Distributed Hybrid Metaheuristics for Optimization

Ben Weber
Department of Computer Science
California Polytechnic State University
San Luis Obispo, CA 93407
bgweber@calpoly.edu

Abstract

A metaheuristic is an intelligent, iterative process that guides a search and can be applied towards optimization problems, such as the Traveling Salesman Problem. Two well studied techniques for solving optimization problems are Genetic Algorithms and Ant Colony Systems. However, each metaheuristic has different strengths and weaknesses. Genetic Algorithms are able to quickly find near optimal solutions, but may end up in local optima. Ant Colony Systems offer an approach that considers a larger portion of the search space, but require well tuned parameters to compete with the convergence rates of genetic algorithms. Traditional heuristics for solving the Traveling Salesman Problem may perform better than these metaheuristics, but require specific knowledge of the problem domain. Using metaheuristics to solve optimization problems allows for generalized problem solving and may be applicable to different problem domains. This paper introduces GoatSpot, a hybrid approach for solving the traveling salesman problem using distributed genetic algorithms and ant colony systems. GoatSpot is used to analyze the performance of a solution sharing, hybrid metaheuristic.

1 Introduction

The Traveling Salesman Problem (TSP) is an optimization problem that consists of finding the shortest closed tour which visits all cities in a given set and is an NP-hard problem to solve optimally. Using an exhaustive search to find the best tour is computationally infeasible for data sets consisting of more than twenty or thirty cities, because it exhibits a factorial run time. There are several techniques that can be used to reduce this runtime, such as branch and bound [8]. These algorithms reduce the search time to an exponential growth, but currently there are no known polynomial algorithms for solving the traveling salesman problem optimally.

The traveling salesman problem has several practical applications, such as robotic movement and vehicle routing. However, such applications do not always require an optimal solution to the TSP. A solution that is within five percent of the optimal solution may be suitable. Consider a robot in a factory line that must weld several locations. As long as the robot can finish the welding within its allocated time, then an optimal solution is not absolutely necessary. However, if the robot is the limiting factor in the factory line, then an optimal solution should be determined.

Another practical application of the TSP that does not always utilize the optimal solution is vehicle routing. Consider a vehicle that must visit several locations everyday. A branch and bound technique could be used to determine a tour to visit the cities, but it may take longer than a day to determine the optimal tour. Therefore determining an
optimal solution would be impractical. There may be additional constraints applied to these real world applications. For example, the vehicle might have designated times to visit the locations. This problem is known as the traveling salesman problem with time windows and is NP-hard as well.

Several different heuristics have been developed to quickly find solutions for the traveling salesman problem, including Nearest Neighbor, Greedy, Clarke-Wright, and Christofides. All of these techniques have been specifically developed for the traveling salesman problem and are not applicable to other optimization problems. For example, the nearest neighbor algorithm builds tours by adding the closest city to the current one until all cities are added. Each of these algorithms completes in polynomial time, but has a worst case ratio of up to two. Due to the unpredictable accuracy of these algorithms, they are not suitable replacements for an exhaustive search.

There is a class of algorithms known as metaheuristics that can be applied towards these optimization problems. A metaheuristic is an iterative algorithm that guides the search process using a form of intelligent reasoning. Most of these techniques do not guarantee optimality and do not have a definite run time, instead they run until a predefined criterion is met. Using these algorithms, it is possible to discover close to optimal solutions in polynomial time. Metaheuristics have been applied to several problem domains including the traveling salesmen problem. However, they require some specific knowledge of the problem domain and may not be suitable for all optimization problems.

Metaheuristics must consider several tradeoffs, the first being a tradeoff between optimality and runtime. This is a tradeoff inherent to heuristics as well, but the worst case ratio for metaheuristics is unknown. Metaheuristics must also consider a tradeoff between exploration and exploitation of the search space. Exploration is the ability of an algorithm to discover the majority of the search space, while exploitation is the ability to improve a specific solution. An algorithm that explores a large portion of the search space is more likely to find an optimal solution, but takes longer to converge. An algorithm that exploits a solution space will quickly converge, but may get trapped in local optima. Usually the behavior of a metaheuristic can be changed by adjusting parameters, but these parameters may be problem specific.

This paper introduces GoatSpot, a metaheuristic designed to improve the exploration and exploitation of a search space using a distributed, hybrid approach. The algorithms used by GoatSpot share solutions in order to simultaneously diversify and intensify the search. Thus the contribution of this paper is to analyze the performance of a solution sharing, hybrid metaheuristic.

2 Related Work

Several different techniques have been developed to quickly find near optimal solutions to the TSP. The best typically get within roughly 10-15% of optimal in relatively little time [9]. There are heuristic based approaches such as 2-Opt and 3-Opt [9], which can be used to improve a solution. Another heuristic, known as nearest neighbor, exhibits \( O(n^2) \) run time. This heuristic sometimes produces optimal results, but is usually an unsuitable choice for solving the TSP. Metaheuristic searches are another approach for solving optimization problems. Metaheuristics may make use of domain specific knowledge in the form of heuristics that are controlled by the upper level strategy.
A metaheuristic algorithm is iterative in nature and attempts to improve the best known solution. This paper explores two well-known metaheuristic algorithms for the TSP: genetics algorithms (GA) and Ant Colony Systems (ACS).

Genetic algorithms solve optimization problems by killing off weak solutions and breeding the stronger solutions. Genetic algorithms represent solutions as chromosomes and perform crossovers and mutations on these chromosomes. The goal of these algorithms is to evolve solutions into the optimal solution. Genetic algorithms can be applied to the TSP by encoding chromosomes as the order that cities are visited. However, there are complications when performing crossovers with chromosomes representing tours, because every city must be visited exactly once. The convergence rates of several different crossover techniques are explored in [11]. Genetic algorithms often converge to local optima, since the offspring generated by crossovers exhibit characteristics of the parents. A fast TSP solver is presented in [13], which uses 2-Opt to mutate solutions and a crossover function to avoid local optima. Genetic algorithms appear to be effective in exploiting a particular solution, but have difficulty exploring the solution space. Genetic algorithms have also been applied to solving the TSP with time windows [10], by extending the fitness function to consider distance and lateness costs.

Another metaheuristic for solving the TSP is modeling ants [2]. Ants are known for their ability to search for food and avoid obstacles. Artificial Ants can be used to solve optimization problems by simulating the pheromone trails left by actual ants. The higher the pheromone level, the more likely an ant is to follow a specific trail. Artificial ants deposit pheromones according to problem-specific rules. An implementation of an Ant Colony System for solving the TSP is presented in [3]. ACS has useful properties beyond the behavior of real ants. For instance, artificial ants have been used to solve the TSP with time windows [6]. A modified version of ACS known as elitist ACS has shown improved results by intensifying searches where the best solutions are found. However, elitist ACS does not share the same reasons for success as real ants [5], because of the extended behavior of elitist ants. Ant colonies are effective in exploring a solution space, but exhibit slow convergence rates.

Genetic algorithms and ant colony systems are effective in solving optimization problems, but each technique must make a compromise between exploration and exploitation. Genetic algorithms are usually much faster than ant colonies, but are more likely to get trapped in local optima. Ant colonies effectively search the solution space, but often require problem-specific parameters. Hybrid approaches attempt to overcome the limitations of these metaheuristics. A hybrid approach in [7] combines genetic algorithms and ant colony systems for faster and better search capability. Other techniques use genetic algorithms to modify parameters used by the ACS, because different problem instances have different optimal parameter combinations [4]. Methods have also been developed that use a genetic algorithm to evolve a population of genetically modified ants [12]. These hybrid approaches overcome some of the tradeoffs inherent to metaheuristics.

3 Metaheuristics for the TSP

Ant colony systems and genetic algorithms are two very different metaheuristics, but both are suitable for solving the traveling salesman problem. Both are iterative and attempt to improve a known solution. They start with random solutions, but converge to
near optimal solutions. Both are based on natural concepts: Ant colony systems simulate real ants, while genetic algorithms simulate evolution. This section first discusses the genetic algorithm component and then the ant colony component of the hybrid metaheuristic. The convergence rates of the individual algorithms are evaluated in section 4, and the approach used to combine the metaheuristics is discussed in section 5.

3.1 The Hybrid Genetic Algorithm

The genetic algorithm developed for the hybrid approach is greedier than the algorithms presented in [10], [11], and [13]. Unlike most of these algorithms, the hybrid approach does not use the 2-Opt method. The goal of the genetic algorithm presented in this paper is to quickly converge to the local optima of a given solution, as the algorithm is designed to be adaptive to solutions. This algorithm produces poor results when run independently, because it requires an additional algorithm to direct it.

The first step of the genetic algorithm is creating an initial population. There are two aspects to consider when generating an initial population: size and diversity. A large enough population size must be used, otherwise a solution may converge prematurely. However, too large of a population may result in a genetic algorithm with poor convergence. Since the hybrid genetic algorithm is designed to converge quickly, a relatively small population is used. The size of the population is equal to one fourth the number of cities, this value was determined experimentally. The diversity of an initial population is important, because the population largely determines the direction of the genetic algorithm. If initial solutions are too diverse, then the genetic algorithm will have difficulty breeding stronger offspring. However, too similar of a population produces a genetic algorithm trapped in small part of the search space.

The simplest approach to create an initial population is to generate completely random solutions. Eventually the genetic algorithm will converge, but many of the initial iterations are wasted due to the poor solutions generated. It is possible to utilize other heuristics when generating an initial population. The initial population generated by the hybrid genetic algorithm is partially greedy and random. Initial solutions are generated by selecting a random number of cities. The solution starts at city zero and chooses the closest city in the randomly selected set. This process is repeated until all cities in this set are added to the solution. The remaining cities are then randomly inserted into the tour. This method creates diverse solutions, even though a greedy heuristic is used. The effects of using a completely greedy initial solution are analyzed in section 4.

Each iteration, the genetic algorithm must determine which solutions from the population to breed and which solutions to replace. The hybrid genetic algorithm always replaces the weakest solution, but selecting the solutions to breed is more random. Each solution is assigned a weight, equal to the inverse of its distance. Then two of the solutions are randomly picked based on these weights. Therefore better solutions are more likely to get picked, but not guaranteed. If the strongest solutions were always bred, then the genetic algorithm would fail to traverse through the search space. In addition to breeding the solutions, the genetic algorithm will mutate a random solution. Solutions are mutated by switching two randomly selected cities. However, this mutation is not significant enough to avoid local optima.

Solutions are bred using a greedy crossover. A crossover point is randomly selected, which determines how many cities the child will inherit from the first parent.
All of the cities up to the crossover point are added to the child. Selecting the cities to inherit from the second parent is more difficult, because duplicate cities must be avoided. The child inherits cities starting at the end of the second parent. If the child already contains the city, then the closest remaining city is selected. This process is repeated until all of the cities have been added to the child. This crossover does not prevent the genetic algorithm from getting trapped in local optima, because of its greedy nature.

3.2 The Hybrid Ant Colony System

The hybrid ant colony system is based on the algorithm presented by Dorigo [3] for solving the traveling salesman problem. The pheromone levels between cities are stored as an n by n matrix, where n is the number of cities. The pheromone levels are initially set to zero. There are three levels of pheromones that can be left by ants: low, medium, and high. A medium amount of pheromones is four times greater than a low amount, and a large amount is twice as large as a medium amount. The number of ants used is equal to the number of cities, and the ants are initially placed at random cities. Each iteration, every ant travels to a new city according to the city selection heuristic. At the end of the iteration, the pheromone levels are decreased by a constant rate. Once an ant completes a tour, a low amount of pheromone is deposited on the ant’s tour and a new ant is spawned at a random city. If the ant discovered a new best solution, then an additional medium amount of pheromone is deposited. Eventually the pheromone levels along a certain path will be sufficiently large, causing the ant tours to converge.

The main heuristic used in the ant colony system is city selection. An ant chooses which city to visit next based on the amount of pheromones leading to each city and the distance to each city. Each city is assigned a weight according to the equation:

\[
weight_i = \left(\frac{\text{longest distance}}{\text{distance}_{ji}}\right) \times (\text{pheromones}_{ji} + 1)
\]

In this equation: i is the city being considered, j is the current city, distance_{ji} is the distance to city j from i, and pheromones_{ji} is the pheromones along the path from city j to i. The longest distance variable is the largest distance in the problem set and is used to normalize the values. Once the weights for each city are calculated, a city is randomly selected based on the weights. The ACS metaheuristic results in large coverage of the solution space, because of the random nature of ants.

4 Convergence of the Metaheuristics

To determine the nature of the hybrid metaheuristics, the convergence rates were analyzed. The first experiment evaluated the ability of the metaheuristics to adapt to a solution and determined if sharing solutions between algorithms could lead to improved searching. The test was conducted on a Pentium 4 2.4 GHz PC and allowed each algorithm to run for three seconds. The problem set contains forty cities and was designed by Dumas and Solomon. Both of the algorithms were rewritten to accept an initial greedy solution to simulate the effect of receiving a solution from another algorithm.

The genetic algorithm and ant colony system metaheuristics required changes to accept an initial greedy solution. Adding this modification to the genetic algorithm was
straight forward, because a greedy solution was generated as part of the initial population. However, the ant colony required a slightly different approach. A solution was generated using the greedy heuristic and a large amount of pheromones were deposited along the tour generated. The greedy solutions were generated using the nearest neighbor algorithm. The nearest neighbor heuristic produces solutions much better than randomly selecting cities, but is still far from optimum. The effectiveness of this technique relies mainly on the symmetry of the problem.

![Convergence of Dumas40](image)

**Figure 1 - Convergence of GA and ACS**

Figure 1 displays the two unmodified algorithms along with the modified version of each algorithm. The results indicate that starting with a greedy solution improves the convergence rate, although the solutions do not reach optimality. The optimal distance for this problem was 254, but the metaheuristics were only able to find a minimal distance of 265. There is an interesting distinction between the two algorithms; the genetic algorithm converges much quicker than the ant colony system, even though they eventually discover almost equivalent solutions. It appears that the genetic algorithm is a better metaheuristics for solving the TSP, since it converges almost instantly. Also, both algorithms were able to improve upon the solution generated by the greedy heuristic. This experiment suggests that sharing solutions between different algorithms may result in better and faster searches.

## 5 A Distributed Approach

This paper introduces GoatSpot, a hybrid approach for solving the traveling salesman problem using genetic algorithms and ant colony systems. Previous hybrid
approaches mainly used genetic algorithms to develop more intelligent ants and fine tune
the parameters necessary for fast convergence. This hybrid approach differs from
previous work [4], [7], [12] involving genetic algorithms and ant colony systems, because
the algorithms are distributed. Ant colonies and genetic algorithms work independent of
each other and only communicate when better solutions are discovered. Ant colonies are
used to explore the solution space, while genetic algorithms are used to improve the
convergence rate of the search.

A distributed approach is suitable for the traveling salesman problem, because the
problem is CPU intensive. The only I/O operations are used for opening the problem set.
A 200 city problem uses less than sixteen megabytes of RAM for the entire program.
Additionally, there is limited network bandwidth used, because only improving solutions
are reported. These properties allow the algorithm to scale to many machines, utilizing
free processor cycles. The distributed algorithm will fully utilize the processor, but could
be rewritten to share it as well. The algorithm could give up the processor after a certain
number of iterations.

GoatSpot was developed as a client/server model using Java sockets. The server
is executed from the command prompt and requires input parameters specifying the
number of GA and ACS clients. The server is responsible for determining the search
parameters for each client, as well as facilitating all communication. The server opens a
port and waits for clients to join. When a client connects to the socket, two new threads
are created: one for receiving data from the socket and one for sending data to the socket.
Once the specified number of clients have connected, the server sends each client the
search parameters specifying the type of metaheuristic the client should use.

Once the search begins, the receive threads wait for solutions to be reported by
clients. When a solution is received, the server determines which clients should receive
the solution and adds the solution to the output queue for the chosen clients. Clients are
chosen in a round-robin fashion, which evenly distributes the solutions. The main
purpose of this technique is to avoid bottlenecks at the beginning of the search, when
several new solutions are discovered. An additional client is chosen by randomly

Figure 2 - GoatSpot Architecture
selecting one of the genetic algorithm clients. Sending the genetic algorithms additional solutions helps avoid local optima.

The client program is comprised of the hybrid algorithms and two additional threads used to send and receive solutions to and from the server. After connecting to the server, the client waits for the search parameters. Once received, the client initiates the search for the specified time period. When a client receives a new solution, it must react to the new knowledge. When a genetic algorithm client receives a new solution, the weakest solution in the population is replaced by the received one. Within a few iterations, the genetic algorithm will either converge to the new solution or replace it with new offspring. When an ant colony client receives a new solution, a medium amount of pheromones are deposited along the tour. The new tour may or may not be utilized depending on the current pheromone levels. Once a specified amount of time has expired, the client informs the server and closes.

6 Performance of the Distributed Approach

The performance of GoatSpot was tested on four different problem sets consisting of 200 cities: all of the cities in a circle, four concentric circles, a 20 by 10 grid, and 200 randomly placed cities. Experiments were conducted in a homogenous environment with 21 Pentium 4 2.4 GHz PCs running Windows XP. One PC was setup as a dedicated server, while tests were performed with 1, 4, and 20 client machines. The different problem sets were solved using different ratios of GA and ACS clients. Each test ran for two minutes. The results of the search were stored by the server.

Figure 3 - Discovered Tours

Figure 3 displays the best solutions discovered by GoatSpot. Optimal tours were found for the circle and concentric circle problems, but not for the grid and random problem sets. The shortest grid tour discovered was 225, a difference of 12.5% from the optimal solution. The shortest tour discovered for the random problem set was 166, but the distance of the optimal solution is unknown. The tour for the random problem is non-optimal, because it could be improved by the 2-Opt method. From these results, it is
obvious that the solutions discovered by GoatSpot are worse than solutions discovered by problem specific heuristics such as 2-Opt. However, all of the solutions were within 15% of optimal.

![Grid 200 city Problem](image)

**Figure 4 - Metaheuristic Performance**

The results recorded by the server for the 200 city grid are displayed in figure 4. The tests performed with more than a single client used an equal number of GA and ACS clients. Excluding the ACS outlier, all of the searches were trapped in local optima within 10 seconds. The remaining runtime did not offer any improving solutions, since none of the algorithms could improve the tour with distance 225. Increasing the number of clients did not lead to the discovery of better solutions or faster convergence. It appears that the single ant colony system was at outlier, because it was the only search without a genetic algorithm. Considering the initial steepness of the lines, it is unexpected that the algorithms level out so quickly. However, the steepness of the lines might be the result of network latency.

Although the traveling salesman problem is suitable for a distributed application, GoatSpot did not take advantage of the large amount of resources. GoatSpot used a minimal amount of network bandwidth, but this was part of the design. The number of messages transmitted is displayed in figure 5. During the tests every processor was fully utilized, but there were not noticeable gains between different numbers of clients. One computer running a genetic algorithm was just as effective as twenty machines running genetic algorithms. This is most likely due to an overlap of solutions explored by different clients. It would be beneficial if the algorithms had knowledge of the previously explored search space.
7 Conclusions and Future Work

Overall, the distributed, hybrid approach offered almost no improvement over a single genetic algorithm. There are several possible reasons for this lack of performance increase. The first reason is the genetic algorithm’s superior performance to the ant colony system for the traveling salesman problem. The ant colonies were unable to improve the best known solution, because they lagged behind the genetic algorithms. The solutions produced by ant colony systems were most likely ignored by the genetic algorithms, due to this delay. Therefore the ants were unable to help the genetic algorithms fully explore the search space, rendering the hybrid approach useless.

The second shortfall of the distributed metaheuristic was the way in which the algorithm was distributed. The search was distributed by running independent searches and sharing solutions. This resulted in a large overlap of work, since increasing the number of clients had minimal effect on the performance of the search. The solution sharing, distributed approach appears to be useless, unless metaheuristics have knowledge of which parts of the solution space have already been explored. However, sharing solutions defeats this purpose, because it results in an overlap of work. This appears to be a fundamental flaw of this distributed technique. Future work will explore other possible methods for distributing GA and ACS metaheuristics.

GoatSpot was designed to utilize the strengths of different metaheuristics in order to find better solutions. However, optimal solutions were not discovered for the grid or the random problem sets. This is most likely due to the limitations of the genetic
algorithms and ant colony systems used by GoatSpot. Individually the metaheuristics were unable to find the optimal solutions and combining them did not help. GoatSpot failed to overcome the tradeoff of exploration and exploitation inherent to metaheuristics.

Originally GoatSpot was designed to solve the traveling salesman problem with time windows, but the GA and ACS metaheuristics were unable to converge to solutions. Ant colony systems were unable to solve the TSPTW, because the additional constraint of time windows is not a behavior displayed by real ants. Pheromones are bidirectional, since ants do not know which direction the previous ants traveled. However, the TSPTW requires unidirectional pheromones to be effective. Genetic algorithms can be modified to handle time window constraints by extending the fitness function. However, complications arose with the crossover function. The greedy crossover used by the hybrid genetic algorithm was unsuitable for time windows. It appears that adding constraints to an optimization problem requires vastly different metaheuristics.

One of the purposes of GoatSpot was to determine if there is a tradeoff between domain specific knowledge of a problem and a large amount of processing power. Both GA and ACS used simple upper level strategies to explore the search space. Genetic algorithms combined the nearest neighbor heuristic with randomness, while ant colony systems utilized a simple city selection heuristic. Additional modifications could have been added to these metaheuristics in order to improve the ability to solve the traveling salesman problem, such as a 2-Opt mutation. However, GoatSpot attempted to use brute force rather than problem specific knowledge to solve the traveling salesman problem. GoatSpot failed to outperform 2-Opt, even with twenty times the computational power. Metaheuristics might not be suitable for generalized problem solving, because there is not necessarily a tradeoff between domain specific knowledge and brute force.
8 References


