Combining Simulated Annealing with Local Search Heuristics *

Olivier C. Martin
Division de Physique Théorique†, Institut de Physique Nucléaire,
Orsay CEDEX 91406 France
martino@ipno.in2p3.fr
and
Steve W. Otto
Department of Computer Science and Engineering
Oregon Graduate Institute of Science & Technology
20000 NW Walker Rd, PO Box 91000
Portland, Oregon, USA 97291-1000
otto@cse.ogi.edu

Abstract

We introduce a meta-heuristic to combine simulated annealing with local search methods for CO problems. This new class of Markov chains leads to significantly more powerful optimization methods than either simulated annealing or local search. The main idea is to embed deterministic local search techniques into simulated annealing so that the chain explores only local optima. It makes large, global changes, even at low temperatures, thus overcoming large barriers in configuration space. We have tested this meta-heuristic for the traveling salesman and graph partitioning problems. Tests on instances from public libraries and random ensembles quantify the power of the method. Our algorithm is able to solve large instances to optimality, improving upon local search methods very significantly. For the traveling salesman problem with randomly distributed cities in a square, the procedure improves on 3-opt by 1.6%, and on Lin-Kernighan local search by 1.3%. For the partitioning of sparse random graphs of average degree equal to 5, the improvement over Kernighan-Lin local search is 8.9%. For both CO problems, we obtain new best heuristics.

1 Introduction

In many science and engineering problems, one must find the minimum of a function of many variables, hereafter called the cost function, where the arguments may be subject to specified constraints. For some problems, there exist very efficient algorithms such as linear programming for obtaining the optimal solution; however, for many combinatorial optimization (CO) problems,

*Appeared in the book *Metaheuristics in Combinatorial Optimization*, edited by G. Laporte and I. Osman., Annals of Operations Research vol 63 pp57–75 (1996)

†Unité de Recherche des Universités Paris XI et Paris VI associée au C.N.R.S.

no such efficient algorithms are known. Then it may be necessary to use heuristic algorithms: perhaps none of the exact algorithms can be used because of one's computational limitations; or simply, a good upper bound on the optimum may be adequate and obtaining a good sub-optimal feasible solution is enough.

For these "hard" problems, the best heuristics are often "simulated annealing" and "local search" approaches. We have combined these two families of heuristic methods into one, arriving at a much more powerful class of algorithms that we call "Chained Local Optimization". This meta-heuristic is very general since simulated annealing and local search are viable techniques for most CO problems. It is also flexible, enabling the incorporation of problem specific aspects of the CO problem. We have implemented the method [1, 2, 3] for two graph-based problems, the traveling salesman and the graph partitioning problems (TSP and GPP). The results are significant performance gains for both the TSP and the GPP. Our method improves the state of the art TSP and GPP local search heuristics, leading to new "champion" heuristics. Furthermore, the algorithms can be and have been implemented efficiently in parallel.

In section 2 we review the status of heuristics for the TSP and the GPP. Section 3 shows how it is possible to combine simulated annealing and local searches. In practice, it is necessary to adapt our meta-heuristic to the CO problem of interest; some of the problem specific aspects are illustrated in section 4 in the case of the TSP and the GPP. Sections 5 and 6 summarize the performances obtained. Section 7 explains how the algorithm was implemented in parallel. Finally, definitions of the TSP and the GPP, along with specific details concerning the local searches, can be found in two appendices.

2 Status of TSP and GPP heuristics

Traveling salesman heuristics

The important exact algorithms for TSP are branch and bound methods, and more recently branch and cut methods [4]. These methods have progressed tremendously in the last ten years, so that instances with N of several thousand have now been solved to optimality [5]. A library of solved instances is available electronically [6], enabling users to test their algorithms.

In terms of heuristics, many methods have been proposed, such as direct tour construction, local search [7, 8], simulated annealing [9, 10], genetic algorithms [11], and neural network approaches [12]. The present consensus [13] is that the heuristic that leads to the best solutions is a local search method due to Lin and Kernighan [8] (L-K). L-K is essentially a breadth-first search (3-opt) followed by a depth-first search (using greedy 2-changes). It is the benchmark against which all heuristics are tested. Simulated annealing (S-A) also gives very good tours, but it is many times slower than L-K [13].

Graph partitioning heuristics

Exact methods for GPP are not as highly developed as for the TSP, and only recently has there been an efficient integer linear programming approach [14]. Numerous heuristic methods have been proposed, ranging from the general purpose simulated annealing [9] approach to methods such as the recursive spectral bisection [15] and compaction methods [16] that are best adapted to graphs that have a built-in geometric structure. For generic (random) graphs, the consensus

[17] is that the "best" heuristics are simulated annealing [9] and a variable depth search due to Kernighan and Lin [18], hereafter referred to as K-L.

Discussion

For most CO problems, and certainly for the TSP and the GPP, as the characteristic size N of the instance grows, the number of configurations (feasible solutions) that are locally optimal under a given local search method grows very quickly with N. We offer the following argument as a way of understanding the relation of Markov-chain-based methods to re-start (from random configuration) methods such as L-K. This argument is not a proof of anything, but gives some intuition.

For many instance ensembles, the distribution of costs (per city or per vertex) found by local search (from a random start) is typically a Gaussian of decreasing width as N becomes very large. This means that multiple tries of algorithms such as L-K and K-L are less and less effective in improving the best found solution, as N grows. If lower cost solutions are truly desirable, it is necessary to improve the *average* (e.g., the per city average) performance of the heuristic. There are two natural ways to do this. First, one can try to extend the neighborhood that the local search considers, just as L-K extends the neighborhood used in Lin's local searches. Second, instead of sampling the locally optimal configurations in a random way as is done by applying the local searches from random starts, it might be possible to sample locally optimal configurations in a more efficient way. An example is to sample the configurations along a Markov chain with a bias in favor of the lower cost configurations. This type of sampling gives power to simulated annealing: in a long run, one improves an already very good solution, one that probably has many features in common with the exact optimum. The standard L-K and K-L algorithms, on the contrary, continually restart from scratch, throwing away possibly useful information. Fortunately, it is possible to combine the good features of simulated annealing and of local search to get the best of both worlds.

3 Chained Local Optimization (C-L-O)

Simulated annealing does not take advantage of local search heuristics, so that instead of sampling locally optimal configurations as does L-K or K-L, the Markov chain samples *all* configurations. The heart of our meta-heuristic comes from the realization that it would be a great advantage to restrict the sampling of the Markov chain to the locally optimal configurations only. Then the bias that the Markov chain provides would enable one to sample the best locally optimal configurations more efficiently than local search repeated from random starts. To do this, one has to find a way to go from one locally optimal configuration to another. This is problem specific and will be discussed in the next section. For the moment, we give a generic, problem-independent, viewpoint.

The algorithm proceeds as follows. Suppose the configuration is currently locally optimal (according to some local search algorithm). This is labeled *Start* in figure 1. Now apply a perturbation or "kick" to this configuration so as to significantly change *Start*. After the kick, we reach the configuration labeled *Intermediate* in the figure. Standard simulated annealing would impose the accept / reject procedure directly to *Intermediate*. Instead, we notice that it is much better to first improve *Intermediate* by a local search and apply the accept / reject test

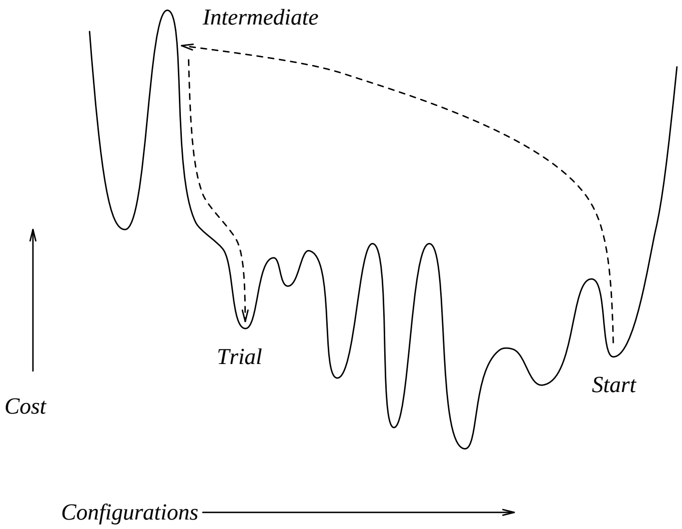

Figure 1: Schematic representation of the objective function and of the configuration modification procedure used in chained local optimization.

only afterwards. The local search takes us from *Intermediate* to the configuration labeled *Trial* in figure 1. Now apply the accept / reject test. If *Trial* is accepted, one has managed to find an interesting large change to *Start*. If *Trial* is rejected, return to *Start*.

The iteration of this procedure is what we call chained local optimization (C-L-O). It can be thought of as a large-step Markov chain generalization of simulated annealing. The approach allows one to do much better than simulated annealing — as shown in figure 1, the accept / reject step is only applied *after* the configuration is returned to a local minimum. Many of the barriers (the "ridges") of the cost landscape are jumped over in one step by the algorithm. Effectively, these barriers are smoothed or eliminated from the landscape. Simulated annealing, by contrast, must climb over each of these ridges in a series of small steps, passing the accept / reject test many times.

To implement the above methodology for an arbitrary problem requires two things: a good local search method and a choice for the kick that is appropriate to the specific CO problem. The first requirement is usually met by one of the widely used local search algorithms for the problem of interest. For the TSP, we have chosen to use the Lin-Kernighan local search, and for the GPP, the Kernighan-Lin local search. Both are the best general local search methods for those problems. The second requirement, a choice of kick, should be adapted to both the CO problem and to the local search method used. For the TSP, we have used a special 4-change move because it is a topologically important modification that is missing in the L-K local search; for the GPP, we do a large k-exchange using a semi-greedy procedure and a connectivity constraint. Details are given in the next section. With these choices, the methodology gives rise to major improvements over both simulated annealing and local search methods, leading to very powerful heuristics for the TSP and the GPP. Comparable improvements should occur for other CO problems as long as the biased sampling of the Markov chain is more efficient than random sampling.

Weaker forms of this meta-heuristic have been proposed. In [19], Li and Scheraga implemented a Markov chain with a kick followed by a downhill search for a protein folding problem. The algorithm did lead to some cost improvement, but the authors did not consider it a general optimization method. Also, contrary to their claim, the algorithm does not satisfy the symmetry constraint of detailed balance (see [1]), so the method is not a simulation of an annealing process and a Boltzmann distribution ($e^{-E/T}$) of solutions is not obtained. Baum ([20] and unpublished) introduced the meta-heuristic as an optimization method applicable to CO problems exhibiting "ultrametric" structure. He called his method iterated descent. Unfortunately, his choice of local search and kick were inadequate, so he was not able to give compelling evidence that the meta-heuristic was powerful. In a different spirit, a number of authors have embedded local searches inside genetic algorithms (GA). In the earliest such work of which we are aware [11], Muhlenbein et. al. used 2–opt to improve children configurations in a TSP before evaluating the cost function. This enabled them to partly overcome the difficulty of combining two parents to create a child of good quality. In [1], we show how the use of many parents can alleviate this difficulty. The method was called a "post reduction procedure" because it is not totally faithful to the GA approach. Finally, note that the C-L-O algorithm can be thought of as a parthenogenetic algorithm: reproduction is done with a single parent. This restriction makes it much easier to create good children. One can think of C-L-O, and especially parallel C-L-O (Section 7), as a type of GA algorithm.

4 Adapting the kick to a specific CO problem

We begin with the TSP where the choice of a kick is clear-cut, elegant, and effective. Suppose for simplicity that the local search embedded in C-L-O is 3–opt. If a kick consisting of a 3–change is used, the 3–opt search will usually bring us back to the previous tour with no change. Thus it is probably a good idea to go to at least a 4–change for the kick when the local search is 3–opt. For other local search algorithms, a good choice for the kick would be a k–change that does not occur in the local search. Surprisingly, it turns out that 2–opt, 3–opt, and especially L–K are structured so that there is one kick choice that is natural for all of them. To see this, it is useful to go back to the paper by Lin and Kernighan where they define *sequential* changes. These are changes that keep the tour connected during intermediate steps. It can be shown that the check–out time for sequential k–changes can be completed in $O(N)$ steps. All 2 and 3 changes are sequential, and the first non–sequential change occurs at k=4. We call it a *double–bridge* change because of what it does to the tour (see figure 2). It can be constructed by first doing a 2–change that disconnects the tour; the second 2–change must then reconnect the two parts. The double–bridge change is the only 4–change that cannot be obtained by composing changes that are both sequential and leave the tour connected. The motivation for this kick is evident from figure 2: it allows a peninsula to hop from one place in the tour to another without much of an increase in the tour length. The double bridges can be generated randomly, or with some bias towards allowing only nearby peninsulas to hop.

Note that if one includes this double–bridge change in the definition of the neighborhood for a local search, check–out time requires $O(N^2)$ steps (essentially a factor N for each bridge). Rather than considering these changes as part of the local search, we include such changes stochastically through the kick, keeping a fast algorithm. Thus we can conclude that an analysis of the changes used in local searches leads to a natural candidate for the kick.

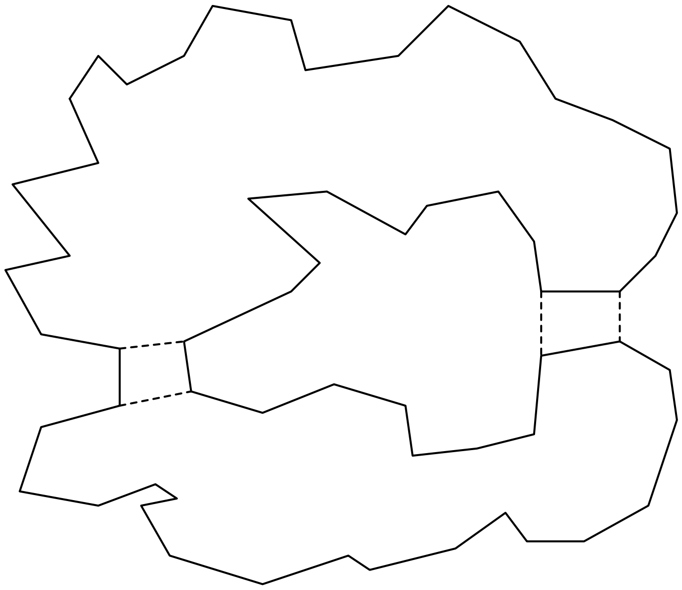

Figure 2: Example of a double-bridge kick (shown in dashed lines). The bridges rearrange the connectivity of the tour on large scales.

We have been not able to obtain such an elegant solution in the case of the GPP, but nevertheless the kick procedure leads to an effective algorithm. Let us first motivate the choice of kick appropriate for geometric graphs (see section 6). Upon visualizing the partitions obtained by K-L using random starts, one sees immediately that K-L generates partitions with many "islands": the subsets A and B usually end up being highly disconnected, the partition is fragmented. This suggests using a kick that exchanges vertices between these islands, and motivates the following procedure for generating a kick. First, in each subset A and B, randomly choose a vertex on a cut edge. These two vertices will be the "seeds". Let X and Y be the set of vertices in A and B that are going to be exchanged by the kick. The sets X and Y are generated by growing a cluster around each seed: one adds sequentially to each cluster vertices that belong to the "other" subset but which are nearest-neighbors of the current cluster. The size of X and Y is chosen randomly ahead of time, but if one cluster can no longer grow (as happens when the seed is in an island), then the cluster growth is stopped and one takes that as the kick. It turns out that this choice of kick is very effective, not only for geometric graphs, but also for random graphs as discussed in section 6.

5 Results for the TSP

We originally tested [1, 2] C-L-O on randomly generated instances with N points in a unit square. For N up to 200, we were able to determine the optimum tour using a branch and bound program. Then we ran chained local optimization using Lin-Kernighan as the embedded local search. In all cases, the optimum tour was found rather quickly. (The average time for finding the optimum was less than one minute on a Sun-SPARCstation for $N \leq 200$.) Then we considered much larger instances where (our) exact method no longer found the optimum. For those instances, we compare C-L-O with local search methods repeated from random starts. We find that chained local optimization improves 3-opt by over 1.6%, and improves L-K by 1.3%. Just how far C-L-O (with L-K) is from finding the true optimum is subject to debate, but the Held-Karp lower bound shows that the average excess length is at most 0.84% (see [13] and Table 1).

Finally, we tested the C-L-O algorithm on large, specific instances solved to optimality by other groups and available [6]. These instances were: (1) LIN-318 [21]; (2) AT&T-532 [22]; and (3) RAT-783 [23]. The numbers denote the number of cities, and the references give the authors who first solved the problem to optimality using branch and cut methods. We found that C-L-O was able to find the optimum solution to LIN-318 in minutes, and the solution to AT&T-532 and RAT-783 in an hour on a Sun-SPARCstation. It is the only heuristic able to find the optimum for these problems.

Comparison of Methods for TSP, N=1,000

Sim. Annealing		Lin-Kernighan		C-L-O	
Excess	Minutes	Excess	Minutes	Excess	Minutes
1.71	791	1.75	30	1.72	10
1.56	1582	1.56	147	1.54	17
1.42	3952	1.43	585	1.42	30
1.33	7903	1.34	1461		
1.27	15806	1.29	2921		
1.22	39513	1.23	5831	1.20	67
		1.18	14600		
				1.07	125
				0.98	237
				0.88	561
				0.86	1086
				0.84	2172

Table 1: A comparison of simulated annealing, Lin-Kernighan from random start, and C-L-O for an $N = 1,000$ TSP (random cities, from [13]). The columns labeled "Excess" give the percentage excess above the Held-Karp lower bound, the columns labeled "Minutes" give running times in minutes. Comparable excesses were placed on the same rows; this caused blank entries in some cases. As can be seen, C-L-O (called "iterated Lin-Kernighan" in the reference) out-performs the other algorithms by a wide margin. C-L-O runs were with temperature set to zero.

One of the most interesting results of the simulations is that for "moderate" sized problems (such as the AT&T 532 or the 783 instances mentioned above), no "annealing" seems to be necessary. It is observed that just setting the temperature to zero (no uphill moves at all) gives an algorithm that can often find the exact optimum. The implication is that, for C-L-O, the effective energy landscape has only one (or just a few) local minima! Almost all of the local minima have been modified to saddle points by the extended neighborhood structure of the algorithm.

A comparison of simulated annealing, Lin-Kernighan from random start, and C-L-O was done in [13], for an $N = 1,000$ instance of cities scattered randomly in the square. The results are reproduced in Table 1.

Table 2 gives results for our C-L-O algorithm applied to 10 instances of $N = 10,000$ TSP, with the cities distributed randomly in the square. The average length per step, L^*, scales as $L^* = \alpha\sqrt{N}$ asymptotically, and the value of the constant, α, is of interest. The results range from an estimate for α of $.7300 \pm .0007$ coming from the initial Lin-Kernighan application (taking about 11 minutes each), to an estimate of $.7235 \pm .0006$ from 100 C-L-O steps (taking about 69 minutes), to $.7208 \pm .0006$ from 5,000 C-L-O steps (taking about 276 minutes).

C-L-O Results over 10 Instances of TSP, N=10,000

	$L^*/\sqrt{N}$	$\pm$	Time (sec)	$\pm$
Initial L-K	.7300	.0007	695	39
After 100 steps	.7283	.0007	1037	42
After 1000 steps	.7235	.0006	4115	96
After 2000 steps	.7220	.0006	7265	192
After 3000 steps	.7214	.0006	10385	298
After 5000 steps	.7208	.0006	16545	463

Table 2: Results for C-L-O on 10 instances of $N = 10,000$ TSP, with cities distributed randomly in the square. The second column gives an estimate of the parameter α and the fourth column is the time for the run, in seconds, on an HP-720 workstation (similar to a Sun SPARC-2 in speed). The columns labeled $\pm$ are the expected errors of the given means. C-L-O runs were with temperature set to zero.

6 Results for the GPP

In [3], we compare C-L-O to K-L and to improvements to K-L. K-L is known to be significantly better than simulated annealing for certain types of sparse graphs of relevance to load-balancing, while being less good for random graphs [17]. Here we give the performances of our algorithm for "geometric" graphs and for sparse random graphs. In [3], we also compare with graphs obtained from real world load balancing instances. Again, the C-L-O algorithm improves significantly on local search.

Performance on geometric graphs

This ensemble of graphs is motivated by load balancing problems. To construct a "geometric" graph, vertices are placed at random inside the unit square; two vertices are connected if and only if they are at a distance $\leq R$. (See figure 3 for an example). As R increases, the connectivity as measured by d, the average degree of a vertex, increases. Neglecting boundary effects, one has $d = \pi R^2 N$. Johnson et. al. [17] did a thorough comparison of local search, K-L, and simulated annealing for these types of graphs. They found that a certain improvement to K-L, which they called Line-K-L (L-K-L), in which one starts K-L on a non-random partition, was by far the best method.

We have compared the performance of L-K-L and C-L-O. For conciseness, we consider only the results for runs where T was set to 0. For our benchmarks, five geometric graphs were

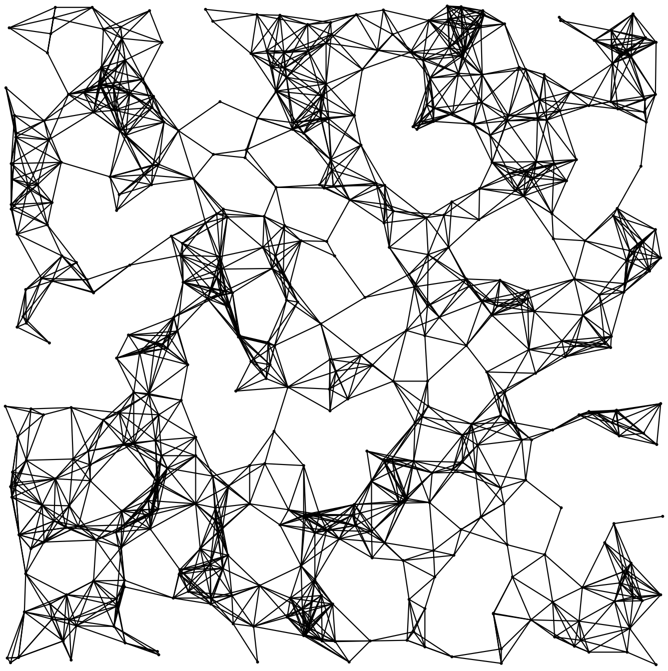

Figure 3: A geometric graph with $N = 500$ vertices and $d = 10$.

randomly generated with $d = 6$ for each value $N = 100, 250, 500$, and 1000. The results for the latter three graphs are given in Table 3. For each graph, we ran L-K-L 2000 times, and did 20 C-L-O runs, each one consisting of 100 kicks followed by a K-L. For "small" geometric graphs, ($N = 100$), both algorithms quickly found the same best solution, so it is likely that the exact optimum is obtained for such small values of N. As N increased, C-L-O rapidly improved over L-K-L. Call "best ever" the best solution found by any method. For $N = 250$, 2000 L-K-Ls were not enough to find the best ever in 3 of the 5 graphs, and L-K-L never found the best ever for the $N = 500$ and 1000 graphs. C-L-O on the other hand always found the best ever among its 20 runs for each graph. One can also compare the average performances, taking into account the different speeds of the algorithms. One run of 100 steps of C-L-O takes about the same computation time as 100 L-K-Ls. Thus from the 2000 L-K-L data points, we obtained, following the method described in [17], the distribution of the best cut found in 100 independent trials. The mean and standard deviation were then compared with the corresponding moments of the best found in each of the C-L-O runs. This put L-K-L under a better light, but C-L-O was still the champion, and all the more so as N increased. In particular, for $N = 500$, the average cut obtained by C-L-O was better than the average best cut found with 100 L-K-Ls; and for $N = 1000$, the C-L-O cuts were almost always better than the best cut found among the 2000 L-K-Ls.

This gives an idea of the relative performances of the algorithms on geometric graphs, and shows that C-L-O is better than L-K-L. Note that we have not tried to fine-tune the temperature to improve the performance of C-L-O. (We chose to present here the $T = 0$ results because the annealing schedule is then parameter free.)

Cut Size for Five, $N = 250$ Graphs

Algorithm					
L-K-L	11.6	13.7	9.6	10.8	13.8
100-L-K-L	4.4	6.0	3.2	4.0	9.5
C-L-O	4.0	6.0	2.3	4.0	8.9
Best Found	4	5	2	4	7

Cut Size for Five, $N = 500$ Graphs

Algorithm					
L-K-L	21.0	16.4	15.4	22.2	18.4
100-L-K-L	12.0	7.0	5.1	10.2	10.6
C-L-O	11.8	4.8	5.0	10.0	7.1
Best Found	9	3	4	8	5

Cut Size for Five, $N = 1000$ Graphs

Algorithm					
L-K-L	26.8	25.1	29.5	23.8	26.6
100-L-K-L	13.1	10.0	14.2	10.2	13.7
C-L-O	12.4	7.6	14.6	7.4	13.2
Best Found	8	5	11	5	10

Table 3: Average performance on 15 random geometric graphs of $d = 6$. There are five graphs each for $N = 250, 500$, and 1000, corresponding to the five columns of the table. For algorithm L-K-L, the value in the table shows the average over 2,000 runs of L-K-L from random starts. For algorithm 100-L-K-L, L-K-L was run 100 times from random starts and the best value was taken. The result shown in the table is the average value of that best, when this procedure is done many times. For algorithm C-L-O, the value in the table is the average over 20 runs of C-L-O, each of length 100 steps. The temperature of the C-L-O runs was set to zero. The values under Best Found are the min over all the previous procedures — this was always found among the 20 runs of C-L-O.

Performance on sparse random graphs

Random graphs (c.f. Appendix II) are universally used for benchmarks in graph partitioning studies. There are two regimes. When the average degree of random graphs is small, the min cut size does not grow linearly with N. When d becomes large on the other hand, the relative difference in performance of algorithms decreases as $1/d$, so that most algorithms perform well on dense graphs. Thus we have chosen an intermediate value, $d = 5$, which leads to cut sizes that scale with N at large N and that enables us to compare our results with those in [17]. Denoting the cut size per vertex for the various algorithms by $C(N)$, we find $C_{K-L}(500) = 0.49 \pm 0.01$. The error is quite large because the fluctuations from instance to instance are important. Interestingly, we have found that the *relative* performance of algorithms can be found to much higher accuracy because fluctuations in the performance ratios are about 20 times smaller. For $N = 500$, Johnson et. al. find $C_{S-A}/C_{K-L} = 0.918$, i.e., simulated annealing (using their implementation) leads on average to an 8.2% improvement over K-L from random starts. However, this does not take into the much greater cpu times necessary for S-A; each S-A run represented about 100 K-Ls. Thus as above, we also compared the expected best of

100 K-Ls: $C_{100K-L}/C_{K-L} = 0.925 \pm 0.006$. The parameter free ($T = 0$) runs of C-L-O used 100 K-Ls, so again the time is roughly comparable with that of S-A and 100 random starts of K-L. We find $C_{C-L-O}/C_{K-L} = 0.909 \pm 0.005$ (the average is over five instances, each with 20 random starts of C-L-O). Thus at $N = 500$, C-L-O is almost 1% better than S-A and 9.1% better than K-L. (Note that we get an additional 0.5% improvement by going to an annealing schedule.) The same number of runs for $N = 1000$ confirm these numbers. We find for that size $C_{C-L-O}/C_{K-L} = 0.911 \pm 0.002$, so that C-L-O improves K-L by 8.9%. Note however, that in taking the large N limit, it is necessary to scale the number of K-Ls used in C-L-O with N. Thus the above result is for 200 K-Ls for each C-L-O run. This requirement is simply due to the fact that each vertex should be successively considered as the starting point of the kick construction. For reference, the relative performance keeping the number of K-Ls at 100 is 0.9157 ± 0.002. We also find $C_{100K-L}/C_{K-L} = 0.944 \pm 0.003$ and $C_{200K-L}/C_{K-L} = 0.940 \pm 0.003$. Not surprisingly, this confirms that repeated random starts is ineffective as N becomes large.

7 Parallel C-L-O

This section discusses a parallel version of the C-L-O algorithm. Through the use of PVM (Parallel Virtual Machine, [24, 25]), we regularly run the algorithm in parallel on a local network of workstations. In fact, our only code is parallel — the single processor results given earlier were simply the parallel code executing on only a single processor.

There are a number of problems where local optimizations parallelize well, but the TSP and the GPP do not because the constraint of maintaining a feasible solution is not readily implemented in a distributed system. Thus we have only considered implementations where a given processor has a complete configuration in local memory. We work in the framework of a distributed memory architecture and, as mentioned above, we use the PVM message-passing system.

We parallelize C-L-O by searching P simultaneous Markov chains on P processors. This produces a population, at any given time, of P configurations. The members of the population share information with each other (that is, the Markov chains are interconnected) by branching and pruning among the members. This is very similar to Darwinian selection for genetic algorithms and diffusion Monte Carlo for some physics problems. This means that the best configurations are duplicated at the expense of the worst ones.

Branching and pruning events occur relatively rarely (as measured in CPU time) so communication times do not dominate the parallel algorithm. We currently use a simple selection process to guide the population of members. Namely, at infrequent intervals (typically every 10 to 100 steps, per processor), we let the best configuration replicate and prune all members of lesser quality. This "winner-take-all" approach is a good strategy late in a long run, where one is searching for the few remaining moves that will take one close to the global optimum. In this asymptotic limit, the parallel algorithm has little waste and its speedup is near-linear. After replication, many processors may contain copies of the same configuration, but they are given distinct random number seeds and so perform independent searches for profitable moves.

Other branching and pruning strategies, such as those that preserve more distinct components of the population, are trivial to implement with our code and these correspond to more typical genetic algorithms (though all of ours are parthenogenetic since each child has only a single parent).

Another issue for parallel C-L-O is load balancing. The search times fluctuate and the code is designed to function for a heterogeneous collection of processors, which implies that the processor speeds may vary. To cope with this, the parallel search is written so that it dynamically adapts in such a way as to give the most searches to the fastest processors. This can be done quite simply and is effective.

8 Conclusions

Many heuristic algorithms have been proposed for the TSP and the GPP. Nevertheless, the standard general purpose algorithms, simulated annealing and local search (L-K and K-L), have been the most effective methods, exceptions occurring only when the instances have special structure that can be taken advantage of with other methods. In this paper, we showed how it is possible to improve on these by combining them, leading to what we call Chained Local Optimization, C-L-O. Rather than start each local search from scratch, one perturbs the current configuration by a "kick" and then applies local search. Our algorithm samples only locally optimal configurations, considerably reducing the search space. For the sampling to be effective, it is necessary to adapt the "kick" to the kind of problem of interest, the smarter the choice of kick, the better the performance. We have shown how to choose good kicks in the context of the TSP and the GPP. For more general CO problems, the "kick" should correspond to a modification of the configuration that is not easily accessible to the local search moves and that is likely to maintain the low cost of the configuration. In our experience, chained local optimization surpasses simulated annealing and local search methods, leading to a state of the art optimization method both for speed and for solution quality. Finally, note that the C-L-O meta-heuristic, is general and can be rapidly implemented once the local search algorithm is coded.

Acknowledgements

We thank R. Bixby, W. Cook, E. Felten, G.C. Fox, R. Friedberg, D.S. Johnson, and H. Simon for help, criticism, and discussions. The comments of the referees were very helpful and allowed us to greatly improve the manuscript. Finally, we acknowledge significant help from the PVM group at OGI for the implementation of the parallelization. This work was supported in part by the grants DOE-FG03-85ER25009, MDA972-88-J-1004, NSF-ECS-8909127, and by a NATO travel grant.

References

[1] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, June 1991.

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

[3] O. C. Martin and S. W. Otto. Partitioning of unstructured meshes for load balancing. *Concurrency: Practice and Experience*, 7(4):303–314, 1995.

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

[5] M. W. Padberg and G. Rinaldi. A branch and cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60, 1991.

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

[7] S. Lin. Computer solutions of the traveling salesman problem. *Bell Syst. Tech. J.*, 44:2245–2269, 1965.

[8] S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *"Operations Res.*, 21:498–516, 1973.

[9] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[10] V. Černy. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J. Optimization Theory Appl.*, 45:41, 1985.

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

[12] J. Hopfield and D. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141, 1985.

[13] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part III (the TSP). Technical report. Bell Labs preprint.

[14] F. Barahona and A. Casari. On the magnetisation of the ground states in two dimensional Ising spin glasses. *Comput. Phys. Commun.*, 49:417–421, 1988.

[15] A. Pothen, H. Simon, and K. P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *"SIAM J. Math. Anal. Appl.*, 11(3):430–452, 1990.

[16] T. Bui, C. Heigham, C. Jones, and T. Leighton. Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms. In *26th ACM/IEEE Design Automation Conference*, page 775, 1989.

[17] 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.*, 37:865–892, 1989.

[18] B. Kernighan and S. Lin. An effective heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, 49:291, 1970.

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

[20] E. B. Baum. Towards practical "neural" computation for combinatorial optimization problems. In J. Denker, editor, *Neural Networks for Computing*, 1986. AIP conference proceedings 151.

[21] H. Crowder and M. W. Padberg. Solving large-scale symmetric traveling salesman problems to optimality. *Management Science*, 26:495, 1984.

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

[23] W. Cook, V. Chvatal, and D. Applegate. In R. Bixby, editor, *TSP 90*, 1992. workshop held at Rice University, April 1990.

[24] A. L. Beguelin, J. J. Dongarra, A. Geist, R. J. Manchek, and V. S. Sunderam. Heterogeneous network computing. In *Sixth SIAM Conference on Parallel Processing*, 1993.

[25] Jack Dongarra, Al Geist, Robert Manchek, and Vaidy Sunderam. Integrated PVM framework supports heterogeneous network computing. *Computers in Physics*, April 1993.

[26] D. S. Johnson. Local optimization and the traveling salesman problem. In *Proceedings of the 17th Colloquium on Automata, Languages, and Programming*, pages 446–461. Springer-Verlag, Berlin, 1990.

[27] S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *"J. Stat. Phys.*, 34:975–986, 1984.

[28] M. Hanan and J. M. Kertzberg. A review of the placement and quadratic assignment problems. *SIAM Review*, 14(2):324, 1972.

[29] B.W. Kernighan. *Some Graph Partitioning Problems Related to Program Segmentation.* PhD thesis, Princeton University, Princeton, NJ, 1969.

[30] A. E. Dunlop and B. W. Kernighan. A procedure for placement of standard-cell VLSI circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-4, No. 1:92, 1985.

[31] Y. Fu and P. W. Anderson. Application of statistical mechanics to NP-complete problems in combinatorial optimization. *"J. Phys. A*, 19:1605, 1986.

[32] M.K. Goldberg and R. Gardner. On the minimal cut problem. In J. A. Bondy and U. S. R. Murty, editors, *Progress in Graph Theory*, page 295, 1984.

[33] M. Berger and S. Bokhari. A partitioning strategy for non-uniform problems on multiprocessors. *IEEE Transactions on Computers*, C-36(5):570, 1987.

[34] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Workshop*, page 175, 1982.

Appendix I: The Traveling Salesman Problem

Given N cities labeled by $i = 1, ...N$, separated by distances d_{ij}, the traveling salesman problem (TSP) consists in finding the shortest tour, i.e., the shortest closed path visiting every city exactly once. We consider only the symmetric TSP where $d_{ij} = d_{ji}$. The problem of finding the optimum tour is NP-complete. The main exact algorithms are branch and bound methods, and branch and cut methods. See Lawler et. al. for an overview [4]. These methods have progressed tremendously in the last ten years, so that instances with N of several thousand have now been solved to optimality [5]. A library of solved instances is available electronically [6], enabling us to test our algorithm.

Many heuristic methods have been proposed for the TSP, among them direct tour construction, local search [7, 8], simulated annealing [9, 10], genetic algorithms [11], and neural network approaches [12]. It is generally recognized that the heuristic that leads to the best solutions is a local search method due to Lin and Kernighan [8] (L-K). Simulated annealing (S-A) also gives very good tours, but it is many times slower than L-K [13]. Since our work builds on both L-K and S-A, we give some further details below.

The Lin-Kernighan local search

L-K is a variable depth local search. One begins with a notion of a neighborhood structure on the set of all feasible solutions (tours). Define the neighborhood of a tour, T, to be all those tours that can be obtained by changing at most k edges of T. One can search for local k-opt tours [7] by starting with a random tour T_1 and constructing a sequence of tours T_1, T_2, ... Each tour is obtained from the previous one by performing a k-change, i.e., by deleting k links and reconnecting the loose ends so as to still have a tour. The k-changes are required to decrease the length of the tour. When the process stops at a tour for which there is no possible improvement under a k-change, the tour is k-opt. Lin [7] introduced and studied the case of k=2 and k=3, and showed that one could get quite good tours quickly. In order to find the globally optimum tour, he suggested repeating the search from many random starts. Later, Lin and Kernighan [8] realized it was better to let k be variable. Essentially, their algorithm (L-K) is a breadth-first search (3-opt) followed by a depth-first search (using greedy 2-changes). It is the benchmark against which all heuristics are tested.

To quantify performances, one can measure the distribution of lengths obtained by running heuristics on a number of instances. The "standard" instances are: (1) instances with cities randomly distributed in the unit square; (2) instances with random distance matrices d_{ij}; and (3) publicly available instances solved to optimality by branch and cut methods [6]. The performance of L-K on these problems has been studied [13, 26, 1, 2]. As N becomes large, the distribution of lengths (normalized to the minimum) of tours found by L-K becomes Gaussian with a width decreasing as $N^{-1/2}$. Thus the performance of L-K is essentially described by the mean value of this distribution. For instances in category (1), the mean of L-K is probably about 1.5% above the minimum. For category (2) instances, it is theoretically known that the minimum length is $O(1)$ for large N. Interestingly, all local search methods except for L-K are rather poor for this category, giving rise to a mean length that increases with N [26].

Simulated Annealing

Simulated annealing has been applied with success to the TSP [9, 10, 13, 17]. One starts by constructing a sequence of tours T_1, T_2, etc... Each step of this chain is obtained by doing a k-change (moving to a neighboring tour). Usually, k is 2 or 3. The stochastic construction of a sequence of T's can be viewed as a modification of local search to include "noisiness." For the TSP, S-A is significantly slower than Lin-Kernighan, but it has the advantage that one can run for long times and slowly improve the quality of the solutions [27], eventually getting comparable or even better results than L-K. (See for instance the studies of Johnson et. al. [13].) In the simplest version of S-A, the elementary move when going from T_n to T_{n+1} is a random 2-change. A number of studies have considered more complicated moves, and also non-random ways of choosing the 2-changes. Indeed, if one uses only 2-changes, the final tour (when the temperature has reached 0) is only guaranteed to be 2-opt. By including both 2 and 3-changes, the final tour becomes 3-opt, leading to significant improvements [27].

Appendix II: The Graph Partitioning Problem

Consider an un-oriented graph G=(V,E), i.e., a collection of vertices V_i, $i = 1, ...N$, and edges $E_{i,j}$ ($E_{i,j}$ joins vertices V_i and V_j). The graph partitioning problem consists in finding a partition of V into two subsets A and B of specified sizes so that the number of "cut" edges is minimized. An edge $E_{i,j}$ is cut if its endpoints belong to different subsets. We concentrate on the "standard" formulation of the the graph partitioning problem in which N is even and A and B are of equal size. (One then talks of the graph bi-partitioning problem, hereafter referred to simply as the GPP.) This is not a significant restriction as unequal sizes can also be dealt with using the same heuristics [18].

The GPP has great practical importance: it is a major ingredient in the problem of cell placement for VLSI [28], chip layout, and program segmentation [29]. These optimization problems use some form of graph partitioning or generalizations thereof [30] in their solution.

The GPP is NP-complete; exact methods are less developed than for the TSP, so that there are few large instances that have been solved to optimality. For heuristics, the situation is quite analogous to that for the TSP. The two "best" heuristics are a variable depth search heuristic due to Kernighan and Lin [18] (K-L), and simulated annealing [9]. Again, these two algorithms are comparable in terms of "quality" of solution, but S-A is substantially slower. However, these algorithms as not very good for many practical instances such as occur in load balancing and layout problems, so much effort has been spent finding problem specific improvements.

The Kernighan-Lin local search and extensions

Just as in the discussion on Lin-Kernighan, it is possible to introduce a notion of a k-change. One calls a 1-change an exchange of one element of A against an element of B. It turns out that 1-opt is a mediocre algorithm, and that going to higher k-opt is very expensive and does not lead to much improvement. Kernighan and Lin [18] suggested a variable k-change algorithm that is much more effective than either 1-opt or 2-opt while being quite fast. Their algorithm is essentially a greedy tabu 1-exchange sweep of sets A and B: at each step, one exchanges the most favorable (or least unfavorable) pair of elements. During the sweep, if one element has already been exchanged, it can no longer be considered (it is tabu) for further exchange during that sweep. If the cut size does not decrease during the sweep, the partition is defined to be K-L optimal. If it does decrease, one takes the partition with the smallest cut during the sweep and uses that as the starting point for another sweep. The cut size is a decreasing function of sweep number, and one quickly reaches a locally optimal partition.

The performance of K-L depends to a large extent on the ensemble of graphs one considers. A natural ensemble of graphs is $G(N,p)$ the ensemble of random graphs of N vertices, each pair (V_i,V_j) being connected with probability p. If p is kept N independent, the average min cut size at large N is given by [31]:

$$< \text{MinCut} >= pN^2/4 - UN^{3/2}[p(1-p)]^{1/2}/2 \qquad U = 0.38...$$

The first term is simply the contribution from a random cut; the second term is the improvement due to the optimization. For sparse graphs, (e.g., $p \approx O(1/N)$), with an average degree $d = p(N-1)$, the min cut size is proportional to N, $< \text{MinCut} >= C(d)N$. Most algorithms perform well on the dense graphs, and less well on the sparse ones. The performances on this last ensemble can be compared on the basis of the $C(d)$ obtained by the algorithms, i.e., on the basis of the cut size per vertex. We find that K-L gives $C_{K-L}(5) = 0.49 \pm 0.01$. See section 6 for further details.

For graphs that have an embedded structure such as occurs in load balancing applications, K-L and S-A behave much more poorly. Thus quite a bit of effort has been spent on trying to improve K-L for these types of graphs. One method, called compaction [16], combines pairwise nearby vertices and runs K-L on the half-sized problem. Then the solution is unpacked to get a new trial partition that itself will be K-L-opted. This procedure can be done recursively, giving rise to a hierarchical or multiple scales algorithm [32]. A second approach consists in using better than random starting partitions. In particular, Berger and Bokhari [33], and later Johnson et. al. [17] proposed using a dividing line for the initial partition, and applied K-L to such starts. The resulting algorithm, called L-K-L, gives for "geometric" graphs (c.f. section 6) as good results as a hierarchical compaction approach while being much simpler. Thus in that section we have restricted ourselves to presenting comparisons of our algorithm to K-L and L-K-L.

Simulated Annealing

In S-A for the GPP, the natural choice is to exchange pairs of vertices between A and B. Another possibility is to move one element at a time, but never let the imbalance be greater than one or k elements [34]. Or, one can simply accept any imbalance, but include an extra penalty cost that grows with the imbalance, so that on average the imbalance stays small [9]. It turns out that moves consisting in exchanging elements pairwise are not efficient; it is simpler and faster to allow any imbalance, and usually this is done in conjunction with a penalty that is quadratic in the imbalance. Relaxing this hard constraint into a soft one gives the system new routes to escape from local minima. Also, it has the advantage of reducing the size of neighborhoods from $N(N-1)/2$ to N. A comparison of S-A and K-L was made in [17]: S-A is better than K-L for random graphs, (they find for random graphs of average degree $d = 5$, $C_{S-A}/C_{K-L} = 0.918$), but significantly worse than K-L for "geometric" graphs.