Ant System Optimization

Joshua Hernandez

6852561

PHYS 4250 : Final Project Report

Introduction

Ant colony algorithms are being used to tackle difficult combinatorial problems. They are currently being used for vehicle routing, network communications and many other types of problems.

They are based off the idea of real ants. They are social and live in colonies. Their behaviour is focused towards the survival of the colony as a whole rather than the individual. The behaviour that is studied is where they look for food. The ant optimization is completely derived from this behaviour.

An ant will go around looking for food, while doing so it will leave behind a trail of pheromones. Then when it finds a food source, it will take some back to the nest while leaving more pheromones behind. The other ants will be influenced by the pheromones left behind and will be attracted to the path the initial ant took. Thought they may not precisely follow the pheromone trail left behind. An important emergent behaviour is the tendency to find the shortest path from a food source and the nest.

In a controlled experiment, ants tend to converge to one path. The experiment consisted on a two bridges leading to the same food source. In the case that the bridges are the same length, the and will eventually predominantly use one of them by random choice. This is a result of some ants a few more ants picking a path over another. Then the pheromone deposits build up and the ants are more attracted to that path. In the case where one path is longer than the other, the ant will lean toward the shorter path.[1]

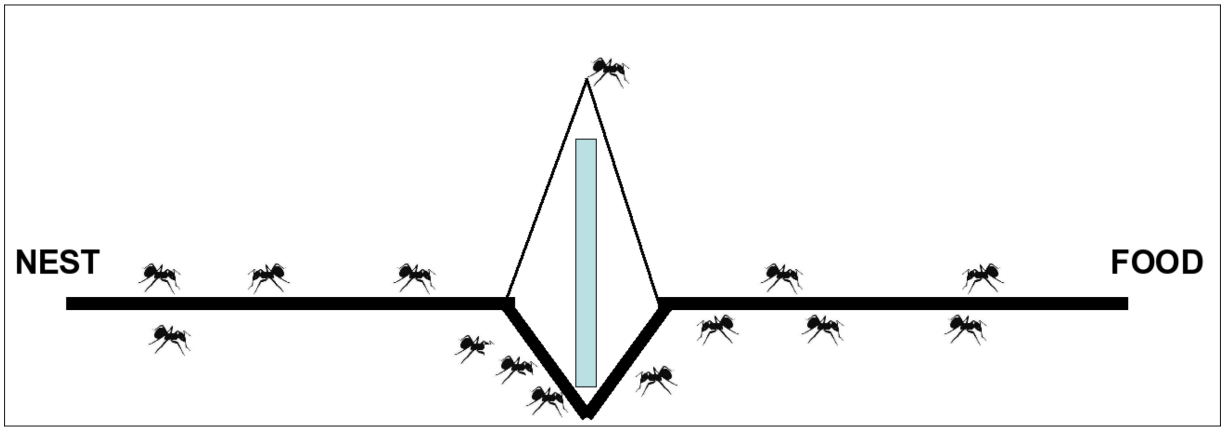

Figure 1 - Binary Bridge Experiment

This of course can be generalized to more complex system, with various paths. In theory the ants will be able to find the shortest path, or at least an optimal path. This is the outline to optimization techniques, in where an individual objects called ants, contributes to the discovery of an optimal

[1] Deneubourg, J.-L., Aron, S., Goss, S., & Pasteels, J.-M. (1990). The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior, 3, 159–168.*

solution. The interaction between the ants and the ability to “passively” find the shortest path, is an emergent behaviour that is not completely understood.

To emphasise the pheromone act as a type of local communication between ants. Ants will affect a particular segment in a path, and only other ants who will be considering that path as a next step will be see it. Also there is a evaporation component to the pheromones. And will slowly forget about bad solutions. If done with care and proper implementation, the results could be better than that of other population-based algorithms.[2][3] As the best path will be re-enforced quickly and in the case where a bad path becomes better than all the others (due to odd fluctuations and anomalies) will eventually be forgotten. One must be careful as to how one implements these features as to not get stuck in local minima.

Ant Colony Method

In ant colony optimization, we create a finite number of ants to explore a space. Each ant will build a solution or at least a component, starting from an initial state to another. Each ant will then will have some information about the problem at hand. Based off each ant’s own performance, the ant will modify how the other ants view the problem. They act independently and simultaneously, resulting in a type of cooperative behaviour that emerges. A solution will have an associated cost that is will be minimized throughout the processes.

In the processes of building a solution an ant, the ant will randomly pick an initial state. From there it will pick from a list of potential states (related to the private history of the ant) then randomly pick one weighted with pheromone levels (related to the publicly known pheromone levels). This stops the ant from making unfeasible solutions just my using only its memory.

The decision on when to release pheromones into the environment and how much depend on the implementation of the method and the quality of the solution. Ants can deposit pheromones while creating a solution, while after building a complete one, or both. The more ants pick a particular path, the interesting it will look for the other ants. The amount of pheromone that is deposited is inversely proportion to the total cost of the solution created. So the shorter the solution the more pheromone is deposited to all the components of the solution.

There is also an evaporation schedule, dependent on implementation. If the evaporation takes place after the creation of all the solutions, then the emphasis is that old solutions will be slowly forgotten. If evaporation takes place during the creation of a solution, the emphasis is that similar solutions won’t be created in a single iteration. After an iteration the current set of ants are terminated.

[2] Fogel, D. B. (1995). *Evolutionary computation*. Piscataway, NJ: IEEE Press.
[3] Holland, J. (1975). *Adaptation in natural and artifcial systems*. Ann Arbor: University of Michigan Press.

Then after of the previous events occur there may be final modifiers depending of the implementation, called a daemon. These actions will further modify the environment, which will be based off the solutions created during the current iteration.

In summery, we have covered the three predominate steps in the method (ant generation and activities, pheromone evaporation, daemon actions) and depending on the implementation of the ant colony method, the events are synchronized or scheduled at different times.

The following is a general form of how the code is implemented.

```
function ACO
    while (termination_criterion_not_satidfied)
        scheduel_activities
            ants_generation_and_activity();
            pheromone_evaporation();
            % optional
            daemon_actions();
        end % scheduel actitvities
    end %while
end % function ACO
%********************************
function ants_generation_and_activity();
    while (available_resources)
        % Creates new ant
        scheduel_the_creation_of_a_new_ant();
        new_active_ant() ;
    end % while
end % function ants_generation_and_activity
%********************************
function new_active_ant()   % ant life cycle
    initialize_ant();
    M = update_ant_memory();
    while  ( current_state ~= target_state)
    %The target state refers to a complete solution,
    %or to a component of a complete solution, built by the ant.
        A = read_local_ant_routing_table();
        P = compute_transition_probabilities(A,M,problem_constraints) ;
        next_state = apply_ant_desicion_policy(P,problem_constraints);
        move_to_next_state(next_state) ;
        if (online_step-by-step_pheromone_update)
            deposit_pheromone_on_the_visited_arc();
            update_ant-routing_table()
        end % if
        M = update_internal_state();
    end % while
    if (online_delay_pheromone_update)
        evaluate_solution();
        deposit_pheromone_on_all_visited_arcs();
        update_ant_routing_table();
    end % if
    die();
end % function new_active_ant()
```

Under scheduel_activities, the events that are indented are executed depending on the implementation. The segment online_step-by-step_pheromone_update, allows for the pheromone trails to be affected during the creation of solutions, while online_delay_pheromone_update, affects the pheromone trails after the creation of the solution.

Different Implementations

Ant System Algorithm

This is the first recognized implementation of this algorithm. We will recap on the what takes place. So we start with m many ants and then they make there solutions. Since this is the first generation of ant, the pheromone trails are all initialized to a small positive constant τ_0 . Since the pheromone is the same everywhere there is no particular preference, so the solutions are created randomly, made of sections of paths we will denote, $path(i,j)$. Where i is the initial node we starts at, and j is the node we end at for that particular segment. After the creation of the solutions, the total cost is calculated L, defined by:

$$L_k = \sum_{(i,j)\in T_k} cost(i,j)$$

Where T_k is the ordered set of pairs of nodes (i,j) that make up the solution for ant k. Then we calculate the pheromone we will add to the path segments.

$$\Delta\tau_{ij}^k = \begin{cases} {}^{\mathrm{Q}}\!/_{\mathrm{L}_k} & \text{if } (i,j) \in T_k \\ 0 & \text{if } (i,j) \in T_k \end{cases}$$

Where $\Delta\tau_{ij}^k$ will be the amount of pheromone added by ant k to $path(i,j)$. Note that Q is a parameter that can be varied by the user. Notice that the smaller the total cost of a solution, the more pheromone is added. So then if we take into account all the ants:

$$\tau'_{ij} = (1-\rho)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k$$

Note that we sum over ants, and where τ_{ij} is the previous iteration (initially it is zero) pheromone level on $path(i,j)$, τ'_{ij} will be the new pheromone level, and ρ is the evaporation constant. Now we start a new iteration. We create m new ants and randomly pick starting positions. Then the ant will now make decision list, denoted $A_i = \{a_{ij} | \, j \in N_i\}$. We denote N_i the list of nodes that can be traversed from node i, and we define a_{ij} as:

$$a_{ij} = \frac{\tau_{ij}^{\alpha} \cdot [cost(i,j)]^{-\beta}}{\sum_{\gamma\in N_i}\left(\tau_{i\gamma}^{\alpha} \cdot [cost(i,\gamma)]^{-\beta}\right)}$$

Were α and β are positive parameters that can be varied by the user, α affects now much the environments affects the decision, and β weighs the decision to nodes of least cost that are accessible. If $\alpha = 0$ then nodes with the least cost would usually picked. If $\beta = 0$ then only path that we a part of good solutions would be picked. So it is evident that we need a certain balance between the two parameters is a must. We define p_{ij}^{k} the probability of ant k picking a particular path (i, j) to be:

$$p_{ij}^{k} = \frac{a_{ij}}{\sum_{\gamma \in N_i^k} a_{i\gamma}}$$

Where $N_i^k \subseteq N_i$ is defined to be the set of nodes that ant k can pick. This is because depending on the problem, the ant may always be able to any node to go to at any time. There may be constraints introduced into the problem that force this to happen. The creation of N_i^k is also dependent on the past history of ant k. We iterate until all the ants create suitable solutions, then we repeat the update pheromone process.

Max-Min Ant System Algorithm

This is a very similar implementation as Ant System, with a few changes. The daemon determines the best solution created in the iteration and only the paths that composed that solutions will have pheromones added to them. Another important difference is that the pheromone values are restricted to the interval of $[\tau_{min}, \tau_{max}]$, as well as the pheromone trails are initialized to τ_{max} . A useful this that can be added into this implementation is to have:

$$\Delta\tau_{ij} \propto (\tau_{max} - \tau_{ij})$$

This forces the relative difference to get smaller, and reinforces the exploration of the solution space.[4] This particular feature is not in the submitted code.

Ant System Rank Algorithm

Again very similar to the Ant System implementation. Except after the creation of solutions we pick the top σ many ants and then they are ranked from best solution to worst solution. Then the pheromone trails are updated in the following manor:

[4] Stutzle, T., & Hoos, H. (1997). Improvements on the Ant System: Introducing MAX ¡MIN ant system. In *Proceedings of the International Conference on Artifcial Neural Networks and Genetic Algorithms* (pp. 245–249). Springer-Verlag.

$$\tau'_{ij} = (1-\rho)\tau_{ij} + \sigma \cdot \Delta\tau_\mu\Big|_{\mu=1} + \sum_{\mu=1}^{\sigma-1} \Delta\tau_{ij}^\mu$$

$$\Delta\tau_{ij}^\mu = \begin{cases} \dfrac{\mathrm{Q_{rank}} \cdot (\sigma - \mu)}{\mathrm{L_\mu}} & \text{if } (i,j) \in T_\mu \\ 0 & \text{if } (i,j) \in T_\mu \end{cases}$$

Where $\tau_{ij}, \tau'_{ij}, \rho$ have there usual meanings and T_μ denotes the ordered set of paths used to form the solution made by the ant with rank μ. And L_μ denotes the cost of the solution made by the ant with rank μ. So the pheromone added is weighed by the rank of the ant, and the best ant gets an extra bit on top.

Ant Colony System Algorithm

Once again it is based of the original Ant System, with key changes. A key difference is the pseudo-random-proportion rule. Let $q \in [0,1]$ be a randomly distributed variable, and let $q_0 \in [0,1]$ be a parameter defined by the user. Then the probability of ant k at node i picking the next node j defined as:

$$\text{if } q < q_0 \quad p_{ij}^k = \begin{cases} 1 & j \in \{\gamma |\, a_{i\gamma} = \max(\{a_{il} | l \in N_i^k\})\} \\ 0 & \qquad\qquad otherwise \end{cases}$$

$$\text{if } q \geq q_0 \qquad\qquad p_{ij}^k = \frac{a_{ij}}{\sum_{\gamma \in N_i^k} a_{i\gamma}}$$

The next difference has two variants. After the creation of all the solutions one variant just picks the best solution and only have that solution update the pheromone trails. While the other variant picks the best solution and then locally optimizes it. Then with the better solution it updates the pheromone trails.

The biggest difference is the dynamic pheromone changes during the creation of solutions. Then an ant chooses to go to node j from node i, the pheromones for that path are then altered by the following equation.

$$\tau'_{ij} = (1-\psi)\tau_{ij} + \psi \cdot \tau_0$$

Where $\psi \in [0,1]$ and τ_0 is the same initialized pheromone level as in the basic Ant System. This in general tries to decrease the pheromone levels as an ant uses the path. This discourages ants from making similar solutions. This is why it is called Ant Colony System, because in all other previous implementations, all the ants independently made a solution. Now the ants make solutions based off what solutions the other ants are making.

The last feature that should be implemented (that isn't in the submitted code) is idea of a candidate list, in where it is a list of nodes that are known to produce good solution when coming from the current node. The ant will look through the list in a sequential manor until it has found a node that can be moved to or the list is exhausted, in which case the regular probabilistic form will be used.

Ant Colony Rank System Hybrid Algorithm with Genetic Algorithm Operators

This is not a standard implementation of the ant colony system, but with my short time working on this, I found this to work quite well. This version of code is submitted. As in the Ant Colony System during the creation of solutions, there is a dynamic change in pheromone levels to deter ants from creating similar solutions. Like the Ant System Rank the best σ_1 many ants are chosen, but now with Genetic Algorithm Operators, I create copies of those solutions and apply the operators. Then all the solutions are bunched together and the best σ_2 many ants will have their pheromone added to the paths that they used in there solutions. The Genetic Algorithm operators act like a local optimizer as in the Ant Colony System for the best solutions. The user can define how long the local optimizer can search for, during each iteration.

Application & Comparison

Traveling Salesman Problem

Table 1

$m = n$
$\alpha = 1$
$\beta = 5$
$\rho = 0.5$

In the traveling salesman problem you nodes that represent cities and you must make a round trip by going to every city only once. It is experimentally found that the constants m, α, β, ρ should be given the values in the table 1. Where n is the number of cities in the problem. With my own version of my code I tested and compared the different implementations.

I used a problem with 22 nodes, and allowed only 500 iterations.

Implementation	Best Solution Found
AS	77.0571
MMAS	76.2111
AS_{Rank}	76.5455
ACS	76.1122
Hybrid	75.3097
Hybrid+As	75.5356

So with in 500 iterations the Hybrid implementation found the best solution. In fact it is the same solution that coincides with the best known solution found by another group for that particular arrangement of nodes. The worst method is the original ant system. This somewhat verifies other claims that the Ant System on it's own isn't as effective as it can be. Hybrid+Reg is the hybrid implementation also combined with the old pheromone trail update. So every ant has their solution given pheromone but only the top σ many will get extra. So here are some results from the ones motioned above.

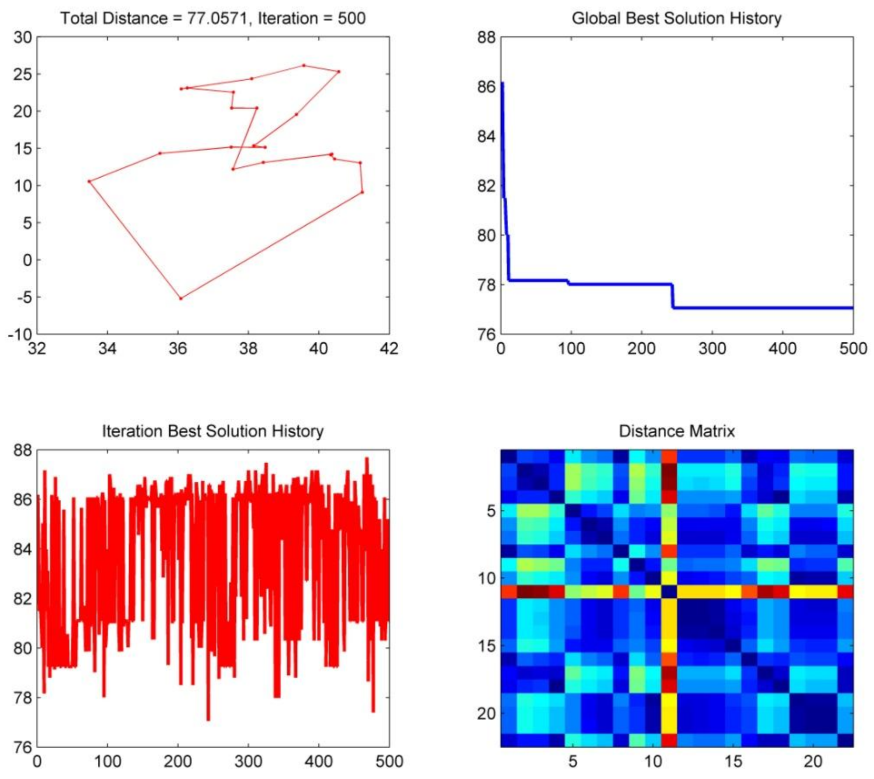

Figure 2 – Ant System (AS)

You can see that after about 250 iterations, the Ant System was stuck in a local minima. As well as the Iteration best was quite erratic during the entire run. This will be a common feature with other related ant systems. For the Max Min Ant System, the interval is $[0.1,5]$. For the AS$_{\text{Rank}}$ and hybrid method, $\sigma = 5$. For the Ant Colony and Hybrid System, $\psi = 0.25$.

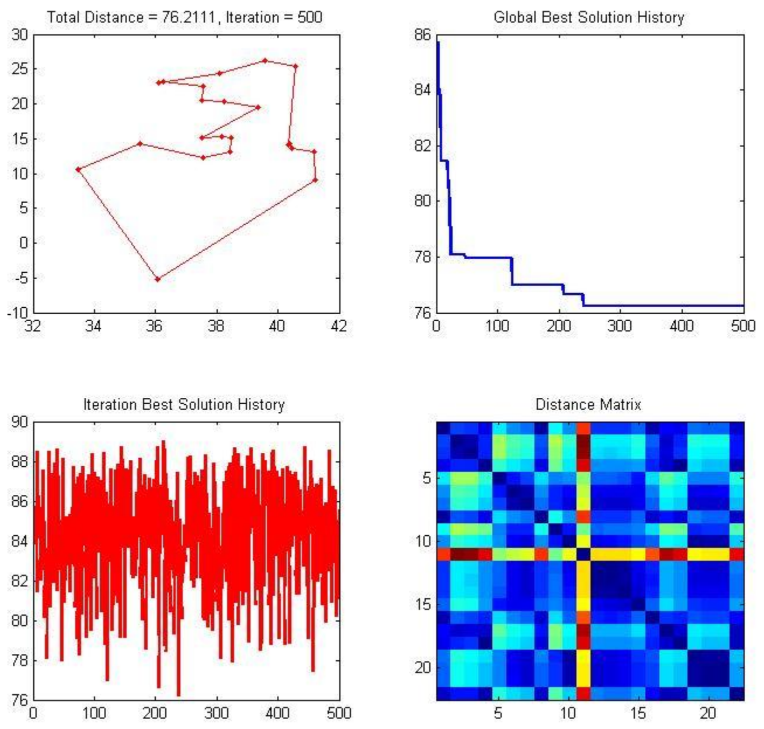

Figure 3 - Max Min Ant System (MMAS)

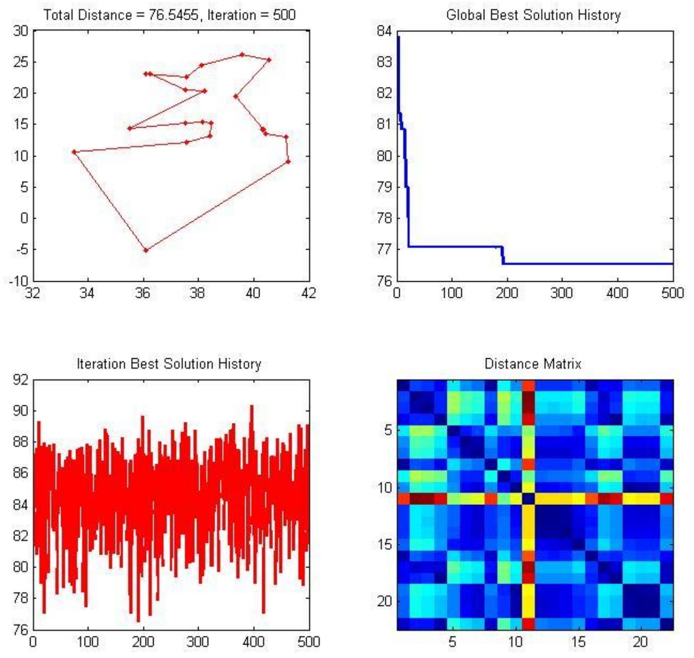

Figure 4 - Ant System Rank (AS_{Rank})

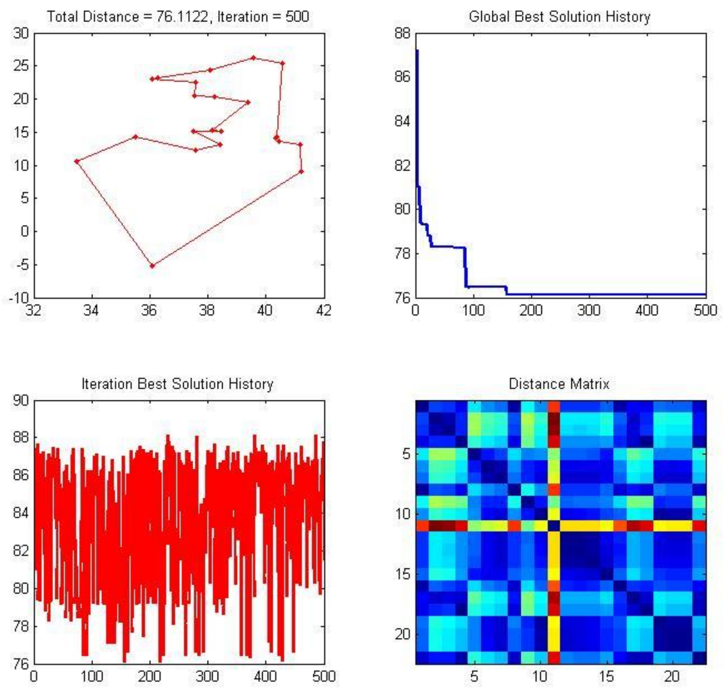

Figure 5 - Ant Colony System (ACS)

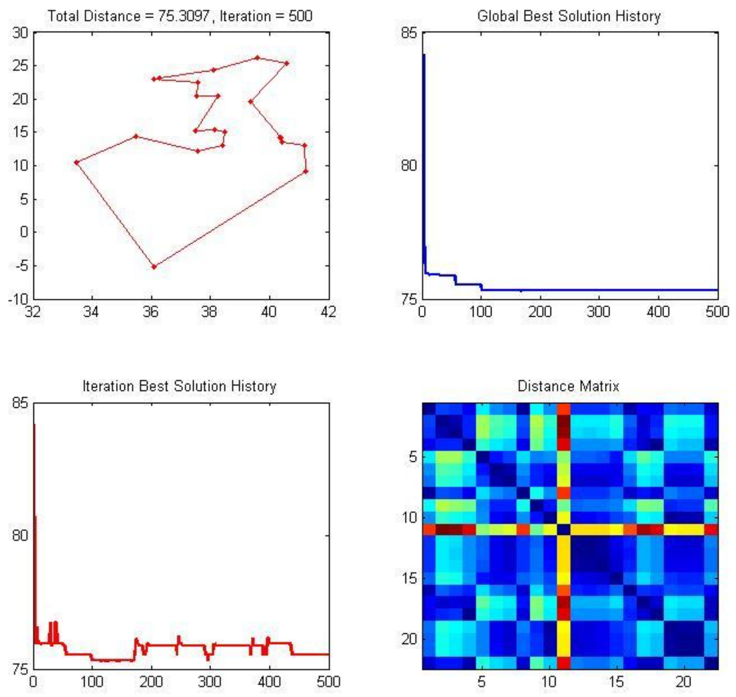

Figure 6 – Hybrid

Notice how the Iteration Best Solution is consistently low. This is probably a direct result from the local optimizer that tries to optimize the best solutions. For reference the image of the best know solution is.

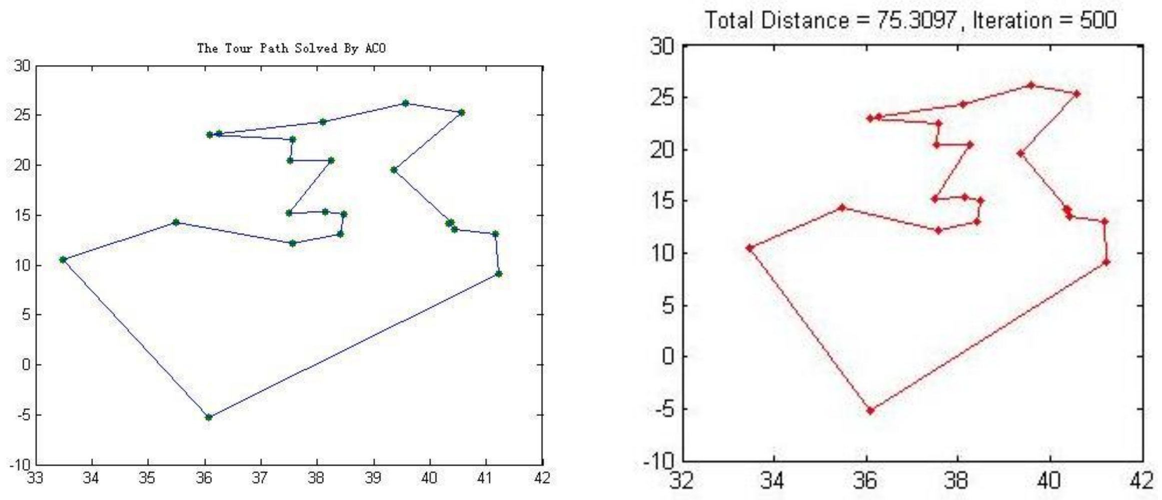

Figure 7 - Solution Comparison

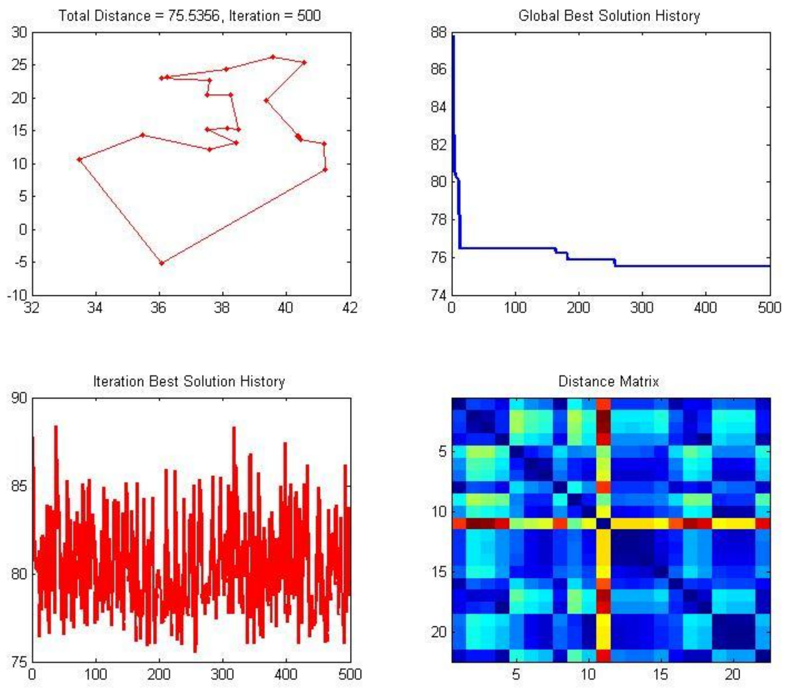

Figure 8 - Hybrid + Reg

Notice how the iteration best solution history is again erratic. This now also be a result from allowing all solutions (particularly the bad ones) to affect the pheromone trails. A common pit fall will all the implementations is that they get stuck. But the Hybrid method would seem the least likely to stay stuck as it continuously produces solutions near the current best.

I then repeated the test for all the implementations for a higher dimensional problem. The solution found by another group is the following image.

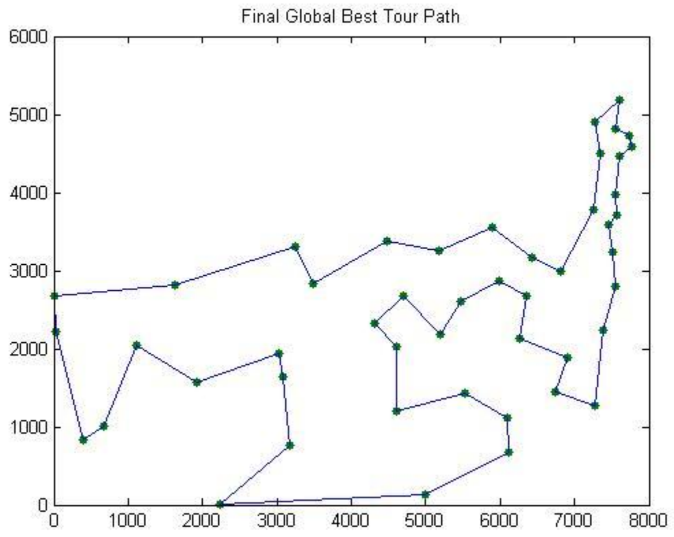

Figure 9 - Best Path for TPS 48 Nodes

With a set values set the same as the last comparison except for the iterations to be at 2000, none of them found this solution. The results are summarized in table 2.

Implementation	**Best Solution Found**
AS	35409.7068
MMAS	34344.0960
AS_{Rank}	34350.7159
ACS	35686.6299
Hybrid	34411.8333
Hybrid+As	34370.9336
Hybrid α	34291.2899
Hybrid β	34167.3023

Table 2

Hybrid α is the Hybrid method with the Local optimization is allowed to search for a slightly longer period of time for better solutions. Hybrid β further extends the ideas with α but I also let it run for 10000 iteration. As you will see, that was unnessary.

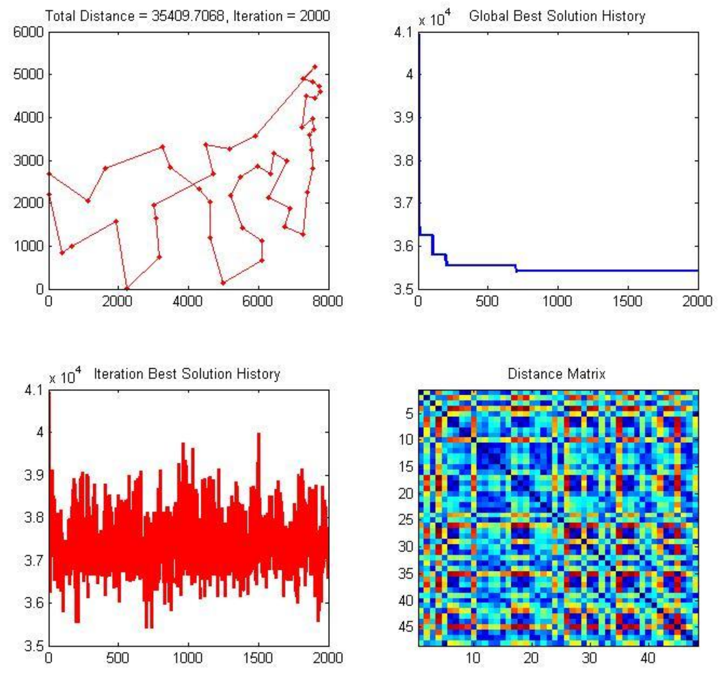

Figure 10 - AS - 48 Node

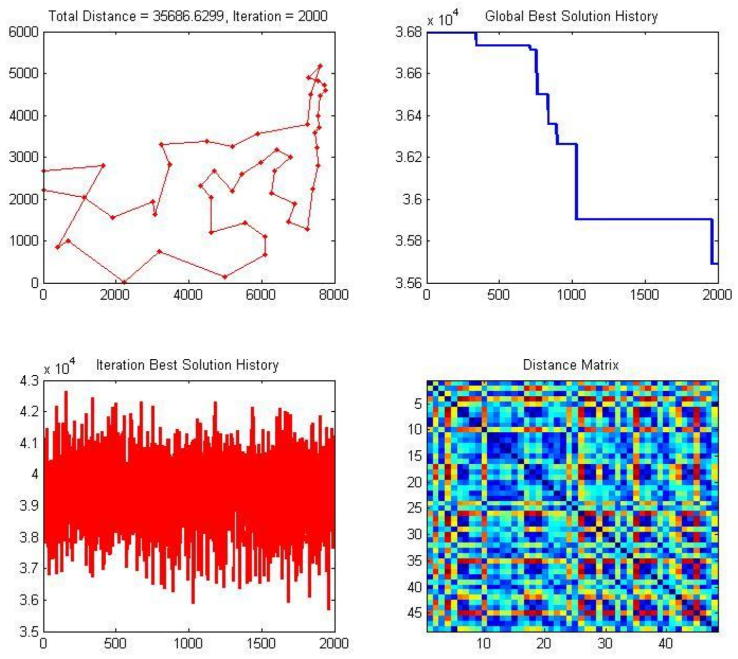

Figure 11 - MMAS - 48 Node

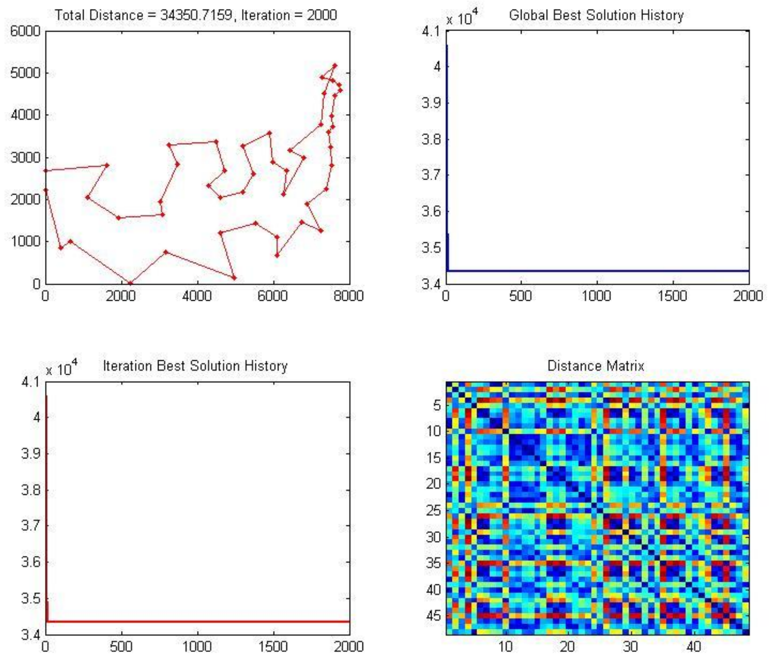

Figure 12 - AS$_{Rank}$ - 48 Node

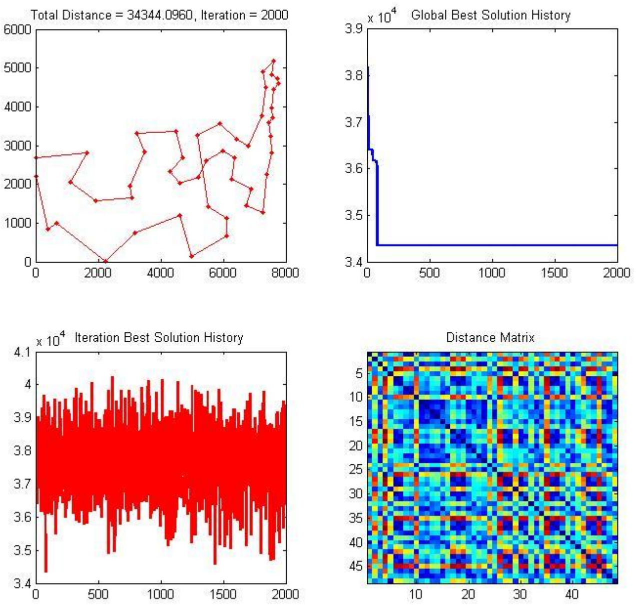

Figure 13 - ACS - 48 Node

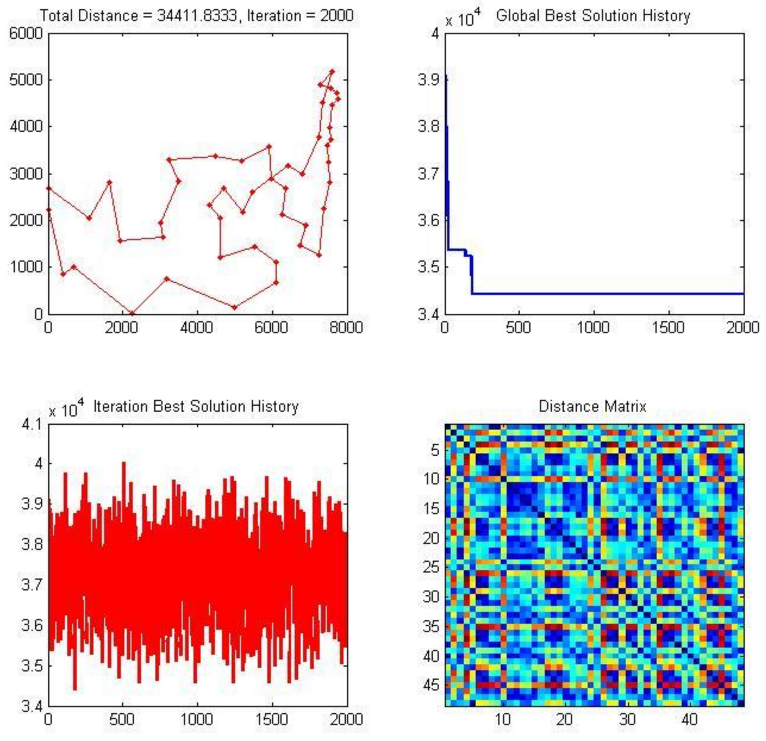

Figure 14 - Hybrid - 48 Node

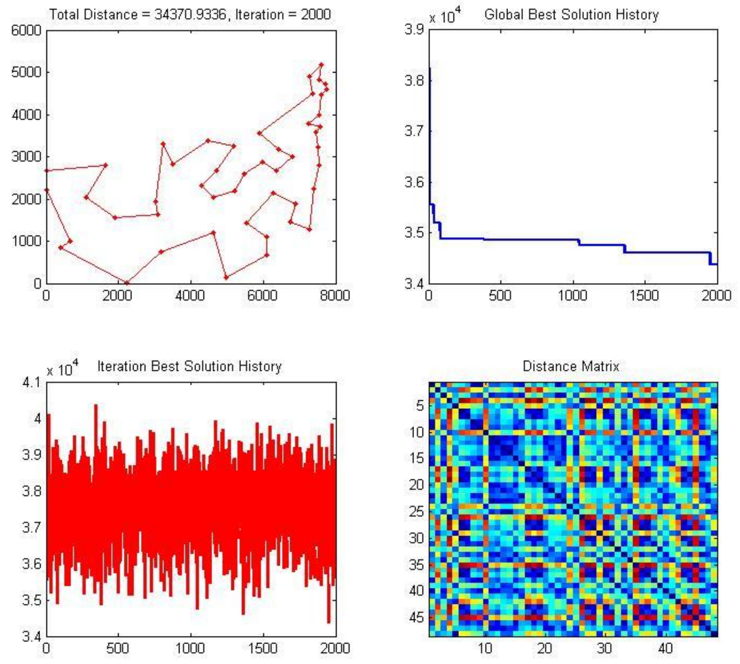

Figure 15 - Hybrid + Reg - 48 Node

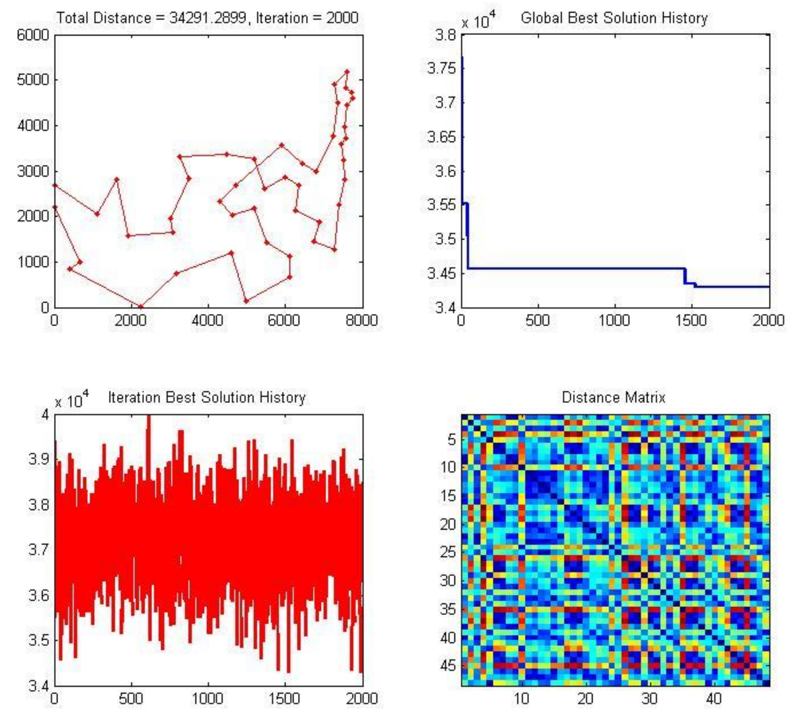

Figure 16 - Hybrid with Optimizer function lengthened

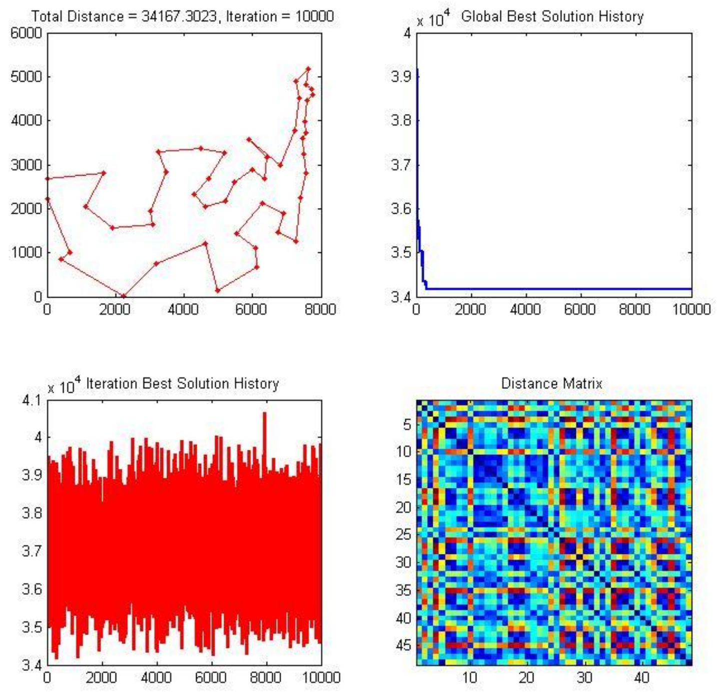

Figure 17 - Hybrid with Longer Optimizer and 10000 Iterations

This was to check if there would have been any significant new results, which clearly there isn't. The later implementations depend on a parameter ψ that may or may not be at the best value.

Quadratic Assignment Problem

In the quadratic assignment problem we have n many facilities and n many locations. We denote the i^{th} construction step to be where the i^{th} facility will be placed. I have not investigated as in-depth. In the submitted code only AS, MMAS, AS$_{\text{Rank}}$, currently are set up to attempt this problem. Since I only have a reference problem of 8 dimensions, all three implementations usually find the best known solution after some time.

Conclusion

The Ant System Optimization and all its variants pose as ideal tools to solve combinatorial problems. I have found that for lower dimension problems Ant System is the least powerful and Ant Colony System and Hybrid work very well. Note that Hybrid uses key Ant Colony features, which make Ant Colony so useful. The higher the dimension gets, the smaller the gaps in effectiveness become. The is due to the fact that with higher dimension problems you want search the solutions space more. Although with a Ant Colony and Hybrid, they always found close to optimal solutions in a very short amount of time. So in the situation where you want dynamic solutions to a dynamic problem (i.e. communication and network problems), they would be prime candidates. The only thing that is a problem, as will all optimizers, is getting stuck in local minima. Well if you look at the iteration best history in previous figures, you will notice that it's not stuck pre-se, it's more that it hasn't found anything better. In the case where it's not creating the best solution every run (which can be determined by the iteration best history graph) you are still exploring the solution space. The problem with Ant System and all variants, is that it may look at old solutions again. Which lengthen the process of finding the best solution. Though this might be fixed if a candidate list is implemented.

References

1. Ant algorithms for discrete optimization Artificial life [1064-5462] yr: 1999 vol: 5 iss: 2 pg: 137 - 172
2. Ant Colony Optimization
   Genetic programming and evolvable machines [1389-2576] yr: 2005 vol: 6 iss: 4 pg: 459 - 460