Implementing Graph Coloring with hybrid ACO and PSO

By Russell O'Brien, Isaak Kabuika, and Elliot Ketchel

Keywords: Graph Edge Coloring, Graph Coloring, Particle Swarm Optimization, Ant Colony Optimization, Optimization, Hybrid

Abstract: Graph edge coloring is a np-hard Problem that is applicable in many real life schedule problems. We attempted to implement a hybrid nature inspired algorithm to solve this problem. We implanted a version of Ant Colony Optimization to solve a graph edge coloring problem, and then implemented a Particle Swarm Optimization to find the optimal parameters for Ant Colony algorithm. The implementation of the Ant Colony algorithm was not very effective at solving the problem. However, the hybrid algorithm had some promising data, especially the convergence of parameters in the Particle Swarm algorithm, which lead us to believe the hybrid algorithm could work well with some further work.

1. Introduction

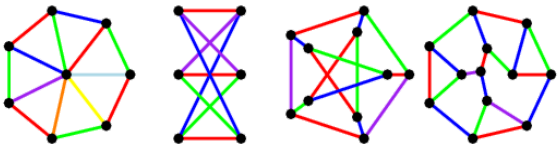
In this project two Nature inspired Algorithms are used to create a hybrid algorithm to attempt to solve a graph edge coloring problem. Graph edge coloring is a np-hard problem, where the very best algorithms tend to take at least exponential time to solve. In Addition there are multiple ways to go about solving a graph edge coloring problem. Because of the massive amount of time needed to solve any sizeable graph edge coloring problem, there are not many efficient methods of solving such a problem, and with numerous real world applications to this problem a good time effective solution could be incredibly useful. To that end we attempted to implement a version of Ant Colony Optimization, or ACO for short, to solve this graph edge coloring problem. In addition, since it is very difficult to find the optimal parameters for an algorithm such as ACO, we implemented a version of Particle Swarm Optimization, or PSO for short, and used this to search for the optimal parameters for our ACO. by hybridizing these two algorithms we hoped to find a time efficient and passable solution for the graph edge coloring problem

2. Graph Coloring

You may be asking yourself, what on earth is graph edge coloring, how the shit does graph edge coloring work, why do I even care about graph coloring. Well settle down, sit back, and keep on reading for all the naughty details. Graph Edge Coloring is a subset of Graph Coloring problems, where instead of coloring the nodes of a graph, you attempt to color each edge. All graph coloring problems require that no two adjacent edges have the same color. In other words a node can have only one edge of

each color connected to it. See Figure 1. for examples. The applications for graph edge coloring include mostly scheduling. The most common are Scheduling round robin style tournaments, dividing time slots for communication between hosts in a network, or scheduling a manufacturing processes. One of the fascinating things about this problem is how different ways of attempting to solve the problem can be used to solve different problems. For example, given a certain number of available time slots you could try and fit everything together as well as possible. or you could try and find the minimum number of time slots needed to fit everything together. The type of graph used also allows for variation in the problem. While there are lots of possible variations to this problem, in our project we have focused mainly on solving the graph with a set number of colors, or in other words a set number of time slots.

Fig. 1.



3. Ant Colony Optimization

3.1. Summary

Ant colony optimization(ACO) mimics the behavior of ants foraging for food. No ant in the colony has access to information about the best locations for food (as well as the shortest path to these locations). To efficiently search for and find food sources, ants explore their surroundings and communicate with each other via laying down and tracing pheromones.

3.2. Origins

When an ant finds a source of food after wandering around randomly, it carries the food back to its home, releasing pheromones along the way. Other ants will follow this trail of pheromones to get to the food source. As more ants use the path that led to the food, the pheromone concentration on the path increases, drawing more ants to that location.

Pheromones wear away over time so that ants can adapt to a changing environment or find more efficient routes to food. If a food source runs out, ants will stop reinforcing the path to its location. With the pheromone concentration not being reinvigorated by a steady stream of ants, the pheromone trail that used to lead to food will eventually disappear, and ants will stop taking the path to the exhausted food source. If an ant finds a shorter (i.e. more efficient) route to food, this route will surpass the original route in pheromone concentration, as it will be able to support a faster flow of ants. The faster flow will lead to a higher pheromone concentration, and ants will begin to abandon less efficient paths.

3.3. Algorithm

ACO operates using these same principles, Although our implementation for this particular problem has tweaked some of the common pieces of ACO. In this algorithm all of the ants work together to color a version of the graph. Each ant will start at a random node and traverse through

the graph going from node to node and coloring edges behind it until it reaches a node where it can no longer lay down a non-conflicting color. Ants do this by picking the next edge to travel, rather than the next node to travel to. Once all the ants have reached a point where they can go no further, the final product is the graph they have partially or mostly colored.

3.3.1. General

The standard method with which each ant selects its next edge is probabilistic, and weighted towards the most traveled edges in the pool of adjacent edges. For each edge an ant could choose to travel next, it calculates the pheromone concentration for that edge. this value is then raised to the power of a given constant. The ant then randomly selects the next edge, where the probability of an edge being chosen is its value from the above product divided by the sum of the products from all potential edges.

3.3.2. Implementation

We implemented a modified version of Elitist Ant System (EAS). For EAS, each ant selects its next edge in its tour based on the probabilistic heuristic discussed in section 3.3.1 (*General*). Once all the ants have finished with a tour through as many edges as possible, the pheromones on them are updated as follows. First, the pheromone concentration on each edge is worn away based on a constant evaporation factor. Next, for every edge, the pheromone concentration is increased by the product of the number of edges successfully colored and a scaling factor. Each possible color has its own pheromone concentration, and the concentration is increased only for the color that was placed on the edge, although it is still evaporated for all colors.

4. Particle Swarm Optimization

In nature, birds flock together to most efficiently find food. Each individual bird in a flock moves based on three basic principles. First, each bird wants to fly toward the area where it has experienced the highest concentration of food. Second, birds will adjust their trajectories to follow their neighbors towards the high concentrations of food that they have discovered. Finally, birds will tend to continue flying in the same direction that they have been flying. When combined, these three decision making factors operating on the individual level cause the whole swarm to gravitate towards the location with maximum food density.

PSO models this flock functionality with a flock of particles in a solution space. In our implementation, we initialize a swarm of randomly placed particles in this space, each with a random initial velocity. Then we assign each particle to a neighborhood of particles based on a specified neighborhood topology. Then we continually adjust the velocities of the particles based on the position of the best solution the particle itself has encountered and the best solution any particle in its neighborhood has found, and move the particles in the solution space based on those velocities. We repeat this process until we reach a maximum number of iterations or until we find the absolute minimum of the test function.

To evaluate a PSO particle, we will run an ACO graph coloring problem, where the input parameters are determined by the particle's position in the PSO solution space. Because we want to optimize 3 different ACO parameters(alpha, rho and elitism factor), our solution space will be three dimensional. Therefore, a particle's position will be defined an array with 3 double values describing the current settings of those three parameters. Similarly, every particle will need a velocity array, containing its current velocities in each of the 3 dimensions. Each particle can then be evaluated initializing a new ACO object using the parameter values taken from the current position in the solution space, then returning the number of graph edges the object was able to color.

2.1 Topologies

Once we have our initial particles randomly placed into our solution space, we assign each of the particles a list of "neighbors" based on a variety of topologies. These "neighbors" are imitating the birds surrounding a bird in a flock that the individual bird can communicate with. In this case a particles neighbor is a particle that the first particle can communicate with in order to learn about other best solutions. There are numerous types of neighborhoods that are commonly used. In this project the type of neighborhood used was a Von Neumann topology. For this neighborhood assignment, all the particles are imagined to be in a 2 dimensional array. Their positions in this array have nothing to do with their actual positions in the solution space, just the order in which they are initialized. This array stays constant for the duration of each experiment. In this array, a particles neighbors are the particles to its left, right, up, and down in the array(for particles on the edges of the array, the array is imagined to wrap around itself). Hence, each particle gets 4 neighbors.

2.2 PSO body

Once the particles are initialized and their neighborhoods are determined, the algorithm can begin. The PSO algorithm involves essentially 4 parts. The first step of the Algorithm is to evaluate each particles solution. As previously mentioned, the solution of a particle is the number of graph edges an ACO algorithm initialized with parameters as dictated by the particle's position in the solution space. After the particle is evaluated it will have a value that can be compared to all other particles. This value is crucial to the next part of the Algorithm.

After each particle has been evaluated the PSO algorithm will update each particles personal best, each neighborhoods best, and the global best. The particle has found a new personal best if the current value is higher than any other value the particle has seen. This value does not take into account any values its neighbors have found, just places it has actually been. The neighborhood best is the best particle that any of the particles in a neighborhood have seen. In other words the neighborhood best for a neighborhood is the best personal best out of all particles in that neighborhood. Finally the global best is the best value found by any particle, or the best personal best out of all particles.

Once the personal, neighborhood, and global bests have all been updated, the particles velocities can be updated. Each particle's velocity is updated based off of its own personal best and the neighborhood best. This means the global best is only used regularly in the global topology, but it is still important to maintain the global best with all other topologies in order to keep track of the best answer. A particle's velocity can be updated based off of the personal best or neighborhood best with different weights. A big problem with many PSO algorithms is when particles get velocities that are too high. This causes particles to leave the reasonable bounds of whatever problem it is exploring, and makes the algorithm nearly useless. In order to prevent this there is a restriction factor that is taken into consideration when updating particle velocities. For this particular application, we needed to impose absolute boundaries on where the particles could explore so that we didn't get parameter inputs that crashed our ACO object(more on this in Hybridization section). The full equation for calculating the velocity of a particle is as follows.

$$V_{id} = \chi(V_{id} + c_1 \in_1 (p_{id} - x_{id}) + c_2 \in_2 (p_{gd} - x_{id}))$$

In this function *i* represents the individual in question, *d* represents the dimension, χ is the constriction factor, c_1 is the weight on the personal best p_{id}, and c_2 is the weight on the global or neighborhood best p_{gd}, and finally $\in_1$ and $\in_2$ are both independent random numbers generated uniquely for each update. A common value for the constriction factor is 0.72984, and common values for c_1 and c_2 are 2.05. The purpose of $\in_1$ and $\in_2$ is to add some randomness to the velocities.

Once the velocities have been updated, the particle's positions in the search space are updated based on their new velocities. To calculate a particles new position the particles velocity for each dimension is simply added to the particles current position in each dimension. Then, once the particles have all had their positions updated the whole process starts over again.

This is repeated for a set number of iterations, and once the max number of iterations is reached the global best as of the last iteration is the final answer.

5. Hybridization

We have already discussed how ACO and PSO will be combined in this report in the beginning of the previous section. This section will be devoted to specific changes we had to make in our algorithms to make the two cooperate with each other.

Firstly, we had to impose strict boundaries on the areas that the particles in the PSO solution space could travel. This is because outside of certain parameter values, our ACO algorithms would fall apart. For example, having a rho value of more than one would result in a pheromone deterioration that would produce negative pheromone concentrations on legs(because pheromones are worn away by factor 1-rho). Negative pheromone concentrations are meaningless, and would throw off many

other calculations, so we prevented particles from going beyond 1 in the rho dimension.

Secondly, we needed to change the logic of our PSO algorithm to search for the max solution in the solution space. Normally PSO is used to minimize a function, but we were trying to find the combination of params that maximized the performance of our ACO algorithm. This change was simple enough to implement in the basic logic of PSO.

Finally, we had to take into account the number of iterations to run each different algorithm for. We knew that because each PSO particle evaluation would require multiple tours of an ACO algorithm, and that many different points in the PSO solution space needed to be evaluated, our algorithm would likely take a long time to run. Therefore, we were faced with the task of determining the minimum number of ACO iterations were necessary to give an accurate answer for edges colored to PSO, so that PSO would be able to improve our parameters but not take a prohibitively long time.

6. Experimental Methodology

Our goals for testing this Algorithm were to determine if our Implementation of ACO could be a possible way to find a solution for the Graph Edge Coloring problem and to determine if our implementation of PSO would actually have an effect on the effectiveness of our ACO. To that end we looked mostly at the trends of the results given over a number of ACO iterations and the trends of the results given and the parameters generated over a number of PSO iterations. We decided that if we can show a solid trend of increasing results over a number of iterations of ACO that we can predict the ACO will eventually find a reasonable result for the Graph Coloring Problem. We also thought that a trend of both improving solutions from our ACO and a convergence of the parameters Generated by our PSO over iterations of PSO would be sufficient to show that PSO does in fact have an effect on the effectiveness of ACO. In order to get a good idea of how these trends appeared, or didn't appear, in our project we simply ran the project a number of times and kept track of all the aforementioned data.

For testing we decided to run the outer PSO algorithm for far fewer iterations than it is conventionally run for. In our project we settles on about 30 iterations. This was decided for two reasons. First the amount of time needed to evaluate each particle was likely to be very large, as the evaluation was running the ACO algorithm. Second, the small number of dimensions, only 3 parameters in this case, combined with the fact that we weren't sure about the convergence of these dimensions, made it seem fairly unnecessary to run for a large number of iterations. We also decided on a fairly small number of particles, and settled on 10, for the same reasons as number of iterations. our Parameters for the PSO were also kept to a constant level. We decided to use the parameters that we determined to be the best in our previous PSO project.

For the ACO algorithm we planned to run each Algorithm for a similar number

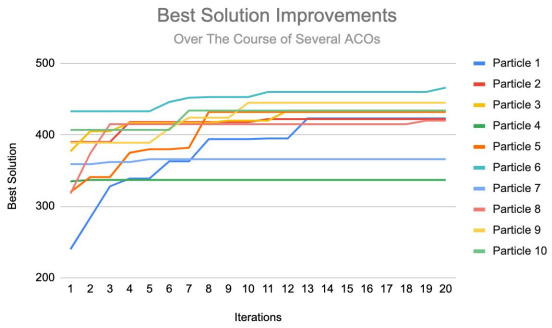
of iterations to our previous ACO project, and settled on 20 iterations for each particle. This was again to keep the overall runtime of the project to a reasonable level. In addition, since a graph was used in solving a problem and Ants worked collaboratively instead of independently, a number of graphs were solved during each iteration of ACO, meaning that there were a very large number of possible solutions generated even though the individual number of Iterations for each Algorithm was not incredibly large. the number of Ants was less important, as they worked collaboratively on the solutions, and so was kept at 30 ants. The number of graphs that were solved during each iteration of ACO, called graph dimensions in our project, was set to 15.

The Parameters for ACO were kept constant, except the three that we thought were not obvious what the best value was. By obvious we mean that it would be obvious that more ants, more colors, or more iterations would allow for better solutions every time assuming the algorithm was working correctly. The parameters that we focused on were the Alpha, Rho, and E Factor, which are all important to the selection of paths and updating of Pheromones but do not have an obvious effect as the value is changed like the other Parameters.

7. Results and Discussion

The first thing we were looking at when it came to results was the trajectory of our ACO algorithm. Figure 2 shows a full set of 10 different particles with different parameter values running for a full 20 iterations of ACO. Each different run improved over time, which was promising, but the trajectory was very irregular. It might improve a lot really quick and then not improve again for another 10 iterations, or it might not improve very much and then improve a lot suddenly in the last few iterations.

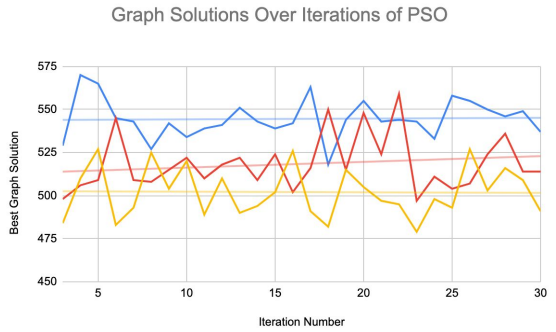
Fig. 2.



This inconsistency but general trajectory tells us that while our algorithm does generate solutions, It does not do a very good job of learning from itself consistently. The ACO algorithm does not do a very good job of improving consistently over time, which makes us believe that our ACO algorithm was not implemented very well for solving a graph edge coloring problem.

The Second point of interest was whether the ACO improved over the course of the PSO algorithm. Figure 3 contains the data of 3 separate runnings of the PSO algorithm and the best solution found by the ACO algorithms at each of the 30 iterations. The trajectory is slightly positive, but only slightly, and the solutions found tend to bounce around a lot without ever reaching a much higher point. Sometimes the best answer found is in the first couple ACO runs, and sometimes it isn't until the end.

Fig. 3.



This Inconsistency is more difficult to place exactly, with the knowledge that the ACO algorithm does not necessarily improve consistently. The failure to improve a large amount over time may be due to the implementation of the ACO algorithm having a limit to how well it can solve the graph. There also could be very little effect of the PSO algorithm on the effectiveness of the ACO algorithm, and the parameters being changed could have very little effect on how well the algorithm performs, However, this seems unlikely given the data about the conversion of the Parameters.

The third a final trajectory we were interested in was the conversion of the three parameters over the course of the PSO algorithm. Figures 4 through 6 show the difference between the highest and lowest values of the three parameters over the course of the PSO. Each parameter is on a separate graph due to the magnitude of the values being very different, but the overall trajectory of each is very similar. The E factor and Rho dimensions have much more variability over the course of the PSO, while the Alpha dimension drops very quickly and only has one spike. However all 3 have a strong downward trend in the difference, or in other words, the variability in values of all the particles converges towards similar values over the course of the PSO algorithm.

Fig. 4.

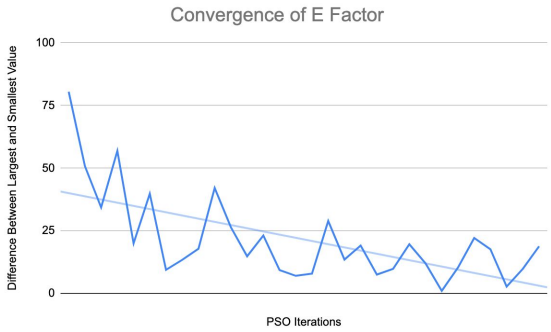


Fig. 5.

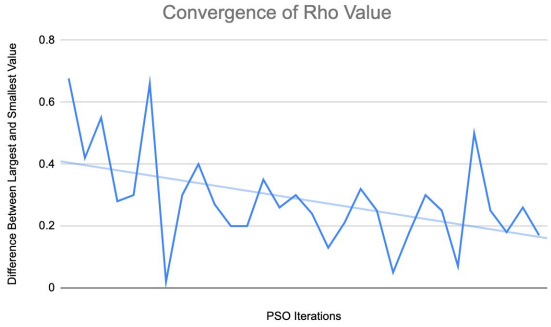
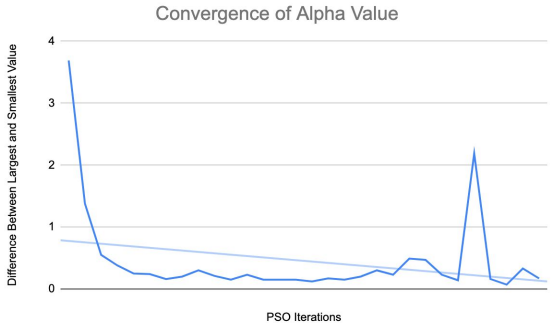


Fig. 6.



This convergence makes it difficult to say if the PSO algorithm is having any effect on the ACO algorithm. Each of the parameters is clearly converging towards a similar value over the course of the PSO, and this fact supports the Idea that there is a more effective set of parameters. If there was no difference in how the different parameters functioned then there would

be no reason for the particles to close in on a particular area of the search space. This also means that there has to be some sort of improvement in the ACO algorithm due to these changes that causes this convergence. Because of this it is difficult to determine if the lack of improvement over the course of the PSO algorithm is due to the PSO having no effect or due to the poor implementation of our ACO graph edge coloring algorithm.

8. Further Work

The ACO graph edge coloring algorithm is the place that needs work in the future to determine if the PSO algorithm could be effective. The ACO implementation we used did not perform at a level that allowed for very useful data about the effectiveness of our PSO. The Main issues that we think caused this lack of improvement in the ACO algorithm was the implementation of pheromones. Because pheromones were used to represent colors on a leg but also used to determine which leg to travel, the ACO algorithm reinforce the same paths over and over again. This is an issue because due to our implementation an ant may not travel every leg, and if certain legs never get travelled then they will never get pheromone, and no new solutions will be found. This is particularly important, because in the grand scheme of the graph edge coloring problem the actual color on the edge doesn't matter. any two colors could be swapped for every edge they appear on and the solution would stay exactly the same. Therefore we would want to implement our ACO with a different method of choosing edges and choosing what color to put on an edge. This would allow for us to explore more new solutions, and hopefully improve much more quickly and noticeably over iterations of ACO.

9. Conclusion

Our ACO implementation looks promising for solving graph edge coloring problems, but it still requires some further work before it is at a level where it can satisfy our goal of finding reasonably good solutions to these problems in a reasonable amount of time. The problems with our Implementation of ACO also made it difficult to determine how effective our PSO algorithm was at finding optimal parameters for our ACO algorithm. However, the convergence of the values in the search space in PSO looks promising for the algorithm to have some effect given a better implementation of the ACO.