CSE2530 Computational Intelligence

Ant Colony Optimization & Genetic Algorithm

Yoon Hwan Jeong, Oskar Lorek,
Ethan Keller, Andrzej Rubio Bizcaino

March 19, 2021

1 INTRODUCTION

In this report we explain how we implemented a *Swarm Intelligence* (*Ant Colony Optimization*) and a *Genetic Algorithm* to solve the problems presented in assignment 2 of *CSE2530 Computational Intelligence*. Our robot must now navigate through the supermarket (modeled as a maze) to find all the needed products while saving time and energy by finding the fastest/shortest route through the maze (supermarket). This a variant of the *Travelling Salesman Problem* and will be implemented with a *Genetic Algorithm*. In order for the robot to know how to quickly get from one location in the maze to another, we will implement a *Swarm Intelligence* based on *Ant Colony Optimization*.

2 ANT COLONY OPTIMIZATION

We will first implement a basic *Ant System Swarm Intelligence* based on *Ant Colony Optimization (ACO)* to teach the robot how it should determine a quick route from specified start and end points. Later on, this *Ant System* will be enhanced and 'upgraded' to an *Ant Colony System*.

2.1 List of features

Some features that could make it difficult to find the end point in the maze.

- loops
- open areas
- dead ends
- zigzag walls

2.2 Pheromones

The equation for the amount of pheromone $\Delta\tau^k$ dropped by an ant k on a path from location i to j in the maze is:

$$\Delta\tau_{i,j}^k = Q \cdot \frac{1}{L_k}$$

The constant Q is a parameter to regulate the general amount of pheromones dropped by the ants and L^k is the length of the path that ant k has travelled to get from i to j. This way, the amount of pheromones dropped by an ant is inversely proportional to the length of that ant's path. The purpose of the pheromones is to stimulate ants to follow shorter paths. This is reflected in the equation. Shorter paths will get more pheromones and will thus attract more ants and longer paths will get less pheromone and will thus attract less ants. The goal of the algorithm is to find the shortest[1] path from a given start to a given end point.

2.3 Evaporation

Evaporation will be modeled as a fraction of the pheromone in the maze that is removed after every generation. The full equation for the amount of pheromone that resides in the maze for a specific path from i to j is defined as the leftover pheromone (after evaporation) plus the total amount of pheromone that all the ants have left on this path. The amount of pheromone left is given by the following equation:

$$\tau_{i,j} = (1-\rho)\cdot\tau_{i,j} + \sum_{k=1}^{\#ants} \Delta\tau_{i,j}^k$$

The constant ρ is a value between 0 and 1 (inclusive bounds) that expresses the proportion of the amount of pheromone that will evaporate. This equation expresses how much pheromone stays on a path after a full generation. The old amount will partially evaporate (regulated by ρ) and the ants will have dropped their share of pheromone as well. The purpose of pheromone evaporation is to attract less ants to paths that are not the shortest. This way if ants find a shorter path, then the previous shortest path will gradually be 'forgotten' as the pheromones there will have evaporated. This makes sure that we do not get stuck in a local optimum.

2.4 Pseudo-code

```
Algorithm 1: Ant algorithm
  Input: generations, antsPerGen
  Result: Best route
  for i ... generations do
      for j ... antsPerGen do
          construct route;
      end
      evaporate pheromones;
      deposit pheromones on constructed routes;
      update best route;
  end
```

```
Algorithm 2: Route construction
  Input: start, end
  Result: Route from start to end
  set current position to start;
  while current position is not end do
      mark current position as visited;
      find available directions;
      if no directions found then
          return null
      end
      select direction;
      add direction to route;
      update current position;
  end
```

These are pseudo-codes for the 'standard' *Ant System* algorithm. We will now enhance it to hopefully get shorter routes and to improve the performance of the ants in tough situations like the ones featured in 2.1. The enhancements and improvements are explained in 2.5.

[1]It is important to remember that ACO is stochastic and will thus not always yield the shortest path, but a short path. We just want this path to be as short as possible.

2.5 Improvements

The pseudo-codes above depict the 'standard' ant colony optimization algorithm, *Ant System*, developed by Dorigo[4]. However, this algorithm has some limitations. For example an ant could get trapped by its own route when traveling in a loop, thus, not being able to find a route from start to end. In order to deal with such a problem, we have adapted the algorithm such that when the ant cannot proceed to any of the neighbouring cells, the ant steps backwards once. By doing so, the position that traps the ant remains marked as visited, stimulating the ant to choose another path. This is similar to Trémaux's algorithm, invented by *Charles Pierre Trémaux*[7].

This enhancement does not only eliminate the problems of ants getting trapped by their own routes. It also makes sure that dead ends and loops in the maze are eventually avoided. Normally, ants would potentially get stuck in dead ends and loops, but with this improvement they will travel back and choose another path on which they will drop pheromones. This will attract ants to the better paths while avoiding the loops and dead ends.

Zigzag walls can be interpreted as lots of very short dead ends. So these are now also avoided because ants will get out of them and not drop pheromone there.

In addition to this improvement, we experimentally noticed that ants would often step past the destination instead of just stepping into the destination cell. To counter this we added the following rule. If the current position of an ant is one step away from the destination, the ant chooses the direction of the destination instead of making a probabilistic decision based on pheromones.

Despite the already significant improvements, we can still optimize the algorithm. We decided to 'upgrade' our standard *Ant System* to an *Ant Colony System* which is an improved *Ant System* introduced by *Dorigo and Gambardella*[3]. This ACO algorithm introduces a new parameter q_0 which is the probability of exploitation. In our context, exploitation means choosing the next step direction towards the cell with the highest pheromone amount. This must be done in balance with the exploration of new routes which might possibly be better than the current best solution. To keep this balance, we introduce a local pheromone update. After an ant constructs a route, the amount of pheromone on it is updated following this equation:

$$\tau_{ij} = (1 - \varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0$$

where $\varphi \in (0, 1]$ is the pheromone decay coefficient and τ_0 is the initial value of the pheromone. All ants will thus modify the amount of pheromones on their routes. After all the ants have constructed a route we apply a global pheromone update. This update rule is similar to the one from the *Ant System* and is performed as follows:

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

where $\rho \in (0, 1]$ is the evaporation rate and $\Delta\tau_{ij}^{best} = Q/L_{best}$ if (i, j) is in the best route, else $\Delta\tau_{ij}^{best} = 0$. The ants' decisions are still shaped by pheromone information and heuristic information (shorter routes are better). But now we introduced a probability, the exploitation probability, that an ant will take a step in the direction of the cell with the highest pheromone amount. This boosted our exploitation while counterbalancing exploration with the new pheromone update rules.

To avoid travelling through sub-optimal paths in big, open spaces we make sure that the pheromones do not evaporate too fast such that we limit the amount of ants randomly 'exploring' the open space. Instead we rely on the pheromones from the ants that found a short path already. This way ants will more likely follow those paths.

Overall this resulted in faster convergence and shorter routes through the maze.

2.6 Parameter optimization

The optimal parameter values depend on the shape and features of the maze. To assess the optimally of a parameter value we performed the following process. For every parameter we select five different values that we would like to test. These five values are selected based on our intuition and the values used in practice. We chose to try out five parameter values per parameter due to computational expenses, but to get more accurate results we could use even more. For every grading maze, we run the algorithm for every parameter, for every parameter value. We isolate the one parameter that we are analyzing and keep all the other parameters constant (with default values). A complete grid search would have led to better parameter values due to the dependency between the parameters. Unfortunately this is computationally costly because of the amount of parameters we have so we decided to stick with isolated parameter tuning. This way we decide which parameter value is the most optimal per parameter by comparing for which parameter value the solution converges the fastest and results in the lowest route length. By 'converging the fastest' we mean getting to a point where the algorithm, for many consecutive generations, does not find a shorter route, in the fastest way possible. The graphs visualizing these isolated parameter tests can be seen in section 6. The optimal parameter values for every parameter for every maze are shown in Table 2.

We stated earlier that the optimal parameter values depend on the shape and features of the maze. Table 2 can help us understand the relation between the parameter values and the maze size/complexity. We can clearly see this e.g. with evaporation rates. Evaporation rates for the 'small' easy maze are higher than the evaporation rates for the medium maze which are again higher than the evaporation rates for the 'big' hard maze. This can be explained by the fact that for the hard maze there are a some open spaces. We do not want the ants to unnecessarily 'explore' every square in this open space. or get stuck in them. We want them to find exits and just quickly get through the space. By making sure that pheromones evaporate less fast, the best route through the open spaces are still noticeable to other ants and they will have lower probabilities of 'exploring' this open space. Another interesting parameter is the exploitation probability. The 'optimal' exploitation probability increases when the maze gets bigger and more complex. This is likely due to the fact that in small, non-complex mazes we can allow ants to 'explore' a lot. On the other hand, in big complex mazes, the ants perform better if ants that have travelled a relatively short route can vaguely 'show the way'. That's why the exploitation probability is high for these mazes. The ants will then pick directions purely based on pheromone amount. Of course this has to be correctly regulated to assure that we do not get stuck in a local optimum.

3 TRAVELING ROBOT PROBLEM

The second part of this assignment is dedicated to the implementation of a *Genetic Algorithm* such that the robot can find the order in which it needs to pick up the products in the maze to maintain the short route. We call this the *Travelling Robot Problem* due to its resemblance to the *Travelling Salesman Problem.*

3.1 Travelling Salesman Problem

The original TSP is a problem that asks the following question: "Given cities and distances between pairs of cities, what is the shortest route that visits all the cities once and returns to the start city?". The original TSP is NP-hard in combinatorial optimization[6].

3.2 Problem topology

Our problem is different in a way that the cities are now products that need to be visited and links between products are routes through the maze from one product to another. However, some complexity is added. For example products can have multiple routes to one another. Also, products that are very close to each other in the maze could have very long routes to each other. We also add a start and end point. We must thus find the shortest route[2] such that the robot travels from the given start to end point while picking up every product once.

3.3 CI approach

Unfortunately, if we were to brute force all the possible product sequence combinations, we would end up with a factorial time complexity. With today's technology this problem would quickly become computationally infeasible. This is where computational intelligence comes into play. We will first use our *Swarm Intelligence* from the first part of this assignment (ACO) to produce the best possible routes between all the possible pairs of products. Afterwards we will use the evolutionary method of *Genetic Algorithms* to find out which sequence of products the robot should use to travel the shortest distance. Since the solution space is so big (factorial amount of permutations) we need genetic algorithms to guide us to the most viable sequence.

3.4 Chromosomes and genes

A chromosome is a sequence of genes. In our case, genes will be products such that chromosomes will be sequences of products that the robot goes through. We encode the genes as integers representing the product id's. The chromosome will be a list of these genes. The order of the genes will be the order in which the robot has to pick up the products in the maze. Since order is an important property to ensure evolution, we have to keep this in mind when designing and implementing the genetic operations.

[2]We will use our developed ACO swarm intelligence to compute best routes. So we must remember that ACO is stochastic and does not always yield the shortest path, but a short path which is hopefully short enough (or the shortest).

3.5 Fitness

The goal is to find the 'best' chromosome. In other words, the sequence of products such that the robot travels the shortest path to get to all these products. To quantify the 'goodness' of a chromosome we define a fitness function. Chromosomes with a high fitness are better than chromosomes with a low fitness. The fitness is computed as the inverse of the length of the route L that the robot has to travel. We visualize the genes of chromosome X and define it's fitness f as follows:

$$Chromosome\ X : [gene_1, gene_2, ..., gene_n]$$

$$L = d(start, X[0]) + \sum_{i=0}^{n-1} d(X[i], X[i+1]) + d(X[n-1], end) + n \quad (1)$$

$$f(X) = L^{-1}$$

The function d(_, _) computes the distance between two points in the maze. The length of the route is the distance from the start point to the first product plus the distances between all consecutive products plus the distance from the last product to the end point plus the amount of products. We add the amount of products to the length of the route because otherwise those squares in the maze are not accounted for while they contribute in the length of the route. Finally the fitness is the inverse of this length. This means that chromosomes that encode longer routes will have a worse fitness than a chromosome that encode a short route.

3.6 Parent selection

An important part of the genetic algorithm is parent selection. How do you select the chromosomes that will be part of the genetic operations? For this problem we used fitness ratios and the process of the roulette wheel selection[3]. After computing the fitness of all the chromosomes in the population. We consider the fitness ratios. The fitness ratio r of a chromosome i is computed as follows:

$$r = \frac{f_i}{\sum_{j=0}^{n} f_j}$$

The fitness ration of a chromosome is its fitness divided by the sum of fitness of the chromosomes in the population. We can interpret these fitness ratio's as a probability distribution. The roulette wheel selection acts as a sampler which will randomly select a chromosome of this probability distribution[4]. Thus, the probability of a chromosome being selected as a parent is proportional to the fitness of that chromosome. Meaning that a higher fitness will lead to a higher probability of being selected and a lower fitness will lead to lower probabilities.

3.7 Genetic operations

The key genetic algorithm operations are: selection, cross-over and mutation. We covered the selection operation in 3.6. Let's now cover cross-over and mutation. Cross-over will allow us to perform recombination between two chromosomes. If this recombination happens between two fit chromosomes, it will likely yield even fitter chromosomes (exploitation). Mutation will allow us to perform

[3]Also known as the fitness proportionate selection.
[4]Roulette wheel selection can be seen as a process in which a wheel is proportionally divided into sections corresponding to fitness ratio's. Pick a random number and check into which slice of the wheel we fall.

random changes in recombined chromosomes. This process could sometimes yield better solutions (exploration) that can then be further recombined (exploitation).

We are handling a sequence of products in which the order is important (in fact the essence). Thus we cannot use a simulation of traditional biological cross-over as this could bring a chromosome in an invalid state (missing products, duplicate products). Therefore we use a variant of cross-over called ordered cross-over. Ordered cross-over between arbitrary chromosomes *A* and *B* will be performed in the following way to get two recombined chromosomes *C* an *D*. Chromosome *A* will cut out a random consecutive section of its genes to be the base for chromosome *C*. Chromosome *B* will fill the missing genes (products) for *C* in the order in which these missing products occur in *B*. This way the chromosomes are valid because they will contain no duplicate genes and order will, in a way, be maintained. Analogously chromosome *D* will be obtained by performing the ordered cross-over in the reverse direction (cutting out of *B* and filling with *A*). This way we get two valid, recombined chromosomes *C* and *D* which will hopefully contribute to the evolution.

Mutation should introduce unexpected, random changes in the chromosomes to help explore the solution space and possibly get out of a local optimum. We perform mutation by swapping two random genes in the chromosome. This way we keep order for the rest of the genes and make sure to stay out of invalid states (missing products, duplicate products). Cross-over and mutation both have a certain probability of happening to a chromosome. This way not all chromosomes will recombine and mutate at every generation. These two probabilities are to be tuned for the best outcome.

3.8 Duplicate visit prevention

An important property of the problem is that products should only be visited once. Meaning that chromosomes cannot have duplicate genes because that would mean that a product is visited twice (and one is not visited at all). To prevent the chromosomes in the population from entering this invalid state we implement the genetic operations in such a way that order and valid state (no duplication) are preserved. The way this is done is explained in 3.7.

3.9 Local optimum prevention

To get out of local optima we can partially rely on the genetic operation: mutation. Mutation ensures that some chromosomes will randomly introduce change in their genes. This can than hopefully result in a fitter chromosome that will further contribute to evolution. When a mutation does not lead to a fitter chromosome, the chromosome will less likely be chosen for genetic operations and 'die'. Additionally, we never state that only the fittest chromosomes are selected. Sometimes 'less fit' chromosomes will be used for genetic operations (explained in 3.6). When recombining and mutating those chromosomes we could 'unexpectedly' get fitter chromosomes and thus hopefully leave a local optimum.

3.10 Elitism

Elitism is a strategy where the fittest chromosomes (the elite material) from the current generation are carried over to the next generation without any alterations (i.e. without applying genetic operations). By forwarding a predefined amount of fittest chromosomes without altering them to the next generation, we guarantee that the best fitness per generation will not unnecessarily decrease [1]. We have applied elitism in our genetic algorithm by introducing a new parameter, *eliteProportion*. This parameter controls the proportion of the population that will be labeled 'elite'. Additionally, we make sure that the amount of elites is at least 1. This way our genetic algorithm selects an elite group that will be present in the next generation (without alterations) and performs genetic operations on the whole population like before. It is needed to mention that the group of fit chromosomes that get labeled 'elite' will still be present in the population for the standard genetic operations. This way we keep the elite material and we ensure the best possible probabilities of having favorable genetic operations.

4 RESULTS

In order to discuss the results. We must make sure that the algorithm halts at a point in time. In order to be certain that our algorithm halts, we let it run until a maximum number of generations have passed or until a convergence criterion is met. The convergence criterion that we use works as follows. We say that the algorithm has 'converged' if, for a predefined amount of generations, the algorithm did not improve.

4.1 ACO

These are the results for the *Ant Colony Optimization* obtained using the aformentioned convergence criterion. Firstly, we will depict the length of the best routes that our ACO algorithm finds. Additionally, we show the mean and standard deviation for the length of the best route over 20 trials in each maze. Table 1 shows the results:

	Mean	Std. dev.	Best	Mean exec. time (s)
Easy	38.000	0.000	38	4.342
Medium	130.300	3.363	127	37.780
Hard	783.300	2.777	777	65.196
Insane	273.400	15.525	255	145.217

Table 1: Ant Colony Optimization results per maze

We got these results by running the algorithm with the parameters shown in Table 2.

	Easy	Medium	Hard
ants / gen.	200	150	150
Q	500	2000	500
local evap.	0.6	0.2	0.1
global evap.	0.6	0.2	0.1
exploit prob.	0.0	0.7	0.9

Table 2: Best parameters for ACO

4.2 Travelling Robot Problem

To get representative results for the *Travelling Robot Problem*, we run the algorithm multiple times with the best parameters found for ACO (Table 2) and the best parameters for the genetic algorithm (Table 3).

population size	20
generations	1600
cross-over prob.	0.7
mutation prob.	0.03
elite prop.	0.1

Table 3: Best parameters for the genetic algorithm

The evolution of the chromosomes can clearly be seen in the performance graph (Figure 1) where we compare the fittest chromosome against the average fittest chromosome per generation.

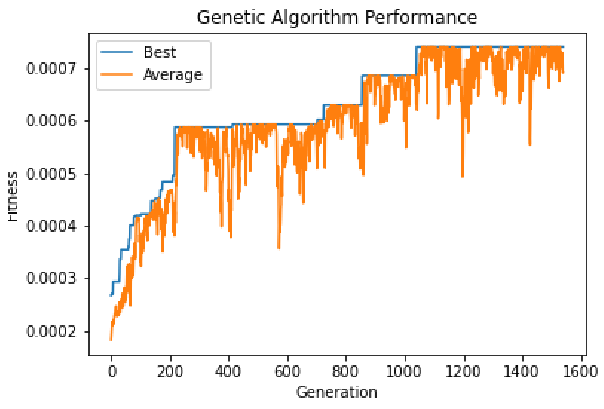


Figure 1: Genetic algorithm performance

The best sequence of products found by our genetic algorithm was the following:

$$1 \longrightarrow 2 \longrightarrow 7 \longrightarrow 5 \longrightarrow 14 \longrightarrow 16 \longrightarrow 4 \longrightarrow 9 \longrightarrow 18 \longrightarrow 8$$

$$\longrightarrow 10 \longrightarrow 15 \longrightarrow 12 \longrightarrow 13 \longrightarrow 6 \longrightarrow 11 \longrightarrow 3 \longrightarrow 17$$

The products are shown by their product ID. This resulted in a route of length 1349 (computed with Equation 1).

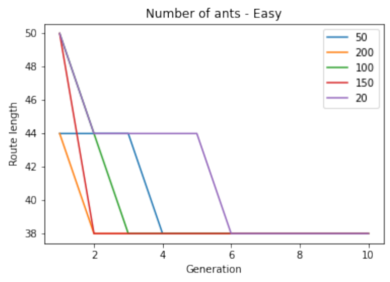
5 CONCLUSION

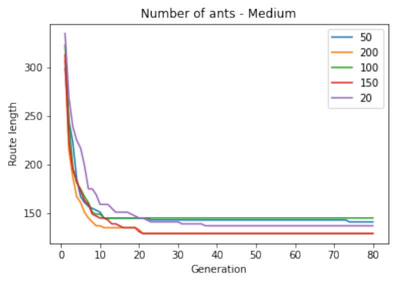
For this assignment we started by developing a basic *Swarm Intelligence* based on *Ant Colony Optimization*. We started by implementing a basic *Ant System* and later enhanced it to a an *Ant Colony System*. The robot can now find short routes through the maze to get from a starting point to an ending point. This *Swarm Intelligence* was then used to compute short routes between the 18 products that the robot had to find in the maze. Subsequently, we developed a genetic algorithm extended with elitism and adapted genetic operations. This was used to find (by evolution) which sequence of products would lead to the shortest route for the robot to travel through the maze.

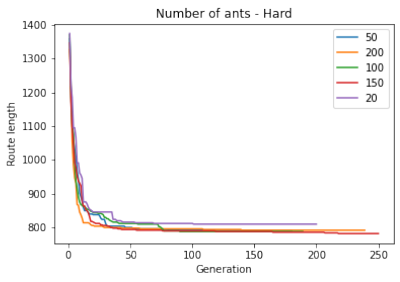
We could still improve the performance of the algorithms by e.g. doing more extensive parameter optimizations (more tested parameter values). Adapting the algorithms to handle even more complex mazes with special features can also be a potential improvement, but this was not the essence of this assignment. Furthermore, there are a lot of ACO algorithms that we could implement and test. Examples of these algorithms are *Rank Based Ant Systems*[2], *MAX-MIN Ant Systems*[8] and *Continuous Orthogonal Ant Colony*[5]. It is not guaranteed that these will result in better ant paths, but we could compare it to our current algorithm and potentially improve it.

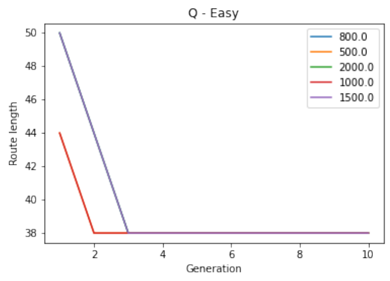
We found this assignment to be a good, interactive learning experience about evolutionary and biologically inspired computational intelligence algorithms.

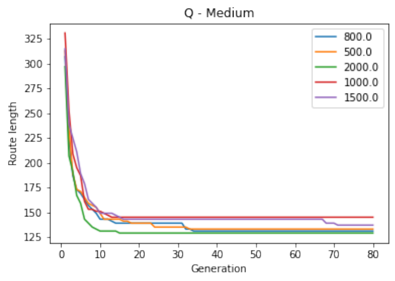
6 FIGURES

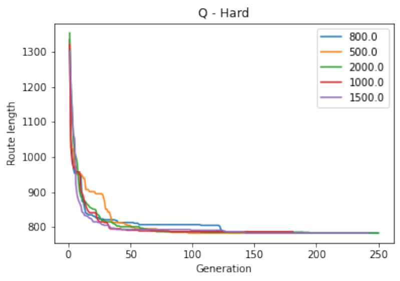

Number of ants - Easy
50
200
100
150
20
Route length
Generation

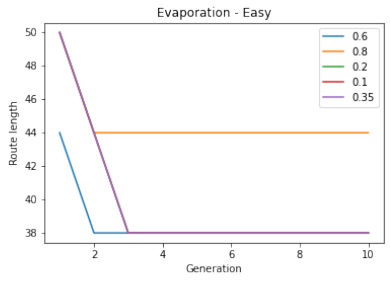

Number of ants - Medium
50
200
100
150
20
Route length
Generation

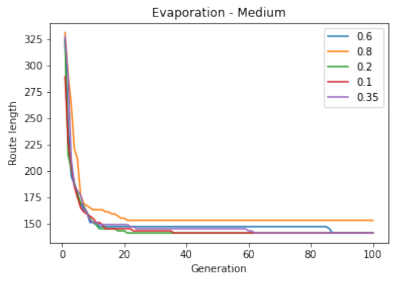

Number of ants - Hard
50
200
100
150
20
Route length
Generation

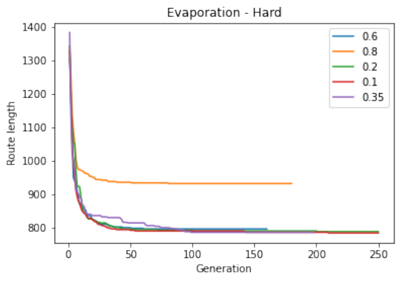

Q - Easy
800.0
500.0
2000.0
1000.0
1500.0
Route length
Generation

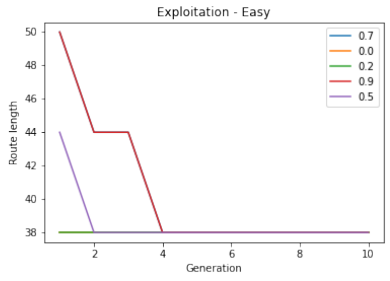

Q - Medium
800.0
500.0
2000.0
1000.0
1500.0
Route length
Generation

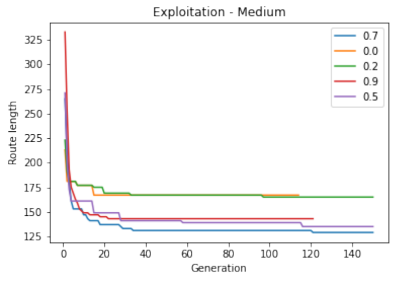

Q - Hard
800.0
500.0
2000.0
1000.0
1500.0
Route length
Generation

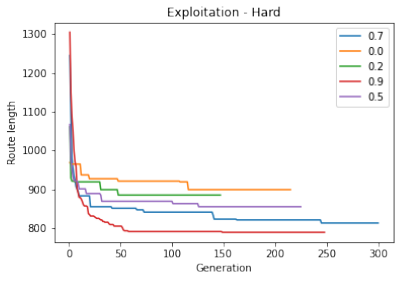

Evaporation - Easy
0.6
0.8
0.2
0.1
0.35
Route length
Generation


Evaporation - Medium
0.6
0.8
0.2
0.1
0.35
Route length
Generation


Evaporation - Hard
0.6
0.8
0.2
0.1
0.35
Route length
Generation


Exploitation - Easy
0.7
0.0
0.2
0.9
0.5
Route length
Generation


Exploitation - Medium
0.7
0.0
0.2
0.9
0.5
Route length
Generation


Exploitation - Hard
0.7
0.0
0.2
0.9
0.5
Route length
Generation

REFERENCES

[1] Shumeet Baluja and Rich Caruana. 1995. Removing the Genetics from the Standard Genetic Algorithm. (1995).

[2] Bernd Bullnheimer, Richard Hartl, and Christine Strauss. 1999. A New Rank Based Version of the Ant System - A Computational Study. *Central European Journal of Operations Research* 7 (01 1999).

[3] Marco Dorigo and Luca Maria Gambardella. 1997. Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997).

[4] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. 1996. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics–Part B* 26, 1 (1996).

[5] X. Hu, J. Zhang, and Y. Li. 2008. Orthogonal methods based ant colony search for solving continuous optimization problems. *Journal of Computer Science and Technology* 23, 1 (January 2008).

[6] Ricard Manning Karp. 1972. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations* (1972).

[7] Jean Pelletier-Thibert. 2011. *Academie de Macon*.

[8] Thomas Stützle and Holger H. Hoos. 2000. MAX–MIN Ant System. *Future Generation Computer Systems* 16, 8 (2000).