An Improved Adaptive Multi-Start Approach to Finding Near-Optimal Solutions to the Euclidean TSP

Dan Bonachea
Computer Science Dept.
University of California
Berkeley, CA 94720
bonachea@cs.berkeley.edu

Eugene Ingerman
Mathematics Dept.
University of California
Berkeley, CA 94720
eugening@math.berkeley.edu

Joshua Levy
Mathematics Dept.
University of California
Berkeley, CA 94720
jdl@math.berkeley.edu

Scott McPeak
Computer Science Dept.
University of California
Berkeley, CA 94720
smcpeak@acm.org

Abstract

We present an "adaptive multi-start" genetic algorithm for the Euclidean traveling salesman problem that uses a population of tours locally optimized by the Lin-Kernighan algorithm. An all-parent cross-breeding technique, chosen to exploit the structure of the search space, generates better locally optimized tours. Our work generalizes and improves upon the approach of Boese et al. [2].

Experiments show the algorithm is a vast improvement over simple "multi-start," i.e., repeatedly applying Lin-Kernighan to many random initial tours. Both for random and several standard TSPLIB [5] instances, it is able to find nearly optimal (or optimal) tours for problems of several thousand cities in a few minutes on a Pentium Pro workstation. We find these results are competitive both in time and tour length with one of the most successful TSP algorithms, Iterated Lin-Kernighan.

1 BACKGROUND

1.1 THE TSP

In the traveling salesman problem (TSP) we are given n points (or "cities") $c_1, \ldots, c_n$ and a positive distance $d(c_i, c_j)$ for each distinct pair of cities. Our goal is to find an ordering π, or *tour*, of the cities that minimizes the *length* of the tour, $d(c_{\pi(n)}, c_{\pi(1)}) + \sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)})$. We will restrict our attention to the two-dimensional Euclidean TSP, which is the special case where the cities are points in the plane and $d(c_i, c_j)$ is the Euclidean distance from c_i to c_j. This optimization problem is NP-hard.

1.2 LOCAL SEARCH ALGORITHMS

Many methods for finding approximate solutions have been studied; Johnson and McGeoch [3] provide an excellent survey. Traditional approaches involve using a construction heuristic to construct an initial tour, then running a local search algorithm to improve it. For a long time, the most successful local search method was the Lin-Kernighan (LK) algorithm [4]. Lin-Kernighan involves the usual 2-opt and 3-opt moves that swap two or three edges in the tour, but tries to overcome the tendency of local search algorithms to find local minima that are far from the optimal solution by allowing some uphill moves that increase tour length.

These algorithms are generally able to find tours within a few percent of optimal; finding tours very close to optimal length, however, is difficult. Iterated Lin-Kernighan (ILK) is an improvement on the standard Lin-Kernighan algorithm in which certain "double-bridge 4-opt" moves (sometimes called "kicks") are performed when the LK local search stalls. (These type of kicks are chosen because they are difficult for LK to undo.) It has been found to be one of the most effective algorithms at finding high-quality solutions, even for large problems [3].

1.3 GENETIC ALGORITHMS

The literature contains many examples of the application of genetic algorithms to the TSP. However, Johnson and McGeoch [3] and others have found traditional, pure genetic approaches are not seriously competitive with other conventional algorithms. However, by incorporating local search techniques, genetic approaches have been improved. Typically, individuals in the population are taken to be local minima under a chosen local optimization method, as in the work of Boese et al. [2] and Merz and Freisleben [6]. A distinctive part of the algorithm presented in the former

paper is the repeated cross-breeding of the *entire* population to create a single child. Our work follows in the same vein, but yields an algorithm with substantially improved performance.

2 THE ADAPTIVE MULTI-START APPROACH

2.1 THE BIG VALLEY

In a 1994 paper, Boese et al. [2] proposed a new TSP approximation algorithm. It is based on the observation that, in most cases, the locally optimal solutions—optimized by 2-opt local search in the paper—share many edges with each other and with the optimal solution. The authors called this observation the "big valley" hypothesis and supported it with several experimental results. Their work was restricted to 2-opt local minima and random problem instances, i.e., problems where points are chosen uniformly at random on a square. The *bond distance* between two tours is defined as the number of edges in one tour not coinciding with an edge in the other. The authors proved that the bond distance is within a factor of two of the 2-*opt distance*, which is the least number of 2-opt operations needed to get from one tour to another.

2.2 ADAPTIVE MULTI-START

How does one exploit the big valley? Boese et al. present the adaptive multi-start (AMS) algorithm. First, they create a population of tours by running 2-opt local search on random initial tours. Then they repeatedly create a "child" tour by adding edges from the "parent" tours with probability proportional to the *fitness* of an edge, where the fitness of an edge is determined by the length and number of the parent tours containing the edge. The edges that occur more often or in shorter tours in the parent population have a better chance of appearing in the new tour. After the legal edges in the parent population are exhausted, the fragments of the partial tour are reconnected at random to form a complete tour, and 2-opt is run on the tour. If the child has a shorter tour length than one of the parents, it replaces the worst tour in the population.

In this work, we improve on the method of Boese et al. by (i) choosing different population sizes and numbers of generations; (ii) using another fitness function; (iii) implementing the child generation more efficiently and effectively; and (iv) using Lin-Kernighan local optimization. While the algorithm of Boese et al. performed only somewhat better than 2-opt, the ver-
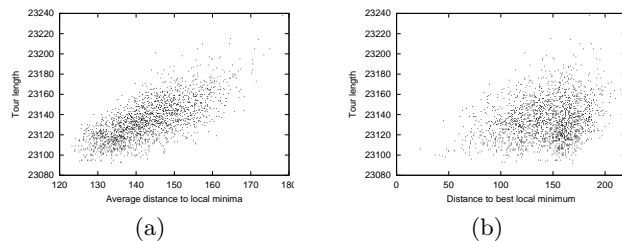

Figure 1: For 2000 LK local minima, (a) the average bond distance to the other local minima, and (b) the bond distance to the best local minimum, for a random 1000-point instance.
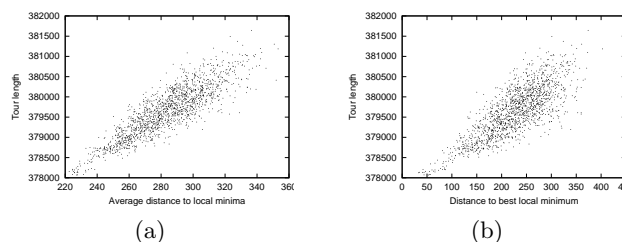

Figure 2: For 2000 LK local minima, (a) the average bond distance to the other local minima, and (b) the bond distance to the best local minimum, for TSPLIB's pr2392 instance.

sion of AMS presented here finds tours of much better quality—in fact often better than those of the Iterated Lin-Kernighan algorithm (see §6).

2.3 THE BIG VALLEY HYPOTHESIS FOR LK TOURS

In our algorithm, we use Lin-Kernighan for local optimization on the tours. We now give some evidence that the big valley hypothesis still holds for LK-optimized tours.

For several Euclidean TSPs, we ran the LK algorithm on 2000 random initial tours and compared the 2000 locally optimal tours to each other and to the best locally optimal tour. Figures 1 and 2 show the results for a 1000-point random problem and a 2392-point non-random problem (from TSPLIB [5]).

Observe that in each case, the quality of local optima correlates with the distance between the local optima. That is, they cluster around the best local optima. Thus it is plausible that by finding locally optimized tours that are near many of the optima, we will find better tours.

The correlation is not as strong between the quality of local optima and distance to the *best* of the local
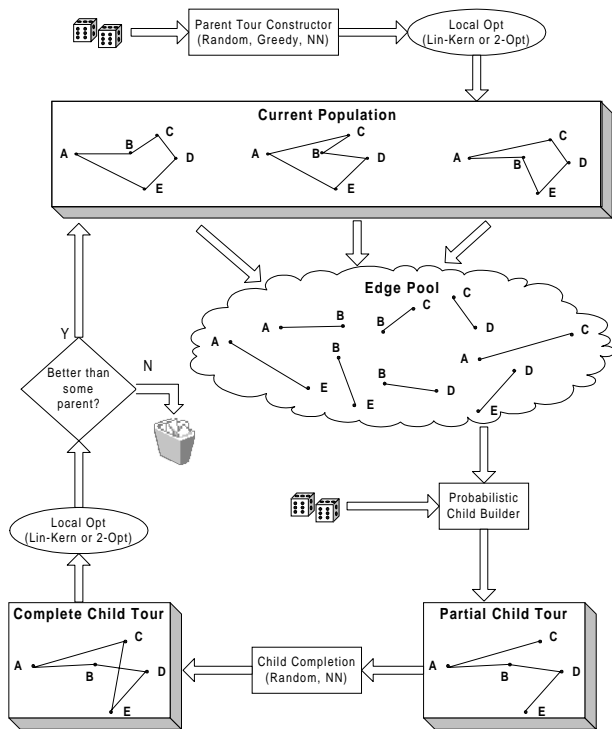
Figure 3: Structure of the algorithm.

optima found. Yet this should not be too discouraging: generally, a number of local optima are very good, though they may differ in a fair number of edges. (Indeed, a globally optimal tour might not be unique.) This explains the lower bulge of points in Figure 1(b), which represent good tours distant from the best local optimum.

The plots give credence to the big valley hypothesis for LK-optimized tours, and suggest that the AMS approach is compatible with LK-optimized tours.

3   THE ALGORITHM

3.1   GENERAL STRUCTURE

Figure 3 provides a conceptual diagram for our AMS algorithm. The algorithm is genetic in nature: at every generation, a randomized cross-breeding operation is applied to the parent tours to create a child tour that contains many of the same edges as the parents. This child tour is subsequently locally optimized and added to the population, provided the child short enough.

The algorithm operates in two phases. In the first phase, it generates a set of initial tours using a tour construction heuristic (such as random, greedy, or nearest-neighbor) and each of these parent tours is lo-

cally optimized. The resulting tours are the initial population.

In the second phase, we begin by collecting the union of all the edges which appear anywhere in the population into an "edge pool," and calculating a fitness value for each edge in the pool based on characteristics of the edge itself and the tours containing that edge. (Note this differs from traditional genetic approaches, which generally assign fitness to individuals in the population, not their components.) Next, we repeatedly choose edges from the pool at random with probability proportional to their fitness values and add them to a child tour we are growing, omitting any edges that would create an illegal tour, until no more edges may be added. Since we may exhaust all the edges in the pool and still be left with an incomplete child tour, we may also need to add some new edges to connect the tour fragments. This completion can be accomplished using an edge-length based heuristic such as nearest-neighbor or greedy, or with a simple-minded random selection. Finally, the complete child is passed through the local optimizer, and compared to the population. If the new tour is shorter than the longest tour in the population, the new tour will replace the longest tour. The new population then becomes the basis for the subsequent generation. As we will see, the population eventually reaches a point where no further progress can be made, and we stop the algorithm (see §5). (One could generate several children, then incorporate the good ones into the population, but this does not improve performance.)

The described algorithm has a large number of parameters that can be adjusted to change its behavior, such as the choice of tour construction heuristic, the size of the population, the child completion algorithm, and the choice of fitness function. These choices are discussed in §3.3 and §5.

3.2   COMPARISON TO BOESE ET AL. ALGORITHM

The general idea of our algorithm is similar to that of the algorithm described by Boese et al. However, there are some significant differences. For example, the Boese et al. algorithm spends about half of the time building the initial population and the other half generating child paths, while our algorithm, when properly adjusted, spends less time on the initial population and focuses on a larger number of generations (a ratio of perhaps 1:1000 instead of 1:1). Our algorithm uses a quality-conscious (and in fact faster) heuristic for child tour completion instead of random completion. It uses Lin-Kernighan instead of 2-opt

for local optimization, which (not surprisingly) produces much better results. It also runs local search on each child path only once, rather than relying on non-determinism in the local search to provide different optimized children. Finally, we use a more sophisticated fitness function for assigning edge probabilities.

3.3 THE FITNESS FUNCTION

The AMS algorithm uses a fitness function to decide the probability of picking each edge. The choice of this fitness function greatly influences the algorithm's performance, both in terms of tour quality and running time. Since there is no single obvious choice, we studied several possibilities and empirically evaluated their effectiveness.

We want the fitness of an edge to correlate positively with factors that contribute to good tours. Possible factors include edge length, the lengths of the population tours that contain the edge, and the edge's *popularity*, or the fraction of parents containing the edge. Parameters in the function make it possible to strike a favorable balance among the factors.

The fitness function we found most effective was of the form

$$Fit(e) = \alpha Q_E + (1 - \alpha)PQ_T$$

for any edge e. Q_E reflects the relative quality (length) of e compared to other edges in the pool, Q_T reflects the relative quality of the best tour containing the edge, and P is a scaling factor dependent on the popularity of the edge. The parameter α, in the range 0 to 1, influences the relative importance of the two terms.

We define Q_E to be a linear function of the length of e, scaled so that the shortest edge in the pool has $Q_E = 1$ and the longest has $Q_E = 0$. Q_T is defined similarly: it is a linear function of the length of the shortest tour in the population that contains the edge, scaled to the range 0 to 1. Finally, we let $P = e^{\beta(\text{pop}(e)-1)}$, where pop($e$) is the popularity of e and the parameter $\beta \geq 0$ controls the importance of popularity.

4 IMPLEMENTATION

4.1 TOOLS

We implemented the algorithm described in §3 in about 2500 lines of C++ code [7]. The implementation of the Lin-Kernighan local search algorithm and the parent tour construction heuristics were taken from the CONCORDE [1] library of Applegate et al., a freely available collection of TSP-solving utilities.

4.2 COMPLEXITY

The complexity for our implementation is $O(pS + g(np \log np + S))$, where n is the number of cities, p is the population size, g is the number of generations, and S is the running time for the local search.

The algorithm generates the parents (complexity $O(pS)$), collects their edges into a pool, and assigns a fitness value (see §3.3) to each edge (taking $O(np)$). The probabilistic child builder, which runs in $O(np \log np)$, repeatedly selects edges from the pool at random with probability directly proportional to their assigned fitness values, and adds them to the child if they are legal tour edges (checking this takes constant time).

In practice, our implementation spends over three-quarters of its CPU cycles on local searches.

4.3 PARALLELIZING AMS

We successfully parallelized the child generation phase of the algorithm, so that many children are generated on different processors concurrently. Because no communication is needed while the local search is performed, performance increases nearly linearly with the number of processors, even when communicating across a slow network. The parallel version allows large problems to be solved quickly when a network of workstations is available. However, for the sake of useful performance comparisons, all results discussed below were gathered from the sequential version of the program.

5 BEHAVIOR OF THE ALGORITHM

5.1 A SAMPLE RUN

Figure 4 shows a typical execution of the algorithm, in this case on a 1000-point random instance. It shows how tour lengths and the diversity of the population change during the run. We define *diversity* as the number of distinct edges in the pool, scaled so that diversity is zero when all the tours in the population are identical, and diversity is one when all the tours are completely disjoint. The new children have widely varying lengths, but the best and worst tours in the population improve steadily as long as the diversity of the population is high. Note that whenever the "new child" line drops below the dotted line indicating the worst parent length, the new child replaces the worst parent in the population. After about 1300 generations, the population "converges": the best and worst
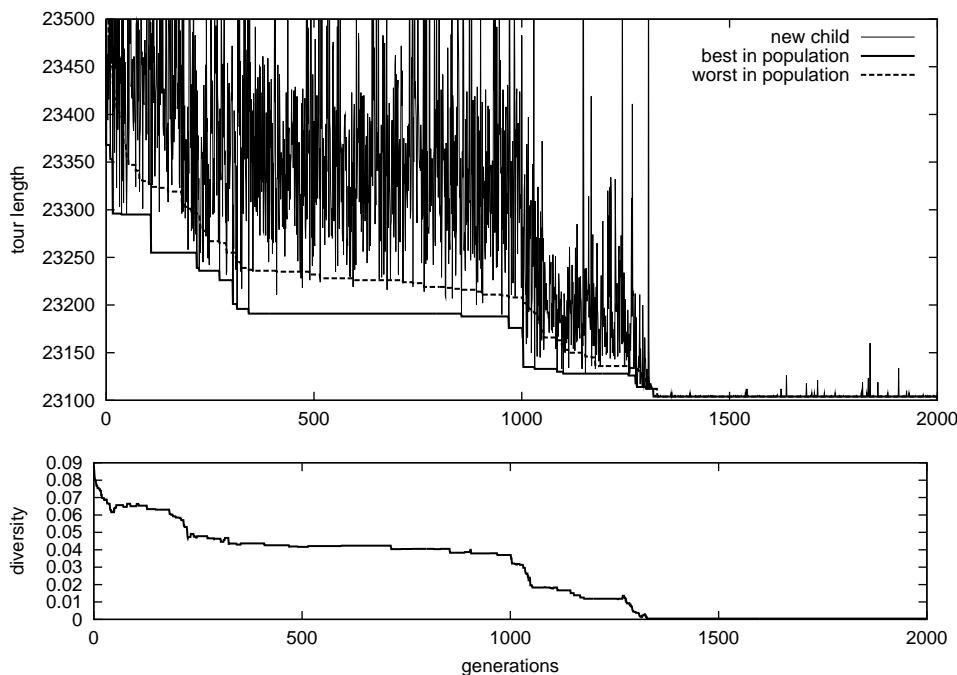
Figure 4: A typical run on a 1000-point random problem: best, worst, and new child optimized tour lengths (above) and diversity (below). (Population size is 10, $\alpha = .05$, and $\beta = 1$.)

tour lengths coincide, the noise in child generation drops to near zero, diversity drops to nearly zero, and the best tour length stops improving.

The correlation between diversity and convergence indicates interesting similarities between AMS and simulated annealing (SA). The diversity of the population in AMS acts like temperature in SA; when diversity is high, the algorithm is exploring many widely separated regions of the search space. But as diversity drops, the algorithm "homes in on" the best solution in the region it has selected. However, while the "cooling schedule" in SA is user-specified, AMS diversity drops according to internal factors.

5.2   SELECTION OF PARAMETERS

The parameters, especially population size, influence the diversity schedule. With only a few parents (less than 5, say), AMS can find a solution very quickly, but it is not very near to optimal. With many parents (20 or more), AMS takes quite a bit longer to reach convergence, but the resulting tour is very good. This behavior is illustrated in Figure 5. (Note that in the figure the population of 20 has not converged in the 2000 generations shown. It ultimately reaches tour quality slightly better than that of the population of 10.)

The value of α also influences the performance. In general higher values of α make the addition of unpopular or poorly performing edges more likely, raising diversity and slowing convergence. Picking α slightly larger than zero, say $\alpha \approx 0.05$ seems to work well. Performance is not as sensitive to β; values near 1 work well.

We find the choice of construction heuristic and child completion heuristic have a moderate effect on the algorithm. Generally, random initial tours work better than nearest neighbor or greedy, and nearest-neighbor tour completion performs better than random tour completion. This is not surprising, since we want as much diversity as possible at the start, and want each child tour to be short. (Boese et al. claim that the choice of child completion heuristic has little effect. This may be a feature of their 2-opt local optimization, or it may be that their experiments involved too few generations to show the difference.)

6   PERFORMANCE AND COMPARISONS

All running times given in this section were measured on 200 MHz Pentium Pro workstations running the Solaris operating system.
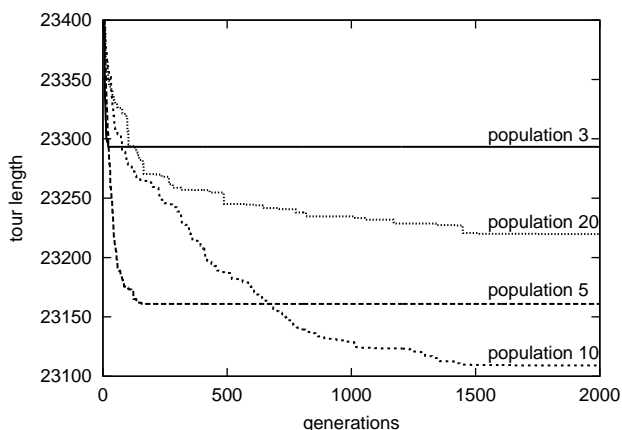
Figure 5: Best population tour lengths for a 1000-point random problem (average of 10 runs) for various sizes of the population.

6.1 DOES BREEDING HELP?

We can first check if the "adaptive" breeding of good tours is worthwhile by comparing to "unadaptive" multi-start, where instead of breeding to get new tours, we repeatedly pick random tours, then apply the same local optimization. Typical results are shown in Figure 6. Clearly, the adaptive method is much faster at finding good tours than repeated LK.

6.2 COMPARISONS TO OTHER ALGORITHMS

The 2-opt-based Boese et al. algorithm also performed better than its unadaptive counterpart. However, because 2-opt itself is inferior to Lin-Kernighan local optimization [3], one application of LK generally surpasses tour qualities achieved by hundreds of 2-opted child generations, at least in the case of the small problems discussed in their article (100 cities placed at random). Performance data is not supplied for larger problems, but it is reasonable to assume LK would continue to out-perform an algorithm that relies upon 2-opt. For a more meaningful comparison, we turn to an algorithm that also makes use of LK.

The CONCORDE library provides an implementation of the Iterated Lin-Kernighan (ILK) algorithm. As this algorithm is one of the most successful at finding high-quality tours [3], it is an obvious candidate for comparison with AMS. Moreover, since AMS uses precisely the same LK local search implementation as ILK, run-time comparisons can be performed easily and fairly.

Our experiments focused on problems of several thousand cities. At these sizes, ILK runs quickly, apply-

ing thousands of "kicks" in a few minutes. However, like LK itself, it eventually gets trapped near a local minimum from which even the kicks find no escape. Thus, for best-quality results, it should be run multiple times with random starting tours. Figure 7 shows single runs of ILK on a 1000-point random problem, where the times indicate the amount of computation required before the algorithm failed to improve further. Also, to give an indication of the difficulty of finding near-optimal tours, the best results of a few repeated ILK runs—sometimes taking tens or hundreds of times longer—are shown.

For comparison, a sample of runs from AMS is also shown, in this case for a population size of 10. While AMS is clearly much slower than a single run of ILK, it seems to reliably find better tours: many single ILK runs are never able to do as well as a typical single AMS run. If we take into account the time needed to repeat ILK until a high-quality tour is found, we see AMS can be significantly faster. Generally, we see AMS is much more stable in its performance, a property we would expect from its population-based approach. A larger population, say of size 20, both improves tour quality and reliability, at the expense of a factor of 5–10 in running time.

In preliminary tests on problems an order of magnitude larger, AMS performance is poorer than ILK. We believe this is due to the global nature of the LK optimization required by each AMS generation. However, with such a computationally intensive problem, one may be able to use ILK itself as the local search in AMS (see §7.1).

6.3 NON-RANDOM PROBLEMS

The previous results are for randomly chosen points. The TSPLIB archive [5] has many more interesting point configurations, taken from applications ranging from geography to printed-circuit board layouts. For many problems, provably optimal solutions (found by branch and bound methods, for example) are known. Table 1 lists the performance of AMS on six differently structured problems, for various population sizes.

As a comparison, ILK was run repeatedly, each time for enough "kicks" (8000–10 000) to obtain a stable result, until somewhat more than the amount of CPU time taken by the corresponding AMS run was expended.

The average lengths, best lengths, and times are determined from ten trials for each problem. We see that for each problem the best AMS result is at least as short as the best ILK result, and often significantly
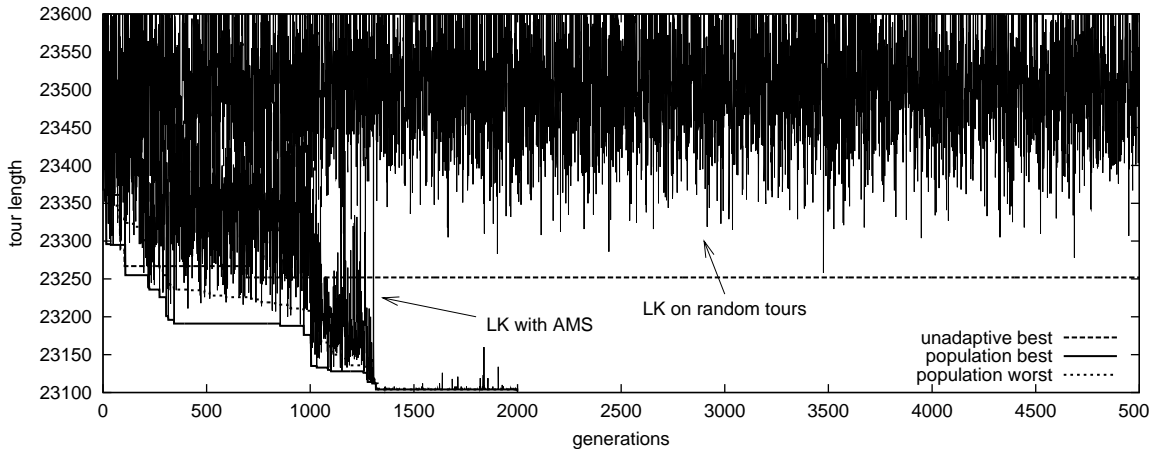
Figure 6: A typical run on a 1000-point random problem, and an "unadaptive" repetitive LK run.
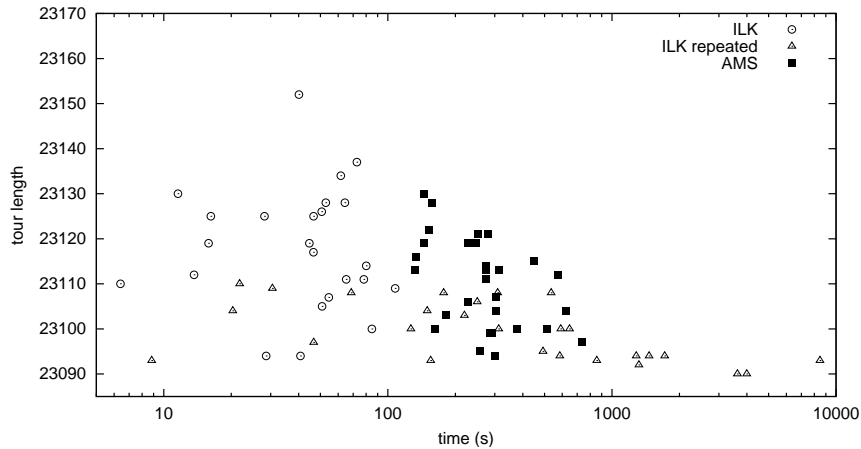

Figure 7: Quality of ILK and AMS tours versus CPU time (on logarithmic scale) to find them.

(considering the closeness to optimality) shorter. In all but one case, AMS also performs better on average, particularly for the largest three problems.

7   FUTURE WORK

7.1   IMPROVING THE ALGORITHM

The number of possible variations on AMS is dizzying. Not only can parameters such as α and population size be varied, but changes could be made in the fitness function, the local search, or the scheme for picking edges and completing children; it would be foolish to think the experiments described more than scratch the surface of what is possible with an algorithm of this type.

An interesting variation is to use a more powerful local search, such as ILK. This is of course quite expensive and only of interest when tours very close to optimal are needed. Preliminary results indicate it to be more effective than simple repeated ILK from random starting tours (a typical method when using ILK). We are currently exploring this approach.

Altering the structure of the population may be helpful. For example, one could increase the size of the population as diversity lowers. Dynamically changing population size may allow more direct control of the algorithm and perhaps yield better results. Or, one could use multiple smaller populations simultaneously. This could improve speed since small populations find better tours faster. The good descendents of these small populations could then be merged into a new population, and breeding could continue.

Table 1: AMS and ILK performance on TSPLIB problems.

name	optimal	AMS					ILK			
		pop.	avg.	best		time	avg.	best		time
(size)	length	size	length	length	(excess)	(min)	length	length	(excess)	(min)
gr666[a]	294358	20	294478	294358	(0%)	12	294411	294358	(0%)	17
u1060	224094	20	224131	224115	(0.009%)	62	224144	224121	(0.012%)	67
pcb1173	56892	15	56894	56892	(0%)	5	56923	56892	(0%)	7
d2103	80450	15	80534	80455	(0.006%)	16	80643	80563	(0.14%)	17
pr2392	378032	15	378064	378033	(0.0003%)	14	378446	378107	(0.020%)	17
rl5934	556045	10	556800	556403	(0.07%)	146	557345	556451	(0.07%)	180

[a]This instance uses a non-Euclidean metric.

7.2 EXTENSION TO OTHER PROBLEMS

While this paper has focused on the Euclidean TSP, the AMS algorithm could certainly be applied to the general (symmetric) TSP problem. Since AMS is based on the Lin-Kernighan local search, which is quite robust, it is plausible that the adaptive multi-start approach we have outlined could make headway on other variations of the traveling salesman problem.

8 CONCLUSION

Adaptive multi-start is a promising approach to finding high-quality solutions to the Euclidean TSP and perhaps more general TSPs. It combines good qualities of local search and genetic algorithms, while avoiding many of the pitfalls of each. The algorithm cannot easily get stalled at a poor local minimum, as local searches can, because an entire population of tours is maintained. New tours are constructed through a heuristic that exploits structural properties of the search space. Finally, the richness of options in designing such an algorithm means improvements are likely.

Acknowledgments

References

[1] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, CONCORDE, a computer code for the traveling salesman problem (preliminary version, August 27, 1997), available at *http://www.caam.rice.edu/~keck/concorde.html*.

[2] K. D. Boese, A. B. Kahng, and S. Muddu, "A new adaptive multi-start technique for combinatorial global optimizations," *Operations Research Letters* **16** (1994), pp. 101–113.

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

[4] S. Lin and B. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," *Operations Research* **21** (1973) pp. 498–516.

[5] G. Reinelt, "TSPLIB—A travelling salesman problem library," *ORSA Journal on Computing* **3** (4) (1991), pp. 376–384. Available at *http://www.iwr.uni-heidelberg.de/iwr/comopt /software/TSPLIB95/*.

[6] P. Merz and B. Freisleben, "Genetic local search for the TSP: new results," *Proceedings of 1997 IEEE International Conference on Evolutionary Computation* (Indianapolis, IN, USA, 13–16 April 1997), pp. 159–64.

[7] The source code for AMS is available from *http://www.cs.berkeley.edu/~bonachea/tsp/*.