run over only those links which are attached to a city which is a member of *con_list*.

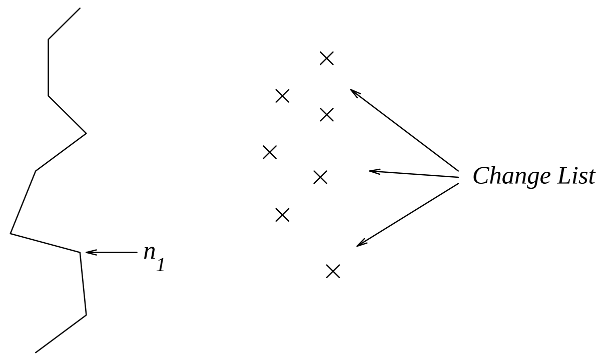

Figure 7: The city n_1 is far from the "action," is not part of the *con_list*, and thus need not be considered in the outermost loop of the 3-opt procedure.

A schematic diagram of the outer–loop optimization is given in figure 7. Drawn there is a potential starting point for the outermost loop of the 3-opt algorithm, n_1. Also shown is the set *change_list*, those cities which have had an attached link move since the last 3-opt. The "sufficiently close" constraint defines the set *con_list* and comes from the usual type of argument where we consider the minimum new distance versus the maximum old distance. The result is the following. The set *con_list* is given by those cities, n_1, for which the following inequality is true for some city, p, in the *change_list*:

$$d(n_1, p) + 2 \cdot min\_link < 2 \cdot max\_link + max\_p.$$

max_p stands for the maximum length of the two links attached to p. This result comes about because n_1 must interact with one of the cities in the *change_list* since it is already known to be relatively 3-opt to everything else. Therefore, if n_1 is to be involved in a 3-change, there must be a link going from n_1 to one of the members, p, of *change_list* and also one of the two links attached to p must be broken.

This optimization allows the Monte Carlo to concentrate on the current region of interest. Using it, sub-linear (in N) time complexity for the 3-opt is possible and this seems to be born out by our timings of the 3-opt (discussed in section 6). Another important point is that the *max_link* which appears in the above inequality need no longer be over the entire tour, but rather just over the set *con_list*. Since *max_link* is itself involved in the definition of *con_list*, one can recurse until nothing changes, thus achieving the smallest possible *max_link*. The fact that *max_link* no longer depends on the entire tour is satisfying. More complex and sharper inequalities than the above can be written down but at some point the computational time spent on finding *con_list* outweighs the potential benefit from a small *con_list*. We have not thoroughly explored all the possibilities.

Interaction with Branch and Bound

Non-exact algorithms such as the large-step Monte Carlo can be used to improve exact algorithms such as branch and bound, and vice versa. For a Euclidean TSP, many of the possible $N(N-1)/2$ links are long and thus unlikely to belong to the optimal tour. Branch and bound algorithms begin by eliminating links from consideration. For instance, for a random city $N = 400$ problem, typically 75% of the links are eliminated by the first pass of our branch and bound program [22]. Since we know these links cannot appear in the optimal tour, we can set the corresponding distances d_{ij} to infinity in the Monte Carlo, effectively removing them from consideration. In practical terms, this causes the bounds to saturate more rapidly and can speed up the 3-opt. Inversely, the Monte Carlo rapidly gives one very good tours. The best of these gives a sharp upper bound of use for the exact methods. Having a good bound leads to significant improvements in the pruning and hence the performance of the branch and bound algorithm and facet finding algorithms.

6 Results

This section contains the results of numerical experiments we have conducted using the large-step Monte Carlo.

Local Search Benchmarks

Our method consists of local-opt searches embedded within a Markov chain. Almost all of the computer time is spent within the local-opt and here we give

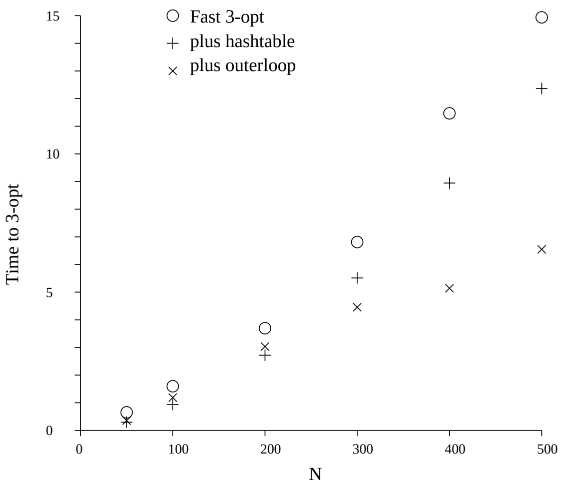

Figure 8: SPARCstation-1 timings to 3-opt a tour after a kick. For each N, the data has been averaged over 100 Markov chain steps. This was done for 20 separate TSP instances and then averaged again. The 3-opt is described in section 4, the two other optimizations in section 5.

some timings as a function of N. The runs were done on a SUN SPARCstation-1 computer and the code was written in the C programming language. Most of our runs attempted to solve the Euclidean version of the TSP, where the cities lie in a plane and the shortest, straight-line distance between city i and city j is taken for d_{ij}. However, the algorithm does not make use of this property, and the Euclidean version is not thought to be an easy subset of all TSPs.

To time the algorithm, we constructed 20 instances of random–scatter TSPs for several N values. The N values considered were $N = 50, 100, 200, 300, 400,$ and 500. For each instance we ran 100 steps of the Markov chain and computed the average time required to local-opt. These numbers were then averaged over the 20 instances. Figure 8 shows the result for the case of 3-opt. The set of points labeled "Fast 3-opt" were obtained using the basic 3-opt algorithm as described in section 4. Fitting these points to a power law gives an exponent near 1.5. Following the arguments given at the beginning of section 4, the algorithm should behave at worst as $d(N)N$ so that one expects an exponent less than 2. Also, the check-out time itself requires $f(N)N$ steps, so the exponent should also be greater than 1, as it is. The set of points labeled "plus hashtable" are runs which include the hashtable optimization discussed in section 5. The effect of the hashtable is not very large since only 100 tours have been constructed in each of these benchmark runs — the hashtable becomes important as one thoroughly samples the tours around the optimum. The last set of points includes, in addition, the outer–loop optimization. For large N the improvement is significant and seems to indicate a different N dependence, possibly sub–linear. The dependence on N when L-K is used instead of 3-opt is similar. Since the algorithm considers more exchanges, the main difference is that L-K is computationally more expensive, our L-K search taking about 1.8 times longer than a 3-opt search.

Performance of the Markov Chain Algorithm

Now we turn to the overall performance of the large–step Markov chain algorithm. We first focus on the ability of the method to solve problems to optimality. For $N = 50, 100$, and 200, we compared our results with exact solutions which were obtained using a branch and bound program written by one of us [22]. We generated instances of the TSP which consisted of cities scattered randomly throughout a square. It was observed that the large-step Monte Carlo was extremely effective. For N up to 200 (beyond $N = 200$ our branch and bound program was unable to converge) the Monte Carlo easily found the true optimum be it with the 3-opt or with the L-K local search. When L-K is used, the average time to solve to optimality is less than one minute for $N = 100$ and five minutes for $N = 200$. For 3-opt embedded in the Monte Carlo, a few minutes was usually sufficient at $N = 100$, while for $N = 200$ less than an hour was necessary. Note that the 3-opt time to solve to optimality is not merely a constant factor larger than the L-K times, instead it is N dependent. This is because there are exponentially more (in N) 3-opt tours than L-K opt tours.

We then ran tests on larger problems solved to optimality by other groups using cutting plane methods. The first instance is the LIN318 problem [9]. The problem is posed as an open tour with fixed ends, but it is easy to recast as a TSP by setting the length of the "link" between the two ends to some large and negative value. Padberg and Grötschel, using a combination of cutting-plane and branch-and-bound methods, were able to find the optimal tour for this problem [6]. The original [9] Lin–Kernighan heuristic using repeated random starts achieved an answer of 41,871, which is 1.3% above the optimal value of 41,345 (see [6] for a discussion as to exactly what rounding strategy was used when constructing the distance matrix). We have confirmed this behavior with our coding of L-K. Simulated annealing achieves a similar result to Lin-Kernighan for this problem [23].

Consider now our large step Markov chain approach. We made many separate attempts on the LIN318 problem with different random starts. When the temperature is zero, the tour gets stuck in local minima, though these are of high quality. This is easily understood by comparing these local opt tours to the exact solution: the links connecting the three "columns" are not easily moved. They can be shifted by applying a bridge move, but unless the other bridge is at the "right" place, the move is rejected. Thus we found it necessary to either use multiple starts or to use non-zero temperatures. After adjusting the temperature, the large-step Monte Carlo consistently found the optimal tour of length 41,345 [6]. This fact gives us confidence in the robustness of the procedure. Another indicator is the fact that the Monte Carlo visits large numbers of tours just above the optimum and in fact, visits them multiple times. For instance, it visits the tour of length 41,349 given in [24] and the evidence from the Monte Carlo is that this is the first sub-optimal tour; the Monte Carlo found no tours between 41,345 and 41,349. In terms of speed, when the local search was L-K, the average time to find the optimum was less than one hour of SPARCstation-1 time. When the local search was 3-opt, the average run time was four times longer.

Run	CPU Hours	Best Tour	% above min
1	100	27,693	0.025%
2	15	27,705	0.069%
3	15	27,706	0.072%
4	30	27,697	0.040%
5	30	27,686	0.000%

Table 3: SPARCstation-1 CPU times and best tour lengths for our 4 longest runs with 3-opt local search. A post-reduction method as described in the text was applied to the 4 best tours and the result of this is run number 5.

We also tackled the so-called AT&T532, a 532 city problem solved to optimality by Padberg and Rinaldi [7] using branch-and-cut methods. They determined the exact optimum to be of length 27,686. The runs which used L-K for the local search always found the optimum, and the average time to solve to optimality was three SPARCstation-1 hours. If one uses instead 3-opt for the local search, the optimum is much more difficult to obtain In Table 3 we present the results of our four longest runs from different random starts for this case. The best tour length from a random start is 27,693, i.e., 0.025% above the optimum. (Note that the average intercity length in these units is 55.) The average of the best length of these runs is 27,700. Thus the Markov chain provides very high quality tours. In order to push the 3-opt Monte Carlo to the limit, we then used a method inspired by the "reduction" procedure of Lin and Kernighan [8, 9]. In our "post-reduction," we took the best tour from each of our four long runs and created a list of cities which have the same incoming and outgoing links in this set of tours. The Monte Carlo was then run starting with tour 27,693 and with the extra constraint that the bridges used in the kick cannot connect to these cities. The number of cities to which the bridges could connect was less than 100, leading to a much smaller space to sample. This allowed the post-reduction run to find the exact optimum of length 27,686. Note that the constraint imposed by this procedure is not very "hard" as the constraint is used only for the location of the bridges, not in the subsequent 3-opt.

To see how ordinary Lin-Kernighan repeated from random starts performs on this problem, we first use the data of Bentley and Johnson (private communication) who have written a fast L-K code. For the AT&T532 problem, their local search from a random start takes on the order of 400 seconds on a VAX/750, and leads to an average excess length of 1.1%. For 100 random starts, their best tour length is 27,797, or 0.4% above the optimum. The probability distribution of tour lengths given by their L-K has an average of 28,500 and a variance of 400. Our coding of L-K gives very similar results.

Finally, we also considered a 783 city problem (RAT783) solved to optimality by plane cutting methods by Cook et. al. [25]. The city positions were obtained by small, random displacements from a regular 27×29 lattice [26]. This instance turned out to be very easy to solve using our Monte Carlo heuristic. The runs reached the optimal tour (of length 8,806) in an average of one hour of SPARCstation-1 time (using the L-K local search). This timing is similar to one for the LIN318 instance. It is clear that the difficulty of a problem is not given only by it's size.

To really compare the stochastic and local search methods requires fixing the total amount of CPU time allowed. For very long runs, a Markov chain approach leads to better results than random sampling because the density of states is simply too small near the optimum, so that random sampling is not competitive. On the other hand, local search methods are much better than for instance simulated annealing for short runs. A big advantage of our method is that a local search is incorporated into the algorithm. This allows our method to be better than simulated annealing and local searches for both short and long runs.

7 Conclusions

Many heuristic methods have been proposed for the TSP. To date, the most effective of these are the local search and the stochastic sampling algorithms. In this paper, we have shown that it is possible to combine these methods into what we call a large-step Markov chain. In this way, only locally optimal tours are sampled, thus reducing the search space dramatically. In addition, to sample this space effectively, a special kind of 4-change is made (the "kick") followed by the local search. For other optimization problems, (graph partitioning, spin glasses, etc...), the "kick" should correspond to a change which is not easily accessible to the local search moves and which is thought to be relevant.

We have implemented this algorithm and applied it to a number of large

TSP instances. In particular, it is able to find the exact solution to the LIN318, the AT&T532, and the RAT783 problems in a very modest amount of CPU time. Our method provides a substantial improvement over the hereto state-of-the-art algorithm of Lin and Kernighan. We also showed how the local searches can be accelerated, providing a method for doing 3-opt which makes the $O(N)$ complexity of the check-out time explicit, and which can be extended to k-connected-opt. Also, the set of links to be considered in the tour improvement can be drastically reduced by a dynamic outer-loop optimization.

Appendix A: Density of States

For an N city problem, there are $(N-1)!/2$ tours if one uses orientation and cyclic permutation symmetry. It is of interest to know the length distribution of these tours. For clarity, we consider random scatter problems and scale the d_{ij} so that the average distance between neighboring cities is N independent. In two dimensions, this is achieved by having the cities in a region whose length scales as $\sqrt{N}$. We will consider a square $\sqrt{N}$ on a side, but for other shapes the argument still holds. In these units, the tour lengths scale as N since one can crudely think of a $4N$ problem as being four problems of size N pasted together. For any particular instance, the minimum tour length is

$$l_{min} = c_0 N + x\sqrt{N}$$

where c_0 is independent of both the particular instance of the TSP and the shape chosen above, and x is a number which depends on the instance. We can think of x as a random variable describing the fluctuations in the length as we consider different instances of the TSP.

Now we would like to estimate the density of states near the minimum, i.e., the number of tours of length between l and $l + dl$, divided by dl. The main question is: how fast does this grow with l and with N? The answer is not known, but several distributions have been suggested [27, 28]. Here, we present a model which is very simple, but which seems to describe well the data for randomly scattered Euclidean TSP's.

Let us first consider the set of all tours. The $(N-1)!/2$ tours have lengths which vary between l_{min} and $l_{min} \cdot O(\sqrt{N})$, so clearly the range is very broad. Most of these tours are of no interest, for instance they have very long links of length $\sim \sqrt{N}$. To model the density of tours near the minimum, we start with the optimum tour and consider doing 2, 3, etc... changes. We will restrict ourselves to localized k-changes where the k links occur in a small spatial region, i.e., are near one-another. Specifically, consider only connected k-changes, for any k. Each connected k-change increases the tour length by an amount which we take to be a random variable with an N-independent probability distribution. To keep the modeling of the density of states simple, we replace the random variables by a typical value f which is some fraction of the inter-city nearest-neighbor distance (and thus is $O(1)$). When constructing the set of low-lying tours, we first perform a k_1-change at some location, then follow this by a k_2-change at another location, and so forth. The $\sqrt{N} \times \sqrt{N}$ square is approximated by M independent patches, where $N/M = a$ (the area of the patch) is taken to be N independent. A k-change as considered above is then viewed as increasing the length of the tour in one of these patches. Each patch is considered to be "on" or "off", and when one is "on" the tour length is increased by f. The total number of tours in this model is 2^M. To get the density of tour lengths, simply count the number of ways to choose a fixed number of patches. This gives a binomial distribution, so that $\binom{M}{p}$ tours have length $l_{min} + p \cdot f$. The above modeling has introduced two scale parameters, f and a, which are required to be N independent. If the model were extended so that k-changes for different k's were treated separately, a multinomial would have resulted but this would not affect the resulting distribution much. We have empirically found that good fits are obtained to the numerical data extracted from instances with randomly distributed cities. We expect the model to be reliable when $p >> 1$ so that the instance fluctuations are unimportant. Also, when too many patches are "on" one does not expect them to stay independent, so the model is innaccurate as p approaches $M/2$. Another effect is due to the fact that we threw away many of the longer tours by the connectivity constraint on the k-changes. This leads to the wrong scaling in the total number of tours, $2^{N/a}$, whereas the correct result is, of course, $(N-1)!/2$.

The model as described above was for the density of all tours. We can also specialize to the case of the density of k-opt tours and it is here that we expect the model to be most accurate. The dilute gas picture now consists of patches of area a_k. As before, our prediction for the density of tours at $l_{min} + pf_k$ is $\binom{M}{p}$, where $N = Ma_k$. The curve in figure 1 shows this prediction for the density of 3-opt tours for a particular 100-city TSP. M (and therefore a_3) was fixed by the requirement that the total number of 3-opt tours be 2^M. f_3 was adjusted to give the correct average tour length. It is important to note that f_3 is the only free parameter. The fit to the data, as seen in figure 1, is quite good.

As we consider higher and higher connected k-opt, i.e., as the local search is improved, we expect a_k to increase since a larger number of link changes is required to go from one k-opt tour to another (at least $k+1$). The total number of k-opt tours is given in this model by $2^{N/a_k}$, a parameterization which has been given previously by Lin [8]. It would be interesting to determine whether the inclusion of disconnected graphs (say all the 4-opt tours) leads to a different N dependence such as $2^{N^\alpha/a_k}$. Note that this model is valid also for higher dimensional randomly scattered Euclidean TSPs, and for other local searches besides k-opt. For instance, it applies to the Lin-Kernighan density of states. The model predicts that any local search method gives a distribution for the relative excess length $(l - l_{min})/l_{min}$ which has an N independent average, and a width which scales as $N^{-1/2}$. A consequence of this is that when comparing at large N different local search algorithms which use random starts, the only important characteristic is the average relative tour length obtained by the algorithm.

Appendix B: Markov Chains

This appendix reviews some general properties of Markov chains in order to answer the questions:

— will an algorithm always find the optimal solution?

— how long should one run if one wants to reach the optimal solution with a given confidence level?

The first question is related to the ergodicity of the Markov chain; the second, to its auto-correlation time.

Since the set of all tours in the TSP is a finite set, the Markov chain can be characterized by a transition matrix T. The matrix element T_{mn} is the probability to go from tour n to m. In practice, the selection of m requires random numbers. Given a starting tour, the application of T produces a sequence, or chain of tours. After some transients, usually the "memory" of the starting point decays, and tours appear with a limiting probability distribution P. P depends on the matrix T, and the goal is to find T's which lead to $P(C)$ large for tours C of short length. This is called biased sampling, and it leads to sampling the tours of interest more efficiently. The simplest way to create such a biased sampling is to use a Metropolis style algorithm: the current tour is changed in some small random way, and the change is accepted with high probability only if the new tour is shorter than the old one.

If $\bar{C}$ is the optimal tour, is $P(\bar{C}) \neq 0$? In the case of simulated annealing, the distribution P is known because T satisfies detailed balance. In particular, the probability of all tours is non-zero (the Markov chain is ergodic) and $P(C)$ depends only on the length of C. For general Markov chains, (i.e., for general choice of the matrix T), very little can be said of the probability distribution P. It is plausible nevertheless that within local-opt tours, our large-step Markov chain is ergodic, and all our runs are consistent with this. In particular, we have checked that $P(\bar{C}) \neq 0$ for many instances (see section 6).

How many tours, M, must one sample in order to have a high probability of reaching $\bar{C}$? There are two constraints here: first, one must have $MP(\bar{C}) \gg 1$, corresponding to the expected number of visits to $\bar{C}$ being much greater than 1. It is then improbable to have 0 visits. Second, M should be large enough so that the probability distribution of tours is indeed given by P: the above mentioned transients must have died away. This decay time can be made quantitative by the introduction of the auto-correlation time τ of the Markov chain. τ is defined by

$$e^{-1/\tau} = |\lambda_1|$$

where λ_1 is the eigenvalue of T of largest modulus and which is different from one. τ can be thought of as the longest characteristic time occurring in the dynamics generated by T. The second constraint now reads $M \gg \tau$. Note that there is not much point in working so hard as to find a T such that τ is as small as N (the number of cities) because the first condition also has to be satisfied: if one takes the analogue of "temperature" to be high (small bias), then τ is $O(N)$ (for modifying every link). Thus one is almost sure to do better than local search with random starts by simply embedding the local search into a Markov chains and introducing some bias into the sampling. On the other hand, one must make sure that τ does not get astronomically large. When simulated annealing is used for the TSP, τ diverges fast as the temperature is lowered because barriers become overwhelming. (Some of these barriers can be visualized by the transformations induced by double bridges.) But if the temperature is not low, there are too many configurations to sample, so again the algorithm is not effective for large N. Thus it is imperative to use large step Markov chains to keep τ from growing too fast as one increases the bias.

In practice, τ can be measured without having to determine the eigenvalues of a large matrix. Ideally, one should find some operator on configuration space which projects out as much as possible the eigenvector corresponding to the eigenvalue λ_1, though in practice this is difficult. We suggest for the TSP taking the operator D which counts the number of links a tour has in common with a given good tour. Then the observable (C_n means the nth configuration in the Markov chain)

$$\langle DC_{n+p}\ DC_n \rangle - \langle DC_n \rangle \cdot \langle DC_n \rangle$$

is proportional to ${\lambda_1}^p = e^{-p/\tau}$, for large p.

Acknowledgements

We would like to thank Bill Cook and David Johnson for useful discussions and for sharing some of their results with us prior to publication. We thank Manfred Padberg for providing us with the AT&T532 instance, Jean Vannimenus for bringing our attention to the work of reference [19], and Richard Friedberg and Paul Rujan for a number of suggestions. The work of O.M. was supported in part by a grant from the City University of New York PSC-CUNY Research Award Program and by NSF-ECS-8909127. The work of S.O. was supported in part by DARPA grant MDA972-88-J-1004, and, that of S.O. and E.F. in part by DOE-FG03-85ER25009.

References

[1] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, NJ, 1982.

[2] M. Held and R.M. Karp. The traveling salesman problem and minimum spanning trees, part I. *Oper. Res.*, 18:1138–62, 1970.

[3] M. Held and R.M. Karp. The traveling salesman problem and minimum spanning trees, part II. *Math. Prog.*, 1:6–25, 1971.

[4] E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys, editors. *The Traveling Salesman Problem.* John Wiley & and Sons, 1984.

[5] M. Grotschel and M. W. Padberg. Polyhedral theory. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem*, chapter 8. John Wiley & and Sons, 1984.

[6] P. W. Padberg and M. Grotschel. Polyhedral computations. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem*, chapter 9. John Wiley & and Sons, 1984.

[7] M.W. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Oper. Res. Lett.*, 6(1):1–7, 1987.

[8] S. Lin. Computer solutions of the traveling salesman problem. *Bell Syst. Tech. J.*, 44:2245, 1965.

[9] S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Oper. Res.*, 21:498, 1973.

[10] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671, 1983.

[11] V. Cerny. Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *J. of Opt. Theo. and Appl.*, 45:41, 1985.

[12] J. Bentley and D. Johnson. A comparison of heuristics for the traveling salesman problem. In preparation.

[13] J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[14] H. Muhlenbein, M. Georges-Schleuter, and O. Kramer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65, 1988.

[15] P. Rujan. Searching for optimal configurations by simulated tunneling. *Z. Phys.*, B 73:391, 1988.

[16] J. Hopfield and D. Tank. Neural computation of decisions in optimization problems. *Biol. Cybern.*, 52:141, 1985.

[17] G. Wilson and G. Pawley. On the stability of the travelling salesman problem algorithm of Hopfield and Tank. *Biol. Cybern.*, 58:63, 1988.

[18] D.M. Ceperly and M.H. Kalos. In K. Binder, editor, *Monte Carlo Methods in Statistical Mechanics*. Springer-Verlag, 1979.

[19] Z. Li and H.A. Scheraga. Monte carlo-minimization approach to the multiple-minima problem in protein folding. *Proc. Natl. Acad. Sci.*, 84:6611–6615, October 1987.

[20] W.R. Stewart. Accelerated branch exchange heuristics for symmetric traveling salesman problems. *Networks*, 17:423, 1987.

[21] D. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, MA, 1973.

[22] E.W. Felten. Best–first branch–and–bound on a hypercube. In *Third Conference on Hypercube Concurrent Computers and Applications*, 1988.

[23] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part III (the TSP). In preparation.

[24] M.W. Padberg and S. Hong. On the symmetric travelling salesman problem: A computational study. *Math. Prog. Stud.*, 12:78, 1980.

[25] W. Cook, V. Chvatal, and D. Applegate. In R. Bixby, editor, *TSP 90*. Workshop held at Rice University, April 1990.

[26] A Library of TSP instances is electronically available. For further details, contact R. Bixby at `bixby@rice.edu`.

[27] B.L. Golden and W.R. Stewart. Empirical analysis of heuristics. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem*. John Wiley & and Sons, 1984.

[28] N. Burgess and M. A. Moore. Cost distributions in large combinatorial optimization problems. Technical report, Manchester University, Manchester, U.K., 1989.