Evolutionary Computational Approaches to Solving the Multiple Traveling Salesman Problem Using a Neighborhood Attractor Schema

Donald Sofge[1], Alan Schultz[1], and Kenneth De Jong[2]

[1] Navy Center for Applied Research in Artificial Intelligence,
Naval Research Laboratory
{Sofge, Schultz}@aic.nrl.navy.mil
[2] Department of Computer Science,
George Mason University
kdejong@cs.gmu.edu

Abstract. This paper presents a variation of the Euclidean Traveling Salesman Problem (TSP), the Multiple Traveling Salesman Problem (MTSP), and compares a variety of evolutionary computation algorithms and paradigms for solving it. Techniques implemented, analyzed, and discussed herein with regard to MTSP include use of a neighborhood attractor schema (a variation on k-means clustering), the "shrink-wrap" algorithm for local neighborhood optimization, particle swarm optimization, Monte-Carlo optimization, and a range of genetic algorithms and evolutionary strategies.

1 Introduction

The Euclidean Traveling Salesman Problem (ETSP), or simply TSP, is a well-known and extensively studied benchmark for many new developments in combinatorial optimization [1],[2],[3],[4],[5],[6], including techniques in evolutionary computation [7],[8],[9]. A very simple yet highly practical extension of TSP is a TSP with multiple sales agents. This problem is defined as follows:

> **Def. MTSP(n, k)** --- given ***n*** cities and ***k*** sales agents, find the ***k*** closed circuit paths which minimize the sum of the squares of the path lengths.

This paper will henceforth refer to this problem as the multiple traveling salesman problem, or MTSP. The MTSP is actually a two-level optimization problem. On the first level we wish to determine the optimal subdivision of cities into ***k*** groups. The second level of optimization is to find the minimum length circuit for each of the groups of cities. This paper will be primarily concerned with the first level of optimization, determining the optimal subdivision of cities into groups, since this area has received less attention in the literature than solving the Euclidean TSP.

S. Cagnoni et al. (Eds.): EvoWorkshops 2002, LNCS 2279, pp. 153–162, 2002.

The MTSP is similar to the k-TSP problem discussed in [5], but excludes the requirement that each circuit include a common base city. The cluster optimization task is also related to the minimum sum-of-squares clustering problem (MSSC) [10]. However, in MSSC the distance measurement for each cluster is the sum of squared distances between each point and the cluster centroid, whereas in MTSP the relevant distance measurement for each cluster is the squared circuit length.

A note of explanation about the definition of MTSP given above is in order. Why do we want to find the ***k*** closed-circuit-paths which minimize the sum of the squares of the circuit lengths, rather than simply minimizing the sum of the circuit lengths? The reason is in order to distribute the work load as equitably as possible amongst the ***k*** sales agents. Thus, in addition to the constraint typically found in TSP of finding the minimum length circuit, we have imposed an additional constraint for distributing the cities amongst the sales agents. Minimization of the sum of the squares of the circuit lengths satisfies these constraints.

2 Approach to Solving MTSP

MTSP is a two-level optimization problem, with one problem being to determine how to subdivide the list of n cities into k groups or neighborhoods (i.e. clustering), and the second problem being to optimize the circuits for each cluster or neighborhood. While it may be possible to route first then cluster [5], we chose to cluster first then route since this seemed to be a more logical decomposition of the problem.

Since our cities lie in a Euclidean space and we wish to minimize local path lengths, it is reasonable to use a distance metric between each city and each cluster center to determine neighborhood affiliation. However, with MTSP we have an additional constraint of distributing the circuit lengths roughly evenly amongst the ***k*** neighborhoods. Since we cannot assume homogeneous distribution of the cities in Euclidean space, and it is quite reasonable to expect that given a randomly generated set of city coordinates (or those found on a real map) some areas will be significantly more densely populated with cities than others, we need another term or condition to better define our clusters.

2.1 Neighborhood Attractor Schema

The central idea behind the neighborhood attractor schema is that a number of moveable attractors are added to the map which have the effect of “capturing” cities. Each attractor has a position (x, y) and an attraction constant (a). The attraction between each city and each attractor is calculated as the attraction constant of the cluster attractor (a) divided by the distance (or squared distance) between the city and attractor. This feature allows the neighborhood attractors to then partition the non-homogeneously distributed cities into clusters such that the second constraint of MTSP (equitable distribution of work) is more easily satisfied. This approach is re-

ferred to as the neighborhood attractor schema. A result of applying the neighborhood attractor schema to MTSP(30,5) is shown in **Figure 1**.

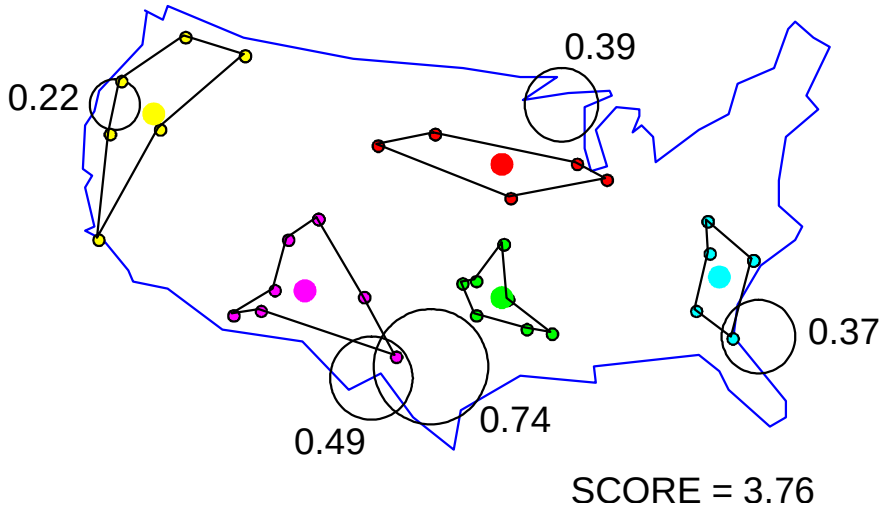

Fig. 1. MTSP(30,5) Optimal Solution Using Neighborhood Attractor Schema

This approach is roughly analogous to a gravitational attractor model where each planetoid or moon is "affiliated" with a planet based upon the gravitational attraction between the planetoid and the planet. In this case the “planetoids” are the cities (each with unit mass) and the “planets” are the basins of attractions, with each having a position (x,y) and mass parameter. The positions and mass parameters for the attractors are optimized using an evolutionary algorithm, particle swarm or other technique.

In **Figure 1** the large circles represent the neighborhood attractors, with their diameters determined by their respective attractor constants (numbers shown next to circles). The smaller solid circles inside each cluster show the unweighted centroid for each cluster, which is used in the shrink-wrap algorithm discussed in the next section.

One advantage this schema has over k-means-clustering models is that the neighborhood attractors need only compete with other neighborhood attractors to determine city affiliations and corresponding path lengths. They do not need to move to the centers of the clusters of cities they attract. This feature gives them more flexibility for partitioning the cities into equitable subgroups, especially important where the distribution of cities is highly non-homogeneous.

2.2 Optimizing Neighborhood Closed Circuit Paths

Given a MTSP(n,k) and a "solution" of neighborhood attractors (and their corresponding city cluster groups), we still need a means of optimizing closed-circuit paths

for each of the neighborhoods in order to score or determine the fitness of our solution. Finding such an optimum for each neighborhood becomes a standard Euclidean TSP. Since we do not wish to address the full computational burden of solving ***k*** neighborhood TSPs before we can even evaluate our solution of ***k*** neighborhood attractors, a shortcut for estimating a good (if not always optimal) closed circuit path for each neighborhood is desirable.

A rather simple and fairly elegant technique for accomplishing this is the shrink-wrap algorithm (though certainly not the only one, see [5] for others). The mean for each cluster of cities is calculated, and then the cities' coordinates are translated in Euclidean space such that the cluster mean is at the origin. The (**x**, **y**) coordinates are then converted to polar coordinates (**θ**, **ρ**), and the order of cities is sorted by the polar coordinates (**θ** first, then **ρ**). The sorted order list is then linked in ascending order, with an additional link added from the last city in the list back to the first. This process is illustrated in **Figure 2**. The cities are then restored to their original Euclidean coordinates, but the polar sorted linked list order is retained.

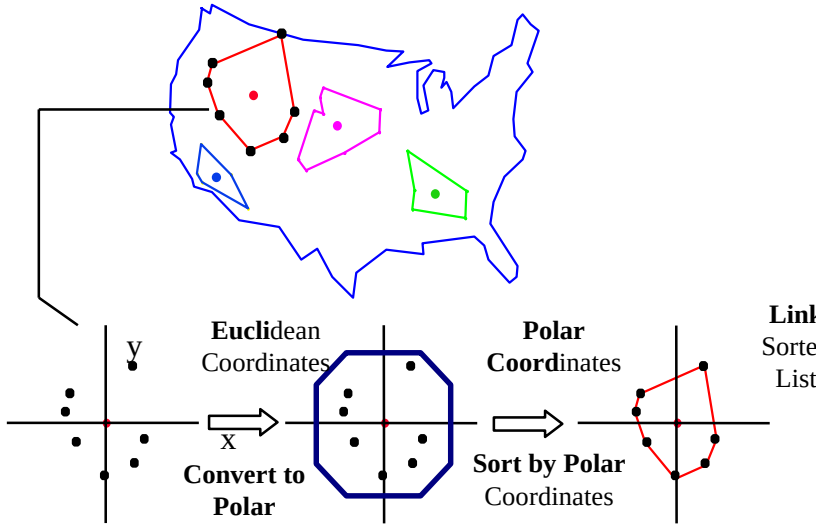

Fig. 2. The Shrink-Wrap Algorithm

The shrink-wrap algorithm thus gives a good first estimate of the node order for an optimum closed-circuit path for each neighborhood. From this we calculate a path length for each neighborhood, and by summing the squares of the ***k*** closed-circuit paths, we can generate a score for each solution set of ***k*** neighborhood attractors.

2.3 Chromosome Representation

A solution to MTSP(n, k) consists of ***k*** closed circuit paths, and may be completely determined for a particular group of ***n*** cities by a vector of ***k*** neighborhood attractors.

The city affiliations for each attractor are determined by the relative parameters of the solution set of attractors, and similarly the closed circuit path lengths are determined using the shrink-wrap algorithm. Since the attractor schema requires three parameters for each neighborhood attractor (**x**,**y**,**a**) with position (**x**,**y**) and attraction constant (**a**), then a solution to MTSP(n, k) can be specified as a vector of length 3*k (each solution consists of k triplets (**x**,**y**,**a**)), where each parameter is a real-valued number between zero and one.

Each neighborhood attractor has an associated linked list of cities (of varying length, possibly including zero length) which define the neighborhood circuit. Genetic algorithms which use mutation operators generally specify a fixed mutation rate, often between 0.001 and 0.1, which is applied to each gene in a chromosome independently of whether any mutations have occurred in other parts of the same chromosome.

2.4 A Menu of EA Designs

A wide variety of evolutionary algorithms have been developed which can be applied to solving optimization problems such as MTSP [11]. In this effort a broad representative sampling of these methods was implemented and applied to MTSP. These techniques may be roughly subdivided into three classes of evolutionary computation techniques: genetic algorithms, evolutionary strategies, and other methods.
The techniques implemented and tested were:

- Genetic Algorithm (μ, λ) using Mutation and Crossover
- Genetic Algorithm (μ, λ) using Mutation Only
- Genetic Algorithm (μ, λ) using Mutation and Crossover, plus Elitism
- Evolutionary Strategy $(\mu + \lambda)$ using Mutation and Crossover
- Evolutionary Strategy $(\mu + \lambda)$ using Mutation Only
- Generational Monte-Carlo Optimization
- Particle Swarm Optimization

Note that the terms “genetic algorithm” and “evolutionary strategy” are used to distinguish (μ, λ) strategies from $(\mu + \lambda)$ strategies, with real-valued representations used for both. The “genetic algorithms” use a strategy with the property that each generation the parent population is replaced by their offspring, commonly known as (μ, λ). When the elitism operator is added, the single fittest member from the parent population is retained in the population the following generation.

The “evolutionary strategies” generate offspring from the parent population, but then the offspring must compete with the parents, hence a $(\mu + \lambda)$ strategy. The populations are concatenated and sorted by score, and only the fittest members survive with the remainder discarded. Since the fittest member is already propagated forward from the parent generation, there is no need to use an elitism operator.

The Generational Monte-Carlo Optimization method used was implemented as a $(\mu + \lambda)$ evolutionary strategy, but with mutation rate of (1.0). During each generation a number of new individuals equal to the population size are randomly generated, and

then they are added to the existing population. The population is then sorted and truncated. During the trials the new individuals were randomly and independently produced, but the evaluation and sorting of them was generational for ease of comparison.

The particle swarm optimization method (described below) defines a population of particles in n-space (where n is the number of genes to be represented) which have position and velocity. The particle positions and velocities are updated based on a combination of parameters including previous states, the global best yet found (analogous to use of an elitism policy), pre-assigned constants, and random factors.

2.5 Particle Swarm Optimization

Particle swarm optimization was first proposed by Kennedy and Eberhart [12] and was inspired by natural biological processes based on the notion of swarm behavior in animal or insect populations. A swarm is defined as a population of interacting elements which is able to achieve some global objective through collaborative search. Swarm optimization relies on the tendency of a swarm population to converge on a center of mass along critical dimensions, resulting in achievement of an optima [13].

In particle swarm optimization (PSO) each individual or chromosome is considered to be a particle in hyperspace having both position and velocity. The particles are then "flown" through hyperspace (a chromosome with n genes would exist in an n-dimensional space), with its velocity and position information influenced by its previous state, its previous best (fittest) state, and the global best for the entire population.

Each particle ***i*** is represented as

$$\mathbf{x_i} = (x_{i1}, x_{i2}, x_{i3}, \ldots, x_{in}) \quad (1)$$

and has velocity

$$\mathbf{v_i} = (v_{i1}, v_{i2}, v_{i3}, \ldots, v_{in}) . \quad (2)$$

The previous best position for each particle is stored, and given by

$$\mathbf{pbest_i} = (p_{i1}, p_{i2}, p_{i3}, \ldots, p_{in}) . \quad (3)$$

The global best for all particles yet seen,

$$\mathbf{gbest} = (g_1, g_2, g_3, \ldots, g_n) . \quad (4)$$

The particle update equations are then given by:

$$\mathbf{v_{i+1}} = w*\mathbf{v_i} + c_1*\text{rand1}*(\mathbf{pbest_i} - \mathbf{x_i}) + c_2*\text{rand2}*(\mathbf{gbest} - \mathbf{x_i}) , \quad (5)$$

$$\mathbf{x_{i+1}} = \mathbf{x_i} + \mathbf{v_{i+1}} \quad (6)$$

where w is inertial weight, c_1 and c_2 are constants, and rand1 and rand2 are random values between 0.0 and 1.0.

When implementing PSO it is important that great care is taken in choosing w, c_1 and c_2, as well as the initial magnitudes of the velocity vectors. A heuristic for annealing w is given by [13], that w should start at 0.9 and be decreased linearly to 0.4 over 1000 generations. In this effort the population generally converged within 100 generations, even using the cited heuristic. Instead, the following parameter values were used:

initial velocity = 0.1 * rand,	where $0.0 < rand < 1.0$
$0.4 \leq w \leq 0.9$,	decreased linearly over 100 generations
$c_1 = 1.0$, $c_2 = 0.1$	

The results of PSO on MTSP are discussed below under **Results**.

2.6 Mutation Operator with Parameter Space Search

The MTSP(n,k) landscape is characterized by plateaus for which mutations often result in offspring with scores equal to those of the parents. This feature inhibits continued progress to the optimum. One approach to address this is to augment the mutation operator with the capability to perform a search in parameter space in order to find a mutation for which the offspring fitness is not identical to that of the parent.

Once a decision is made to generate a mutation (using a generated random number compared with the mutation rate), a mutation delta is "suggested" to the mutation operator. The operator adds the mutation delta to the parent individual and generates an offspring individual. The fitness of the offspring is measured, and if the fitness is different from that of the parent, the offspring is kept. However, if the fitness of the offspring is the same as that of the parent, then the mutation operator begins incrementing the mutation delta, generating and testing offspring until an offspring is generated which has a different fitness from the parent. The parameter search must recognize when it has reached the edge of the parameter space (e.g. edge of the map), after which it may change the direction of delta. The parameter search must also recognize when it is not possible to find an acceptable mutation using that parent, for which it has the ability to select a new parent. The parameter search rule is given:

Parameter Search Rule:
if **fitness(offspring)==fitness(parent)**
then **increment mutation delta by 10%**
(if limit of parameter space reached, change sign of delta)
generate new offspring from parent
(if selected parameter set fully searched and no new)
(fitness is found, then select another parameter to mutate)
repeat until **fitness(offspring)<>fitness(parent)**

3 Results

Results were generated for three different problems: MTSP(30,5), MTSP(20,3) and MTSP(50,3). The different algorithms applied are listed below. Each algorithm was run for 100 generations, with 10 independent runs performed using different random seeds.

- Genetic Algorithm (μ, λ) with Mutation & Crossover (GA-MC)
- Genetic Algorithm (μ, λ) with Mutation, Crossover and Elitism (GA-MCE)
- Genetic Algorithm (μ, λ) with Mutation Only (GA-M)
- Evolutionary Strategy ($\mu + \lambda$) with Mutation & Crossover (ES-MC)
- Evolutionary Strategy ($\mu + \lambda$) with Mutation Only (ES-M)
- Particle Swarm Optimization (PSO)
- Generational Monte-Carlo Optimization (GMCO)

All of the algorithms were at least fairly successful at finding good solutions to each problem. The normalized results of all of the algorithms are shown following in **Figure 3**. The genetic algorithms that did not use elitism performed worst. This is apparently due to the need for high exploration and the loss of good solutions when the parent generations are completely replaced by their offspring. Preserving the best parent and adding it to the offspring generation (as shown with GA-MCE) substantially improves performance, and yields one of the best algorithms overall.

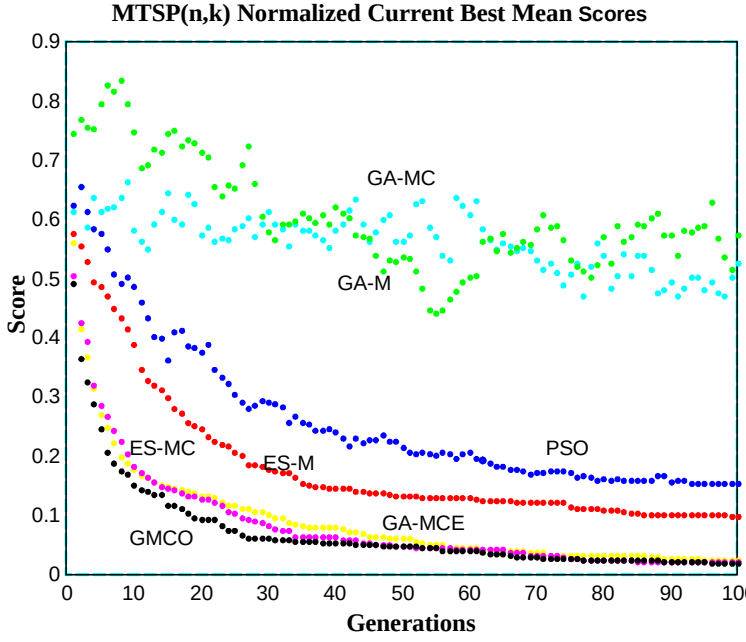

Fig. 3. Normalized Results of All EAs on MTSPs

Particle swarm optimization performed reasonably well, but often failed to locate the global optimum solution. Since this algorithm is quite sensitive to its parameter settings, and comparatively little effort was made to fine-tune PSO for this application, it is quite possible and reasonable that better results could be obtained with PSO with different parameter settings. One obvious target is a different annealing schedule for the inertial weight **w**.

The evolutionary strategies performed better than PSO and the GAs without elitism. The algorithm ES-MC, employing both mutation and crossover, performed significantly better than ES-M, which used mutation only. Addition of the crossover operator seems to provide an effective mechanism for finding better solutions. In fact, the algorithm ES-MC was one of the top performing techniques.

Perhaps the most interesting result is the excellent performance of the GMCO strategy. This algorithm essentially operates as an evolutionary strategy where a number of random individuals are generated each generation, and then the offspring are added to the parents, they are sorted and the fittest members are kept. Thus, this algorithm combines strong elitism along with high exploration. Overall, this technique seemed to offer the best results. An interesting question is whether it would work as well on more complex problem sets with larger values for ***k***, the number of agents.

For each algorithm the chromosome population size was kept at 10. A number of runs (though not the full matrix) were done with higher values of ***k*** and larger population sizes. A strictly subjective observation was that for best performance population size should scale with ***k***. Thus, for 3-5 agents a population of 10 was sufficient, but for ***k***=10, a more suitable chromosome population size was 20-30. This suggests that the complexity of the problem increases not so much with the number of cities, but with the number of sales agents. However, the issues of complexity and scalability need to be studied more thoroughly using larger numbers of cities and various combinations on the number of sales agents and chromosome population sizes.

4 Conclusions and Future Research

In this paper a variation of the Euclidean Traveling Salesman Problem (TSP), the Multiple Traveling Salesman Problem (MTSP), was presented and discussed. This problem is representative of many real-world multi-level optimization problems, and thus offers a platform for development and testing of evolutionary computation paradigms for solving such problems. An approach to solving MTSPs was presented using the Neighborhood Attractor Schema, the Shrink-Wrap algorithm, and a variety of evolutionary computation algorithms and paradigms. The results were encouraging, but also showed that much more could be done to understand these problems and evolutionary computation approaches to solving them.

References

1. Bellman, R.E., "Dynamic programming treatment of the traveling salesman problem", Journal of the ACM, 9:61—63 (1962)
2. Bellmore, M. and G.L. Nemhauser, "The traveling salesman problem: a survey", Operations Res. 16, 538-558 (1968)
3. Karp, R.M., "Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane", Math. of Operations Research, 2:209—224 (1977)
4. Papadamitriou, C.H., "The Euclidean traveling salesman problem is NP-complete", Theoretical Computer Science, 4:237—244 (1977)
5. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G and D.B. Shmoys. The traveling salesman problem. Wiley, New York (1985)
6. Arora, S., "Nearly linear time approximation schemes for Euclidean TSP and other geometric problems", In 38th Annual Symposium on Foundations of Computer Science, pages 554-563, Miami Beach, Florida, 20-22 October (1997)
7. Goldberg, D.E., "Messy genetic algorithms: Motivation, Analysis, and First results", Complex Systems, Vol. 3, pp. 493-530 (1989)
8. Jog, P., Suh, J.Y., and D. Van Gucht, "The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem", Proc. of the 3rd Intl. Conference on Genetic Algorithms, pp. 110-115, Morgan Kaufmann, (1989)
9. Julstrom, B.A., "Insertion Decoding Algorithms and Initial Tours in a Weight-Coded GA for TSP", Genetic Programming 1998: Proc. of the Third Annual Conference, pp. 528-534, Morgan Kaufmann, 22-25 (1998)
10. Hansen, P., and N. Mladenoviæ, “J-Means: A new local search heuristic for minimum sum-of-squares clustering”, Pattern Recognition, Vol. 34 (2) 405-413, 2001.
11. De Jong, K., Course on Evolutionary Computation, George Mason University, (1998)
12. Kennedy, J. and R.C. Eberhart, "Particle Swarm Optimization", Proc. IEEE Intl. Conf. on Neural Networks, pp. IV:1942-1948 (1995)
13. Eberhart, R.C., and Y. Shi, "Evolving Artificial Neural Networks", Proc. Intl. Conf. on Neural Networks and the Brain - Beijing (1998)