Ant Colony Optimization for the Traveling Salesman Problem

Max Bucci, Megan Maher, and Nikki Morin

April 05, 2015

1 Introduction

In this work, we use Ant Colony Optimization (ACO) to find the shortest tour of a collection of virtual cities. ACO takes its inspiration from the foraging behavior of ants and uses swarm intelligence and probabilistic technique to search for the optimal path between all the nodes in a graph. We implement two variations of the ACO algorithm: Ant Colony System (ACS) and Elitist Ant System (EAS). We then perform experiments to test the effect of multiple parameters on the performance of each of our systems, and how the systems compare to each other overall.

2 Ant Colony Optimization

Ant Colony Optimization (ACO) is an algorithm modeled from the behavior of ants seeking paths between a colony and a food source. During this process, ants lay down trails of pheromones to indicate which paths they have chosen, and use preexisting pheromones to help determine the course of their paths; over time, the amount of pheromone accumulates on path edges that are travelled multiple times. The pheromone naturally decays, meaning older paths are less likely to be followed. This results in a positive feedback cycle that leads ants to increasingly better paths. The formative idea of the algorithm is that path edges which consistently appear in the low cost solutions are likely to be present in the optimal solution.

ACO employs swarm intelligence methods that are especially suited to solving problems with no central coordination or external guidance, collectively finding an optimal solution that would usually escape an individual. The ants work in parallel and independently of each other, and their only communication is indirect, namely through the presence of pheromones. The diffusion of pheromones over time helps ensure that less efficient choices made earlier become less and less influential in the path finding process as the iterations increase.

Several heuristics exist to tailor the performance of the ACO algorithm. There is the number of ants, the number of iterations, α is the degree of influence of the pheromone component, β is the degree of influence of the heuristic component, ρ is the pheromone evaporation factor, ϵ and τ_0 are the factors controlling the wearing away of pheromones in ACS, e is the elitism factor, and q_0 is the probability that ACS will choose the best next edge instead of choosing probabilistically.

Solution tours are constructed iteratively according to a probabilistic method that an ant uses to choose the next edge of its tour. An ant may not visit an edge that it has already visited, but for each unvisited edge that connects to the current city the ant is at, the ant will choose an edge with probability:

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{h \in J^k}^H \tau_{ih}^\alpha \eta_{ih}^\beta}$$

,

where p_{ij}^k is the probability of ant k choosing the edge between cities i and j, α and β are the parameters mentioned previously, τ_{ij} is the pheromone level on edge ij, η_{ij} is the visibility (or the inverse of the length) of edge ij, and in the denominator, we are calculating the sum of the (pheromone$^\alpha$ $*$ visibility$^\beta$) for every edge that has not been visited and connects to the current city. Therefore H is all edges and h is in J^k, or the unvisited edges of ant k at it's current city. This probability is calculated the same for both of our implemented systems: Elitist Ant System and Ant Colony System.

While ACO algorithms are not necessarily guaranteed to find the shortest path in a very complex and high dimensional graph, they find satisfying solutions with comparatively little computation. At the completion of the tour, the entire path is retraced and the pheromone update strategies for the chosen implementation of the ACO algorithm are applied. This is where the implementations of EAS and ACS diverge.

2.1 Ant Colony System

Ant Colony System (ACS) is a simplification of the Ant-Q algorithm, an ACO algorithm combined with reinforcement learning rules from Q-learning. ACS is as equally powerful as the Ant-Q algorithm, but less complex. In ACS, pheromones are evaporated on every edge of the graph according to the wearing away heuristic (ρ). The new pheromone level for every edge is:

$$\tau_{ij} = (1 - \rho)\tau_{ij},$$

where τ_{ij} is the pheromone level on edge ij and ρ is the variable previously described.

Additionally, pheromones from the best tour so far are put down, multiplied by ρ. Most uniquely, pheromones are worn away during the actual tour construction: whenever an ant chooses an edge to add to its tour, the pheromone of that edge decreases according to the equation with the τ_0 term. This is to simulate more realistically the true process of ant foraging, where an ant that travels a particular edge will unintentionally take some of the pheromone with it when it goes. The effect increases total exploration of the graph.

These factors mean that after all ants have completed their tour, the pheromone level on each edge in the graph should be:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bsf},$$

for all variables previously described, bsf as the best found tour so far, and ant k in the range 1 to m, the total number of ants.

2.2 Elitist Ant System

In an Elitist Ant System (EAS) pheromones are evaporated on every edge of the graph in the same manner as ACS. However, in contrast to ACS, in EAS pheromones from each ant are put down: the amount put down is denoted in the formula as $\Delta\tau$ for ant k and edge ij, and is computed as the inverse of the length (L) of ant k's current tour:

$$\Delta\tau_{ij}^k = 1/L^k$$

Finally, pheromones from the best tour so far are put down, multiplied by the elitism factor denoted by ϵ. This means that after all ants

have completed their tour, the pheromone level on each edge in the graph should be:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + \epsilon\Delta\tau_{ij}^{bsf},$$

for all variables previously described, bsf as the best found tour so far, and ant k in the range 1 to m, the total number of ants.

2.3 Travelling Salesman Problem

In our work, we employ ACO to solve an instance of the Travelling Salesman Problem (TSP). The traveling salesman problem is a combinatorial problem that asks Given a list of cities and the distances between each pair of cities, what is the shortest possible tour that visits each city exactly once?. TSP is NP-hard, meaning that there is currently no known solution to the problem that can be resolved in polynomial time. For this reason, it is a prime application for ACO, which, while not guaranteed to find the optimal solution, can find satisfactory solutions with comparatively low computation.

3 Experimental Methodology

To test our implementations of Ant Colony System and Elitist Ant Colony we chose to begin with the rule-of-thumb parameter configuration and test variations from that. The rule-of-thumb configuration is as follows:

- number of ants = 10-30 (we chose 10)
- $\alpha = 1$
- $\beta = 2 - 5$ (we chose 5)
- $\rho = 0.1$
- $epsilon = 0.1$
- $q_0 = 0.9$
- $e = 5$ (for Elitism)

We treated this parameter configuration as our control test and measured subsequent tests in terms of this, as well optimal length. These tests included changing the number of ants and the value of rho, running these across four unique TSP problems. Our TSP problems had between 2000 and 3000 cities (specifically, 2103, 2152, 2319, and 2392 cities). On commodity machines these problems required 10-30 minutes to complete, for 10+ ants and 25 iterations. Because of this relatively large computation time, we chose to take three run averages for each parameter configuration. Three because it gives a decent sense of average performance while allowing for a larger range of parameter configuration as compared to 5 or 10 runs per configuration.

Ant Colony System and Elitism were each tested with the same five parameter configuration, all variances of the rule-of-thumb configuration. In addition to the control mentioned above, we had 20 and 30 ant variations, as well as rho values of 0.01 and 0.5. As we were somewhat limited by computation time, we chose to vary ant count and rho because we believed these two variables (keeping number of iterations constant) would have the greatest effect on performance. Running these five parameter configurations took an entire night each to run for ACS and Elitism.

4 Results

As the figures below show, increasing the number of ants improved performance (decreased tour length) across the board. But, had the largest impact in the Elitism System. The explanation for this should be trivial, more ants gives more possibilities of finding a shorter tour. In general though, the number of ants didnt have a large impact on performance making, at most, a 0.5% difference in ACS and 4.2% in Elitism. These results both result from the pr3292 problem between 10 and 30 ants.

a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a

Additionally, a rho value of 0.5 was the best performing configuration in all tests. This result is surprising as it goes against the rule-of-thumb configuration, which has a rho value of 0.1. As rho is the evaporation factor, a possible explanation for this difference is diversity. Since we are running on relatively few iterations and not finding the optimal length, a higher value of rho allows for more diversity across iterations, taking "bigger steps" between iterations as opposed to creeping towards an optimum.

a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a

The most significant result though is the performance of ACS versus Elitism. As the figure below shows, ACS seriously out performed Elitism in all tests. ACS achieved an average best of 90.8

Finally, our tests showed Elitism to be more sensitive to parameter changes than ACS. As can be seen in the tables below, ACS is more or less constant across its five tests. Elitism, however, has much greater variation in its performance. For a given problem, ACSs best and worst tour length varied at most 18000 or 3.8% for pr3292. Similarly, Elitisms best and worst varied 85000 or 15%, also for the pr3292. This contrast between best and worst could again be due to Elitisms best-tour update rules. As the worst had a low rho and best a high rho, the increased evaporation may have countered the reliance on best-tour. But, as mentioned earlier it is hard to make hard conclusions without further testing, something we would have done with more time and/or more powerful computers.

5 Further Work

In our work, we have explored only two variations of ACO algorithms: ACS and Elitist Ant System. However, there are multiple other variations that would be interesting to explore. We discuss how we could have designed further tests in our experiments, discuss the MAX-MIN Ant System, and explore other applications of ACO.

5.1 Extensions of our experiments

We have chosen to test the effect of the number of ants and ρ on the performance of ACS and EAS. However, there are other parameters that we have used in our algorithms that would be worthwhile to adjust in order to test their contribution to the performance of the systems. We could test the effect of β, which is the degree of influence of the heuristic component, especially with tests of α, which is the degree of influence of the pheromone component. Additionally, although we tested the effect of ρ and the number of ants on the performance of our systems, we were unable to fully test all the ant numbers that we wanted, due to time constraints. Although we didn't see a large difference between the ant numbers we tested, perhaps with a larger number of ants and more time, we could see a larger effect on performance. Additionally, with more time, it would be very interesting to vary the number of tours to see how they affect the success of the program.

5.2 MAX-MIN Ant System (MMAS)

In the MAX-MIN Ant System, all pheromone values are explicitly restricted between two values. These two restriction values can be determined dynamically with the following values:

$$\tau_{max} = \frac{1}{\rho L^{bsf}}, \tau_{min} = \frac{\tau_{max}}{c},$$

where τ is the pheromone level, ρ is the phereomone update factor passed as a variable, L^{bsf} is the length of the best found tour so far, and c is a parameter.

If we do not wish to determine L dynamically,

we can substitute L^{bsf} with L = length of optimal tour, if this value is known.

The pheromone update rules are slightly different than in some of the typical ACO algorithms, where each edge is updated according to the equation:

$$\tau_{ij} = (1 - \rho) * \tau_{ij} + \Delta\tau_{ij}^{best}$$

with the same parameters above, where τ_{ij} is the pheromone level on edge ij. $\Delta\tau_{ij}^{best} = 1/L_{best}$ when the edge ij was in the best ant's tour, otherwise, $\Delta\tau_{ij}^{best} = 0$. This means that only the ant that has found the best tour so far will be placing additional pheromone on its best-so-far tour edges.

With this new MAX-MIN system, we could measure the effect that extremely high or extremely low pheromone levels have on the success of the system. By restricting pheromone values to a specific range, we avoid these extremel high or low pheromone levels that could weight ant decisions in a way that would discourage exploration.

5.3 Other Applications

We have chosen to use our ACO algorithms to solve the traveling saleman problem (TSP). However, there are numerous other application of this optimization algorithm. Such applications include:

- Multi-Knapsack Problem:
  Given a set of n items and a set m knapsacks, where each item has a weight, w, and a profit, p, and each knapsack has a capacity, c, how can we distribute the items so that each knapsack has the maximum total possible profit, without exceeding the capacity of that knapsack. This is the same as saying:
  maximize:
  $$\sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij}$$
  Where x is 1 if item j was assigned to knapsack i, or x is 0 otherwise.

- Software Project Scheduling Problem:
  For software development projects, proper scheduling is key. Nowadays, reducing the time and cost of a project by even a small margin is worth pursuing. This problem faces the issues of task scheduling and human resource allocation. WIth increasingly larger numbers of tasks and employees, this problem becomes NP-hard, and therefore ACO is useful in finding a satsifying solution in an acceptable amount of time.

- Quadratic Assignment Problem:
  In this problem, we are given a set of n facilities and n locations, with a distance between, d_{ij}, for each pair of locations i and j, and flow, f, between pairs of facilities. In this problem, we aim to find locations for all the facilites so as to minimize the sum of all the distances * appropriate flow:
  $$\sum_{p=0}^{P} d_p * f_p$$
  for pair $p = i$ to j, in the set of all pairs, P.

- Vehicle Routing:
  This problem considers a total number of customers, and aims to send a fleet of vehicles on separate routes to deliver to the customers, minimizing the total distance trav-

eled. Each fleet must start and end at the home node.

6 Conclusions

We have implemented Elitist Ant System and Ant Colony System, two variations of Ant Colony Optimization algorithms, and in our experiments applied them to examine their ability to solve the Traveling Salesman Problem. Additionally, with both algorithm implementations we have conducted experiments varying the number of ants and value of ρ, in order to observe the effect of these parameters on the performance of the algorithms. Overall, we noticed that ACS outperformed EAS, consistently finding solutions of length closer to the optimal length for that problem. We also observed while varying select parameters that changing the number of ants in our experiments tended to have minimal effect on the performance of the algorithm for both EAS and ACS. However, higher values of ρ improved performance while lower values decreased performance.