Towards a Multilevel Ant Colony Optimization

By: Thomas Andreé Lian, Marilex Rea Llave

Supervisor: Morten Goodwin, Associate Professor Ph.D

IKT 590 - Master's thesis

Spring 2014

Department of Information and Communication Technology

Faculty of Engineering and Science

University of Agder

Grimstad, 02 June 2014

Abstract

Ant colony optimization is a metaheuristic approach for solving combinatorial optimization problems which belongs to swarm intelligence techniques. Ant colony optimization algorithms are one of the most successful strands of swarm intelligence which has already shown very good performance in many combinatorial problems and for some real applications.

This thesis introduces a new multilevel approach for ant colony optimization to solve the NP-hard problems shortest path and traveling salesman. We have reviewed different elements of multilevel algorithm which helped us in construction of our proposed multilevel ant colony optimization solution. We for comparison purposes implemented our own multi-threaded variant Dijkstra for solving shortest path to compare it with single level and multilevel ant colony optimization and reviewed different techniques such as genetic algorithms and Dijkstra's algorithm.

Our proposed multilevel ant colony optimization was developed based on the single level ant colony optimization which we both implemented. We have applied the novel multilevel ant colony optimization to solve the shortest path and traveling salesman problem. We show that the multilevel variant of ant colony optimization outperforms single level.

The experimental results conducted demonstrate the overall performance of multilevel in comparison to the single level ant colony optimization, displaying a vast improvement when employing a multilevel approach in contrast to the classical single level approach. These results gave us a better understanding of the problems and provide indications for further research.

Acknowledgements

We would like to express our gratitude to our supervisor Dr. Morten Goodwin for making this Master Thesis possible. For the continuous support and help with in formulating a good master’s problem, as well as being available to discuss ideas and concepts being met by well-reasoned arguments along the way.

Thomas would like to thank his fiancé Silje Wang Jensen, with the unquestionable support allowing him to work night and evening shifts, weekdays and weekends as a security analyst at NTT Com Security, as well as working on his master’s thesis. It would not have been possible without their patience and help keeping him in harmony and reminding him to talk about something else than work and ants.

Marilex would like to thank her fiancé Jørgen Tobiassen Bornø for everything because none of this would ever been possible without the love and the fantastic support from him. Last but definitely not the least she wishes to thank her family in the Philippines, her beloved mother Celia Rea Llave and two sisters Maricel and Lexmaricel for being understanding, supportive and contributing to their encouragement to the completion of this work.

List of Figures

List of Abbreviations and Acronyms

ACO – Ant Colony Optimization	

ACO – Ant Colony Optimization
AI – Artificial Intelligence
AS – Ant System
CO – Combinatorial Optimization
EAX – Edge Assembly Crossover
FDS – Fire Dynamics Simulator
FN – Furthest Neighbor
GA – Genetic Algorithm
GCP – Graph Partitioning Problem
IGA – Insular Genetic Algorithm
IRP – Internet Routing Protocol
ItDMACA – Interactive Distributed Multilevel Ant Colony Algorithm
MACA – Multilevel Ant Colony Algorithm
MACO – Multilevel Ant Colony Optimization
MB – Mega Byte
MMAS – MAX-MIN Ant System
ML – Multilevel
ML# – Multi Level # (No. of levels)
MG – Multi Grid
MG# – Multi Grid # (No. of levels)
MG-ACO – Multi Grid Ant Colony Optimization
ML-ACO – Multi Level Ant Colony Optimization
MT-Dijk – Multi Threaded Dijkstra
NN – Nearest Neighbor
NP – Non polynomial
OSPF – Open Shortest Path First
PDACO – Population Declining Ant Colony Algorithm
PDMMAS – Population Declining MAX-MIN Ant System
RPG – Role Playing Game
SIDMACA – Semi-Independent Distributed Multilevel Ant Colony Algorithm
SPACO – Shortest Path Ant Colony Optimization
TSP – Traveling Salesman Problem

Table of Contents

Chapter 1 – Introduction

Within the world we live, there exists a multitude of problems for which logical “smart” solutions cannot solve them within the lifetime of humanity these are the so-called NP-Hard problems. But for several of these problems there exists solutions within nature that results in good enough solutions. In multiple cases if one where to simulate nature the problems becomes solvable by not looking for the best solution but a good enough solution. For many of these solutions and problems, the cost of getting the last percent of improvement takes as much time as the first 99 percent of improvements.

There exist several nature copies in various environments, those being the simulation of ant behavior, flocking of birds and other animals, genes, neurons and other artificial intelligence algorithms. What these all have in common is that they act in an intentionally way to create un-foreseen solutions, their behavior or movement is not pre-determined and they system will churn out a different “good” solution every time.

In our thesis we look at the Ant Colony Optimization (ACO) in particular. ACO is a probabilistic technique for solving computational problems in finding good paths through graph [1]. ACO is used to solve both practical optimization and combination optimization problems.

The first ACO algorithms were introduced in the early 1990’s by Marco Dorigo in his Ph.D. thesis [2]. The development of ACO algorithms was inspired by the real ant colonies. More specifically, it was inspired by the observation of their foraging behavior and how they can find the shortest paths between their nest and food source as shown in Figure 1 below.

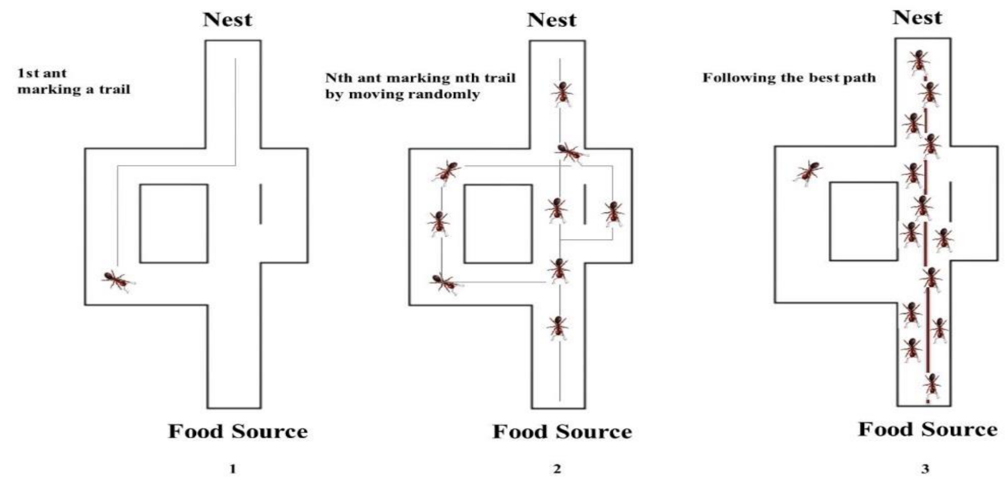


Figure 1 : Self-adaptive behavior of a real ant colony

Ants communicate with each other by producing scented chemicals known as pheromones. These pheromones are created from glands found all over their bodies. When ants are hunting for food, they explore an area randomly. While exploring, ants leave pheromones trail on the ground so that other ants can smell it. As soon as they find the location of the food source, ants will deposit pheromones depending on the quality and quantity of the food they carry. In choosing paths for the next ants, they will likely to follow the paths with strong pheromone concentrations enables them to find the shortest path. This indirect communication of ants via pheromone trails also known as stigmergy [3].

Stigmergy is an important mechanism for swarm intelligence [4]. This kind of stigmergetic information has a huge impact in the utilization of collective knowledge. For instance, its effect to change the way the environment is locally perceived by the ants as a function of all past history of the colony. As pheromones are subject to evaporate over time when no new pheromones are deposited, the strength of pheromones become weaker. Then the ant colony will forget about the past history but will still continue to search towards new direction.

ACO algorithms are one of the most successful strands of swarm intelligence [5]. This swarm intelligence technique has been shown to perform well in solving shortest route problems [6], scheduling problems [7], and assignment problems [8]. However the ability of swarm intelligence was not limited to these types of problems as has been shown to work even within the field of image processing [9].

Finding the shortest path at minimum cost from the nest to the destination or food is the objective for both artificial and real ants [10]. Real ants do not jump; they explore the environment to find a path to reach the food. And they are capable of adapting to changes in the environment. For instance, finding new shortest path once the old the path is no longer feasible due to a new obstacle. Artificial ants has the same behavior, they are exploring step by step through next state of the problem without looking ahead just like real ants.

In Figure 2A real ants follow a path between nest and the food source. In Figure 2B an obstacle is interposed and it shows the equal probability that ants will choose to turn left or right. Ants choosing the shortest path will rapidly reconstruct the interrupted pheromone trail compared with those ants choosing the longer path. Therefore shortest path will be rich in pheromones and the larger number of ants will likely to choose the same path as shown in Figure 2C.

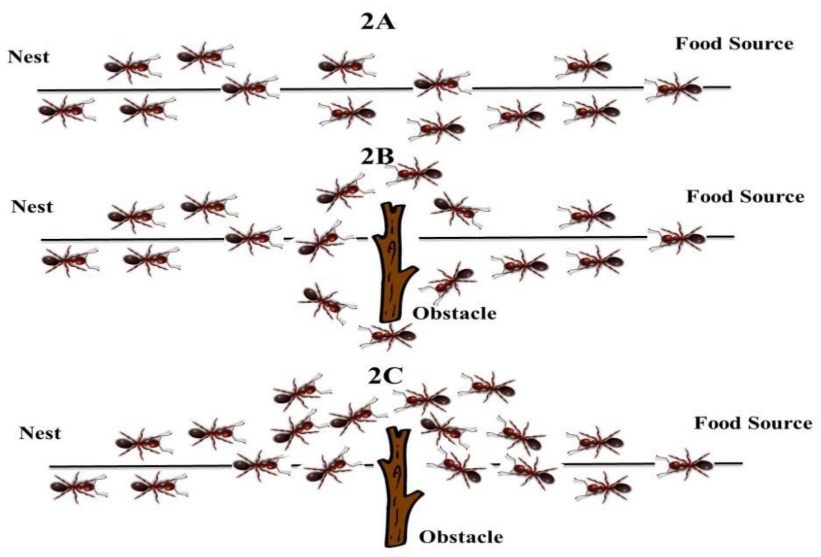


Figure 2: Real ant behavior in finding shortest path between nest and food source

To improve the overall efficiency of ACO, there are now many variations of ACO with extra capabilities. Some of the most popular variants of ACO algorithms include Ant System (AS) [6], Ant Colony System (ACS) [11], Max-Min ant system (MMAS) [12], Population declining ant colony optimization algorithm (PDACO) [13], and Elitist ant system [14]. In [15] they have made an analysis of the effectiveness of these variations through observation of its several properties.

1.1 Background

In this thesis we will go through the elements of pathfinder and the traveling salesman problem (TSP). These are problems where there are no simple & cheap solutions for.

When performing path finding over a 2 dimensional area where one wants to go from position A to B in the shortest amount of distance there are several ways to achieve this. One of these solutions is the algorithm referred to as Dijkstra's algorithm which performs a type of distance check to every position between start and stop in a radial manner this can be read further in 3.1 General Path Finding. This is a good algorithm for multiple problems, but it quickly becomes too expensive as it has a cost of $O(|N^2|)$ meaning that the cost will raise exponentially as the complexity of the problem increases.

For the TSP, the problem is to visit every city in an area once in the least amount of cost. This problem differs from the shortest path in the way that in the shortest path one can only go from one point to the surrounding points. Whereas in the TSP every point is connected to every other point and giving rise to a problem therefore can be defined as NP-Hard. This means that the problem itself cannot be solved in polynomial time. For TSP there are also several ways to attempt to solve it, some examples of this is the use of genetic algorithms (GA) which can be read more in 3.1.3.1 Genetic Algorithms (GA).

To solve these problems there is an algorithm called ant colony optimization (ACO). ACO can be explained in simple terms. Let's say there is a room with two doors, lights turned off and filled with thick fog so you can only see 10 centimeter in front of you. Since we do not know anything of the inside of the room then we can assume it has any shape or objects in it and the second door can be placed anywhere. So how do we solve this problem because it sounds pretty much impossible?

Let's give all the people entering the room one at a time shoes with fluorescent paint which diminishes over time on the soles. Now when the first person enters he will walk on random in the room until he exits, now the next person will have a base line, he can choose to either follow the paint or to do his own path, then over several people we will have a path most traveled, while the least traveled paths will have dissipated leaving us with a decent path, and might even the best path.

Path finding is simply the plotting of the shortest route between the initial position and the final location potentially in difficult and unknown areas. Path finding algorithms (PFA) has many different usage areas such as computer games [16], robot motion and navigation [17] , automated vehicles [18], and transportation networks [19].

In game programming technologies path finding is an important element. Path finding strategies are for the most part employed as the core of any Artificial Intelligence (AI) movement system. To know how to get from point A to point B is something many computer games require. Whether it is a role playing game (RPG) or a simple puzzle game, navigating the game world needs more intelligence from the game character.

The game agents should not walk through walls, get stuck or failed to make their way to the target destination. This is where path finding comes in to play as an important part of computer games. These strategies will find a path from any coordinate in the game world in which game agents will move according to the calculated path. Therefore it is considered as the basic building block of most computer games.

Agent movement is the greatest challenge in the design of realistic AI. In this study they presented how path finders were used in computer games and their weak points particularly when dealing with real-time path finding [20]. They also focused on how machine learning techniques can be used to improve agents' ability to deal with real-time path finding.

Path finders are also useful in field of robotics [17]. These algorithms are used to guide robots especially around difficult terrain without human intervention. For instance the Mars Pathfinder as shown in Figure 3 [21] was landed a base station with a roving probe on Mars in 1997 [22]. The Mars Pathfinder successfully roamed on the surface of the planet while sensing the terrain. Moreover it had returned an unprecedented amount of data such as chemical analysis of rocks and soils, extensive data on winds and other weather means. Mobile robots roaming in indoor and outdoor environments have navigation and path finding units for them to move autonomously.

Figure 3: Mars Pathfinder Sojourner Rover [21]

Figure 4: Pharos in Autonomous Vehicle Competition [24]

Automated or driverless vehicles are another usage area of path finding [18]. Path finders for automated vehicles are generally stated as moving from one place to another. One of the most known development technologies for autonomous cars is Google's car. The Google driverless car has the ability to sense the environment and navigate without human input [23]. This means that automated vehicle will be able to find its way while avoiding the obstacles and reach its goal. The autonomous vehicle named Pharos as shown in Figure 4 which participated in 2010 Autonomous Vehicle Competition was introduced in this study [24].

They proposed a solution for high-speed vehicle motion control, real-time path planning for obstacle avoidance and long range terrain perception. In [25] they generally stated that if complete and accurate information about the environment will be available it will be sufficient to use a standard path finder algorithm.

In transportation networks, shortest path needs to be immediately determined. This is mainly due to road network applications whereas an immediate response is required. Therefore most of transportation applications will depend upon heuristic shortest path algorithm rather than standard path finding algorithms [19]. Figure 5 [26] shows a proposed solution was successfully extracted the structure of geographic

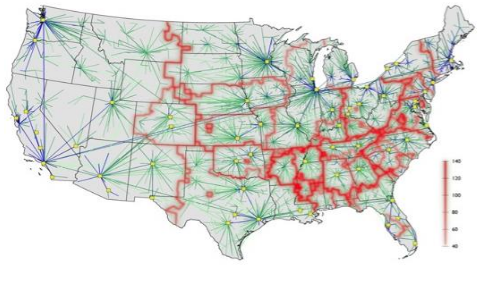


Figure 5: Effective Borders and Shortest Path Trees [26]

borders inherent in multi-scale mobility networks where a new and efficient computational technique was presented. Furthermore this technique identified a unique set of connections in the network and a network backbone that correlated strongly. Most of these algorithms that are commonly used are group depending on their simplicity and good complexity.

1.2 Report Outline

In this master thesis, we will present a novel multilevel approach to ACO. The performance of multilevel will be tested through comparability with other solutions within path finding such as Single-level ACO, Dijkstra's algorithm and our own multi-threaded Dijkstra (MT-Dijk) implementation. We will also perform a comparative test on multilevel ACO on the TSP comparing against single level ACO, greedy (Nearest and furthest neighbor) as well as a random for a baseline.

The thesis is structured in several chapters where each tackling their own feature or area. The introduction of ant colony solutions in general in Chapter 1 – Introduction is followed by a short recap of the research in the area within both traveling salesman and shortest path problems in 1.1 Background.

In the following Chapter 2 – Problem Definition will present the goal of the research as well as some of the means for providing the results and comparability of the research.

In Chapter 3 – State of the Art deal with the summary of some researches within several fields related to pathfinders, multilevel algorithms and use of ACO within search problems for shortest path compared against Genetic Algorithm (GA) and Dijkstra followed up by a similar review on the use of GA and ACO within TSP.

Then for Chapter 4 – Proposed Solution, the simulation setups and environment is described as well detailing the choices made within the simulations. Also its performance with various modifications to runtime variables, for which the future research and comparisons where made upon allowing for coherent results to be collected.

All experimental results are compiled and reviewed in Chapter 5 – Experiments & Results. And the conclusion of the research can be found in Chapter 6 – Conclusion followed by the indications of future work in Chapter 7 – Future Work.

Chapter 2 – Problem Definition

The goal of this study is to research and develop methods for enabling a multilevel variant of ACO. The way we propose to obtain the project goal is by deploying a multilevel approach to the ACO vertex scheme. By conducting empirical experiments with symmetric TSP and shortest path problem we will achieve knowledge from both direct and indirect observation. So we are able to test the multilevel variants of ACO which has not been done before.

The methodology used is to comparatively show how multilevel variants of ACO perform in comparison to single level ACO, Dijkstra's algorithm and other solutions within path finding. The comparisons will be done over series of synthetic and realistic tests measuring their individual capabilities under certain circumstances. By giving the data ranging from path distance, time and various resource consumptions, it will provide a comparative result upon the performance of a multilevel ACO.

The importance of this study is to show that multilevel variant of ACO can be deployed to increase the performance of ACO algorithm. This means it will help to improve the optimizations ability of ACO and reduce the computation time in obtaining the optimal solution.

The multilevel scheme is expected to be better than single level since vertices are grouped into vertex collections, allowing for the vertex set to be simplified and allowing for sets to become easier to solve as the amount of choices is reduced.

Chapter 3 – State of the Art

This chapter will discuss the well-known general path finding followed by defining what heuristic and metaheuristic approaches are. The objective of this chapter is to present the elements we used for constructing the multilevel scheme as well as presenting an overview of different metaheuristic approaches applied in solving shortest path problem and TSP. In the last section will be a discussion on the limitations of existing solutions for the shortest path and TSP.

3.1 General Path Finding

Within path finding there multitudes of algorithms aimed at solving the path finding problems. The so called Dijkstra's algorithm is viewed as the most well-known method for finding optimal path's in several optimal path or route problems [27]. The method finds the optimal path from its initial position to all other positions within the problem.

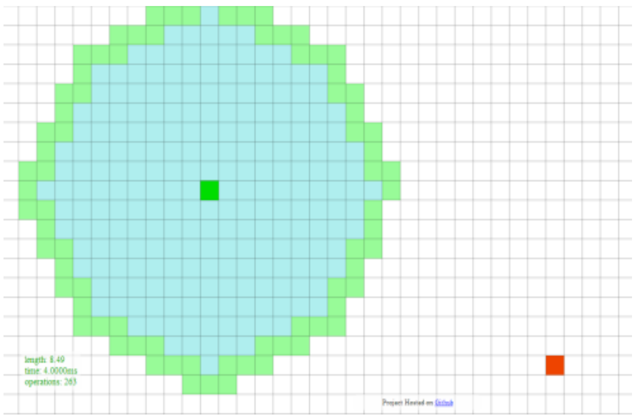


Figure 6: Dijkstra's algorithm searching area schematic diagram [28]

This is done by employing a concentric expansion as shown in Figure 6 [28], having each vertex having a set of children. This allows for taking a children and following the lineage, in that way the optimal path can be fetched from the child to the start. This method checks every vertex as it performs no prediction of the usefulness of any future or current vertex or child in the world before it reaches its target destination. This way the algorithm has to keep track of the cheapest discovered cost of traveling from the initial vertex to every vertex visited.

Dijkstra's method is by these facts an exponentially expensive problem to run, as it needs to go through $O(|Vertices^2|)$ making it an exponentially expensive problem as the vertex count increases. The A* algorithm is an extension of Dijkstra's algorithm [29] using heuristic method to estimate how seemingly good vertex is close to the target, allowing it the ability to eliminate "fruitless" branches.

3.1.1 Heuristic

Heuristic search methods are used to accelerate the process of finding a satisfactory solution instead of optimal solution when time resources are limited [30]. It is an important technique for solving different problems in artificial intelligence and operations research. The heuristic method can be divided into two types: Construction Heuristic and Improvement Heuristic.

A construct heuristic determines a tour according to some construction rules and will never try to improve upon this tour [31]. Whereas a tour is continuously built and parts which are already built will remain unchanged throughout the algorithm. Furthermore construction heuristics produce a solution incrementally and by iteration a new element will be selected to combine the model in solution search to the problem [32].

The improvement heuristics are used to define a set of solution changes, which represents alternative moves that can be taken during optimization. These types of heuristics are commonly used to a complete initial solution until the time of search is reached. In this manner, the initial solution can be created randomly [32].

One of the well-known heuristics is A* algorithm which builds from Dijkstra's algorithm and can make a choice on what vertex to grow on before others.

3.1.2 Metaheuristic

Metaheuristics is an iterative high-level process which guides and modifies the subordinate heuristics operations that may provide a sufficiently good solution to an optimization problem [33]. Genetic algorithms (GA) and ant colony optimization (ACO) are good examples of classical metaheuristics. Each of these follows different paradigms and philosophies in solving different optimization problems.

3.1.3.1 Genetic Algorithms (GA)

GA have been successfully studied to be able to solve complex problems in both engineering and other scientific fields. This algorithm was inspired by the biological approach within evolution, using concepts from natural selection and genetic inheritance to be able to adapt to a given environment be it static or changing.

Evolution starts by randomly generating a population pool of solutions. In each generation, the individuals are evaluated by a function adequate to the given problem in order to determine the fittest individuals. The best ranking individuals will be selected as parents to breed new generation. Through mutation and crossover it is more possible to construct a better population by iteration. This process will be performed repeatedly until an optimal solution is found or the fixed number of generation is reached. The graphical representation of GA metaheuristics can be seen in Figure 7 [34].

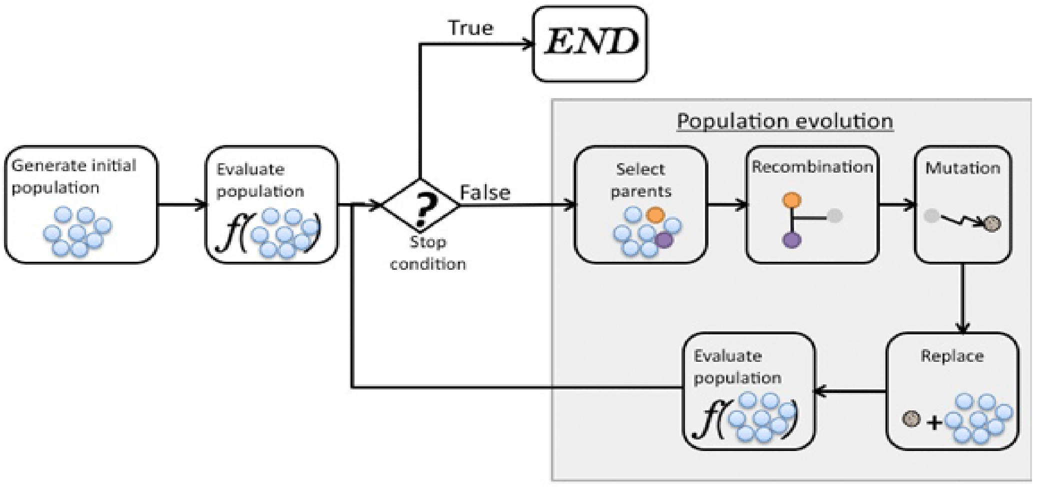


Figure 7: Graphical representation of a GA metaheuristic [34]

3.1.3.2 Ant Colony Optimization (ACO)

The metaheuristic approach ACO has been successfully applied to many applications particularly in combinatorial optimization problems. ACO is defined vaguely containing very few rules on how it function, the only defined elements is the pheromone weighted random movement and that ants leave pheromones on the traversed map in some way. This leaves multiple aspects up to the implementer, such as how to store, increase and decrease pheromones and rules of movement. Some elements can be defined by the problem for an example TSP do not allow re-traversal and other such “rules”. Other than this the ACO is in its initial definition left as a blank slate. ACO framework is shown Figure 8 displaying the basics of ACO metaheuristic in Figure 8 [35].

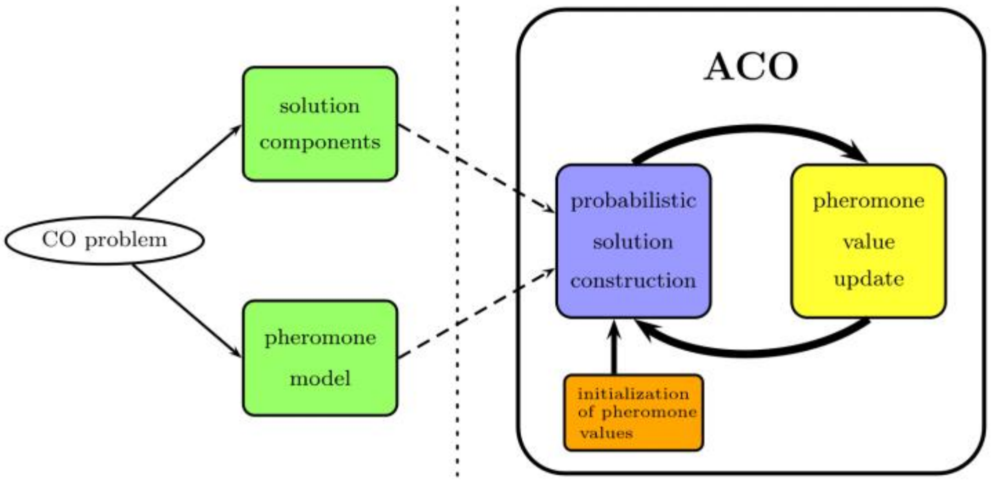

ACO
solution components
CO problem
pheromone model
probabilistic solution construction
pheromone value update
initialization of pheromone values

Figure 8: ACO metaheuristic framework [35]

In general combinatorial optimization (CO) problem one has to check every finite set of solutions in order to determine the best one. With ACO one defines a set of values called a pheromone model to be used in generating a probabilistic solution, reinforcing and improving the pheromone model with each ant ran over the problem.

Figure 9 shows the technical description of ACO metaheuristics which was originally presented by [36]. In ACO metaheuristics a finite size of artificial ant colony can generate best solutions in the shortest amount of time. Therefore it shows itself efficient in solving NP- hard problems.

```
const Phermones = value
VisitedEdges = new List<edges>

RunAnt(from)
      VisitedEdges.Add(from)
      While (initPosition != goalVertice)
            to = roulette_GetNextVertice()
            VisitedEdges.Add(from.via_Edge.to)
            from = to
      Foreach(VisitedEdges as a)
            a.DepositPhermones(Phermones / VisitedEdges.Sum(b=> b.distance))
      Foreach(AllEdges as a)
            a.DecreasePhermone();
```

Figure 9: Pseudo code of ant colony metaheuristics

3.2 Multilevel Algorithms

Multilevel techniques have been used especially in the area of multigrid methods [37]. Most of multilevel techniques were applied to graph-based problems such as mesh partitioning [38], graph coloring [39] and TSP [40]. The multilevel idea was originally proposed by Barnard and Simon to speed up the spectral bisection [41]. And Hendrickson and Leland made an improvement for solving graph partitioning problem [42].

The multilevel algorithm scheme is relatively simple. From the original problem, some sets of smaller problems are made by successive coarsening until the given limit criteria is reached. These set of problems will produce a hierarchy in which a set of problem in a given level is always smaller than the set of problem in the next lower level as shown in Figure 10.

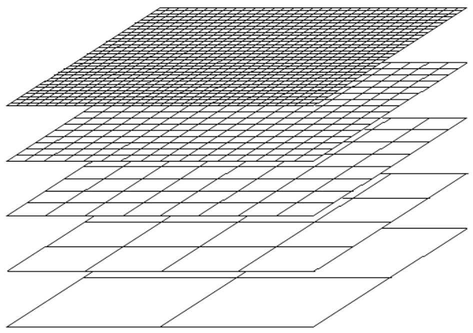


Figure 10: Standard multi grid [58]

Afterwards, a calculated solution for the smallest problem will successively transform into a solution for the next higher level until a solution to the original problem is obtained.

According to Barnard and Simon, multilevel algorithms required three elements which include contraction, interpolation and refinement [41]. In contraction a series of smaller graphs to be constructed must retain the global structure of the original large graph. Interpolation defines a given vector of a contracted graph whereas it is possible to interpolate this vector to the next larger graph with good approximation to the next vector. While in refinement given an approximate vector for a graph, it will be expected to calculate more accurate vector efficiently.

The basic idea of multilevel approaches to vertex sets or other concatenated sets, one usually seeks simple rules for simplifying the sets. One approach for this is to find branches or vertices which can be folded it into single cluster vertices as shown in Figure 11 where 1 and 2 is cluster vertices or sets of $1 = \{B, G, F\}$ and $2 = \{C, D, E\}$ leaving the sets new problem set as $\{1, 2, A\}$. This process can be repeated until there are no more possibilities to simplify the problem.

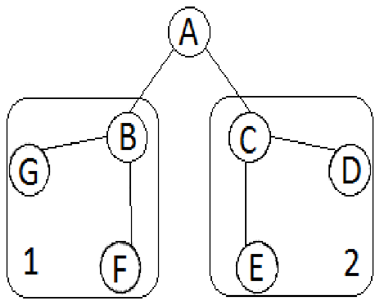


Figure 11: Collapsing of vertices into super vertex

This reduces complexity on each level allowing for the set to be solved at a simple level which eliminates elements from the next and more complex level. This can be performed until reaching a reduced set of the first level which should be substantially simpler to solve than the full set.

Mesh partitioning problem involves partitioning a mesh into a given number of parts then each part has the same number of elements, and the number of vertices that straddle the parts is minimized. It is an important problem that has an extensive application in many areas. Multilevel algorithms are a successful class of optimization techniques which solves mesh partitioning problem. Mesh partitioning was used to investigate the multilevel ACO (MACO) which is a relatively new metaheuristic search technique for solving optimization problem [43]. Their goal was to suggest modification for MACO improvement and to evaluate them experimentally.

Graph partitioning problem is an important component of mesh partitioning. Due to its complexity they mainly concentrate on the partitioning itself. Multilevel techniques group of vertices together to form clusters and these clusters will define a new graph. The process will be performed repeatedly until the size of the graph reach the limit. Figure 12 illustrates the basic idea of multilevel algorithm in solving mesh problem from sample mesh (a) up to partitioned mesh (d).

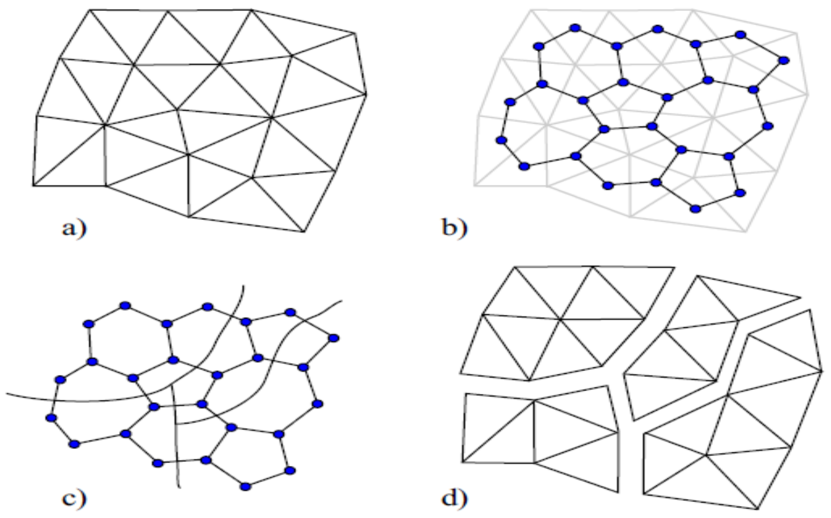


Figure 12: Mesh Partitioning using multilevel algorithm (a) sample mesh (b) mesh with induced graph (c) graph partitioning (d) partitioned mesh [43]

Even though the use of multilevel algorithms already established an effective way to speed up and globally improve the partitioning methods, there are much room for improvement and potential for further investigation. One of the proposed possibilities is to perform the mapping of the graph onto the grid [43]. However a proper mapping convergence is required to attain better results. There is a wide range of possibilities to be considered in the future but the most interesting are the merger of MACO with other method and the possibility of parallel implementation of MACO algorithm.

The interactive distributed multilevel ant colony algorithm (ItDMACA) and semi-independent distributed multilevel ant colony algorithm (SIDMACA) which is based on parallel interactive colony approach were presented in solving mesh partitioning problem [44]. They presented the multilevel framework in Figure 13 that characterizes a level based refinement method in order to increase convergence of the solution to larger problems.

Coarsening is a graph contraction procedure that is iterated L times by finding the largest independent subset of graph edges to collapse them. While refinement serves as a graph expansion procedures on a partitioned graph.

On the left side of Figure 13 the graph was coarsened down to 4 vertices and partitioned into 3 levels. The solution was successively extended and refined as shown on the right side of Figure 13. Moreover a high quality of partitioned was still achieved at each level of refinement.

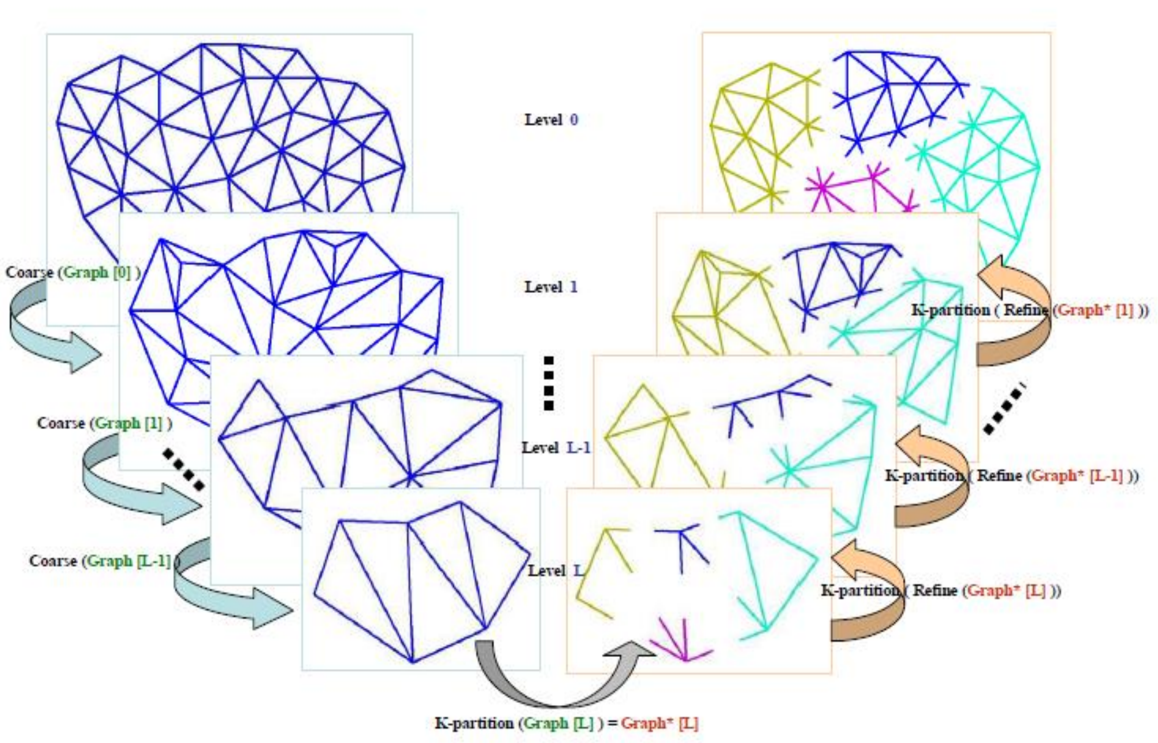

Level 0
Level 1
Level L-1
Level L
Coarse (Graph [0])
Coarse (Graph [1])
Coarse (Graph [L-1])
K-partition (Graph [L]) = Graph* [L]
K-partition (Refine (Graph* [1]))
K-partition (Refine (Graph* [L-1]))
K-partition (Refine (Graph* [L]))

Figure 13: Three phases of multilevel graph partitioning

These two distributed MACA versions shows better quality performance. Nevertheless ItDMACA considered being more sensitive on the parallel performance efficiency according to experimental evaluations and SIDMACA managed to obtain better quality solution for less computational time.

Graph partitioning problem (GCP) is another graph-based problem were multilevel algorithms are used. In GCP, vertices of a graph have assigned colors such that two adjacent vertices will not share the same color and so that the number of colors will be minimized. A graph contraction of 4 coloring graph is shown in Figure 14 [45].

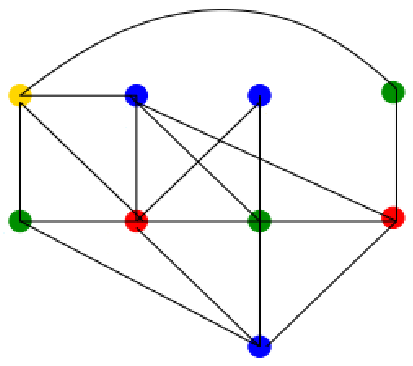


Figure 14: An example of graph contraction for the coloring problem [45]

In 2004 a multilevel algorithm for solving GCP was presented [39]. Since the problem is a graph based, the same coarsening procedures as graph partitioning was used. The matching procedure was also based on algorithms used in graph partitioning with one very important difference. Instead of matching nearest- neighboring vertices, matching are done between those vertices that are not adjacent. The coarsening is terminated once the initialization started.

A multilevel approach was implemented for solving TSP [40]. The multilevel paradigm for TSP is represented in Figure 15. The top row of Figure 15 corresponds to the coarsening process where dotted lines represent matching of vertices. The matching process only chooses optimal edges and then the optimal tour will be found by the end of this process. However in the absence of information for optimal tour, nearest neighbors are used in matching vertices.

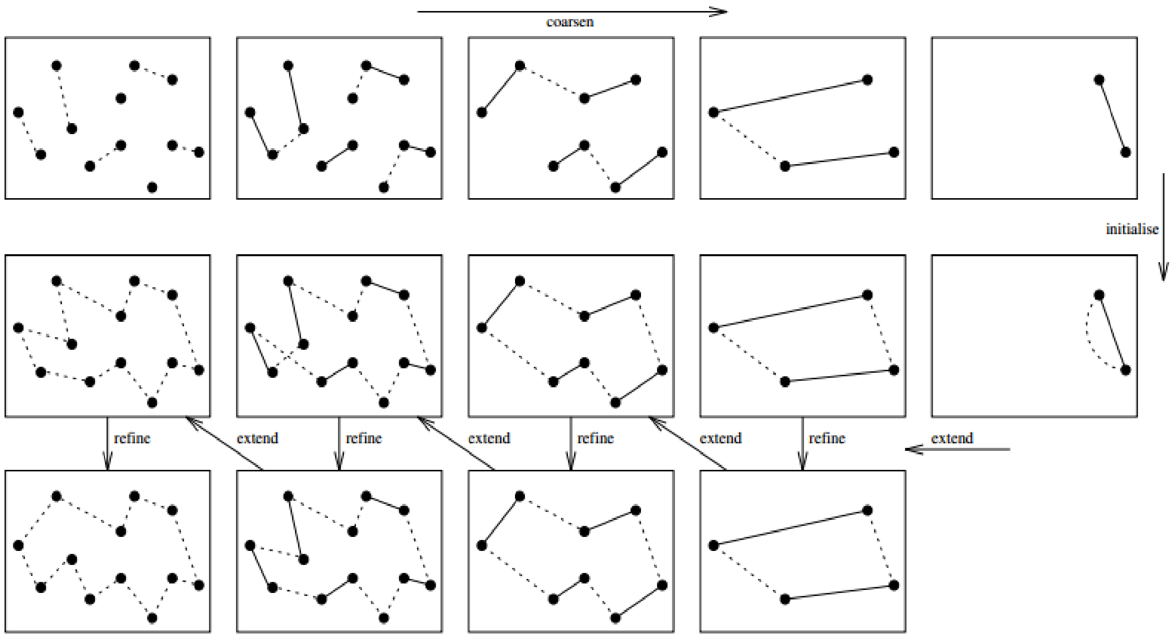


Figure 15: A sample multilevel algorithm for TSP [40]

The only variable parameter they have used within matching process is grid spacing h. For each coarsening level a maximum matching distance h are chosen allowing vertices to be matched with the nearest neighbors. To achieve this, the smallest rectangle containing all the vertices will be found and overlay it with a square grid of spacing h. On the other hand, if there are unmatched vertices found, they will randomly choose a cell containing unmatched vertices. In refinement step, they have used the chained Lin-Kernighan algorithm since it is one of the best heuristic in solving Euclidean TSP tour [46].

From third level onwards the chains of fixed edges were decreased down to a single edge. When the problem was reduced to one fixed edge and two vertices the coarsening process will end. At this point the initialization of tour will begin.

Another common use of multilevel approaches is within graphics where one creates a mesh of a 3D model and then refines the mesh as the view comes closer to details of interest [47]. This is an iterable solution to draw vertices in the distance by employing a sort of dynamic multilevel approach. The main concept is placed around the idea of 3D being polygons and that the more polygons the higher the rendering cost and level of detail can attain.

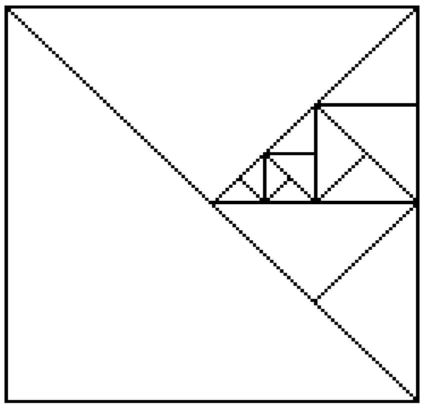


Figure 16: Mesh refinement on vertices of interest.

To enhance the rendering of complex features, a type of dynamic multi-leveling on the mesh needs to employ. This means that the model can dynamically collapses down to a lower polygon model as the model is pulled further and further away from the users view. In Figure 16 an example of several levels can be seen of a polygon where it can in theory be cut in half an infinite amount of times, possibly gaining an infinite amount of detail or one can collapse the polygons and their features to a lower resoluted problem.

3.3 Shortest Path Search Problem

The goal of shortest path problems is to find an existing path between two vertices with the lowest cost measured in a certain way decided by the problem, for an example distance or time. Calculating the shortest path is a vital combinatorial optimization problem in computational geometry and other fields including graph search algorithms, geographical information system, network optimization and robotics.

Among others, the shortest path algorithms are to determine the closest distance between two geographical locations [26], routing in packet networks [48], and to balance network utilization [49]. Undoubtedly, the shortest path is one of the aspects of great significance to telecommunications and navigation. In shortest path algorithms, we are given a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges with assigned cost C_{ij} associated with each edge $(i, j) \in$ E. A path will be considered the shortest path if it has the minimum length from among all other paths that begin and terminate in given vertices.

There are different variants of shortest path problem which involves single pair shortest path, single source shortest path and all pair shortest path [50]. The single pair shortest path finds the shortest path between a pair of vertices. While single source shortest path finds the shortest path from the initial vertex from all vertices. Lastly, all pair shortest path finds the shortest path between all pairs of vertices.

In Figure 17 [51] presents an example solution for the shortest path problem. This particular solution was used as an AI movement in the game Left 4 Dead's programming of zombies to find a "natural" path of movement between present location and target destination.

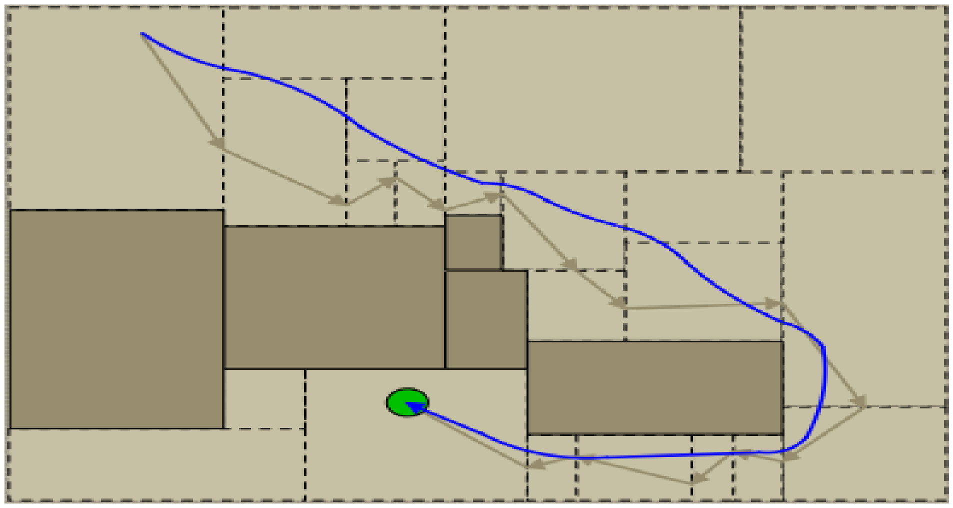


Figure 17: Example of a shortest path problem [51]

The following presents some of the main algorithms that are commonly engaged for the shortest path search problem will be presented together with their respective features. These algorithms include the well-known Dijkstra's algorithm, ACO and GA.

3.3.1 Dijkstra for Shortest Path

The conventional Dijkstra's method provides the shortest path from a source to all other vertices in the map. The search region of Dijkstra generally expands concentrically to check each vertex that makes the search time long. Consequently, this method cannot obtain a solution in real time for larger problems.

An improved Dijkstra was proposed in order to overcome the drawbacks of this method [27]. The extended Dijkstra proposed a method for the shortest path search which limits the concentric expansion of the search region to lessen the number of vertices to be searched. It means that Dijkstra method was applied in both directions so it will search from the beginning to starting point and to target destination. The comparison of both algorithms was shown in Figure 18.

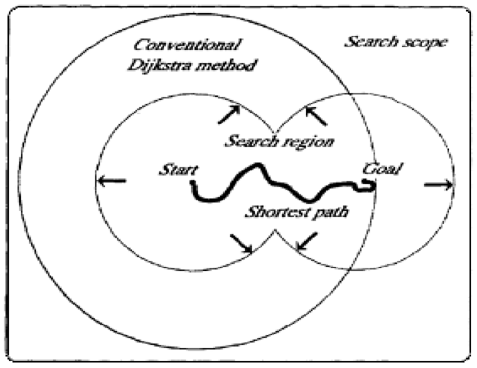


Figure 18: Comparison of Dijkstra method and extended Dijkstra method [27]

The extended Dijkstra's begins by marking the starting point and destination. After that it will calculate the movement cost from the starting point to every connecting vertex and from the destination point to every connecting vertex as well. The vertices with smallest or minimum cost value will be marked. Afterwards mark the vertices with smallest movement cost that are connected to the visited vertices to either starting point or destination. The last step will be performed repeatedly while waiting for the starting point and destination search to overlap.

Experiments show that the proposed solutions have confirmed its effectiveness by means of simulation [27]. Moreover, a near-optimum solution can be obtained rapidly after they positively decreased the search time. To conclude, extended Dijkstra can thus be applied to other path search problems that demands real time solution.

Another improvement of Dijkstra's algorithm was presented to improve the efficiency of road network route planning in finding optimal routes [52]. Through the analysis of strengths and weakness of Dijkstra they found that the main drawbacks of this method are storage structure and the searching area. They made an improvement of storage structure by compressing the data storage space and by storing the data in an adjacency list. For the improvement of searching area, they increased the road network scale and distance of neighbor vertices.

To demonstrate the performance of the proposed solution Figure 19 shows the advantage and disadvantage of these two algorithms with 3600 vertices road network.

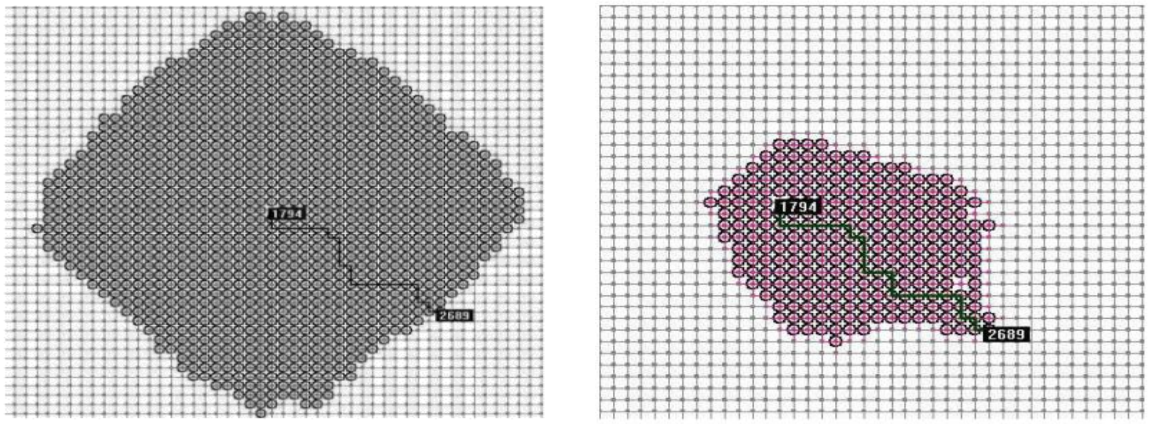


Figure 19: Searching area schematic diagram of classical Dijkstra and improved Dijkstra [50]

As a result, the improved algorithm reduced the scale of algorithm searching, minimized the time complexity and improved the efficiency of the algorithm. Hence, for this reason this study showed that the improvement of Dijkstra's algorithm was reasonable and effective.

In China there are multiple large high-rise building and underground structures. Thus fire evacuation has become a huge challenge in China and other countries where the concentration of people is high. There is some evidence that fire evacuation cannot meet the people's desired need for security, therefore intelligent fire evacuation plans has been the key focus of the research. Dijkstra's algorithm was combined with an intelligent fire evacuation system [53].

The Dijkstra's method was used to the fire evacuation plan in a situation shown Figure 20. The position of vertex 1 is the exit where numbers are the weights, varying based on the locations condition, where a larger weight indicates a poor route.

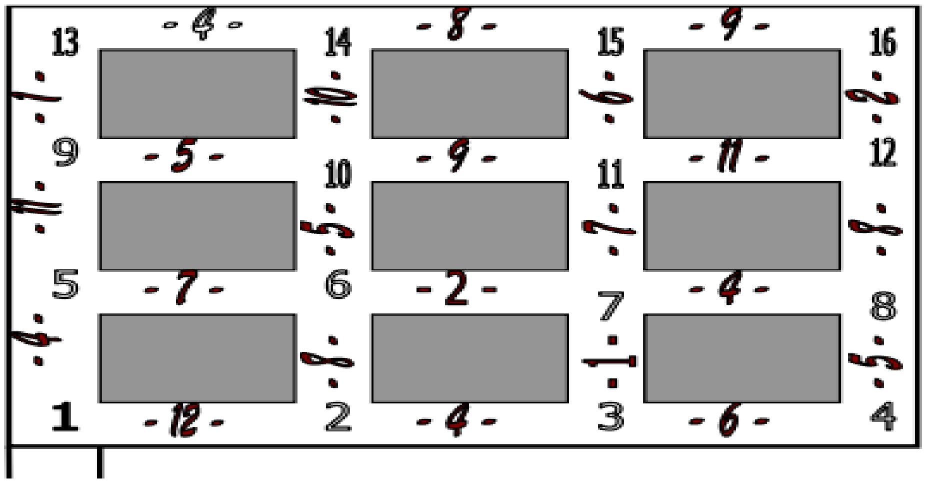


Figure 20: Simulation plan of a mall using Dijkstra method [53]

Smoke, gas, temperature and other sensory units were installed at the fire scene sending signals with the scenes information to the intelligent fire evacuation system. The data is analyzed and processed to represent the new status of the environment. Figure 21 shows the environment with the escaping route map with safe direction arrows based on Dijkstra's algorithm.

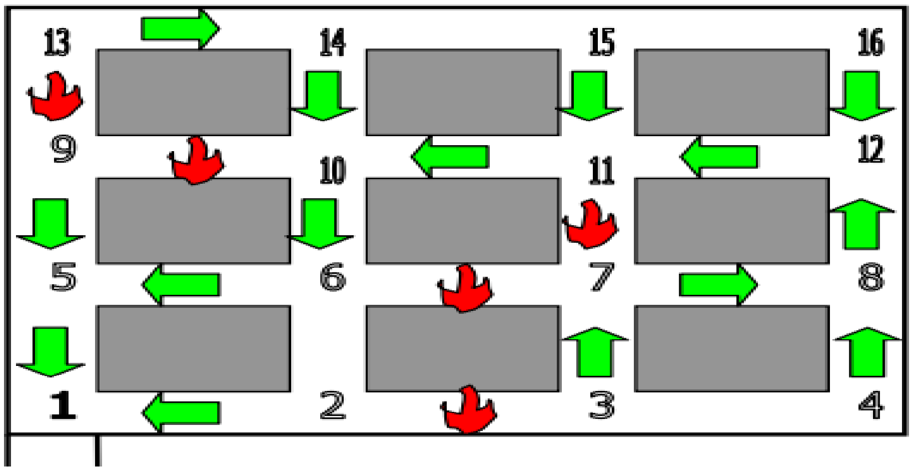


Figure 21: Escaping route of the fire scene [53]

3.3.2 Genetic Algorithm (GA) for Shortest Path

As mentioned in Section 3.2 shortest path can be also used for network utilization. Routing of data packets has a strong influence to network utilization. It is an important engineering task on the Internet which consists of finding a path from a source to destination host following a protocol. Open shortest path first (OSPF) is one of the most generally used intra-domain Internet routing protocol (IRP).

A GA solution was proposed to optimize OSPF weights with the intentions of minimizing the network congestion [54]. The traffic flow in networking is routed along shortest paths with outgoing links towards the destination. The path length is the total number of weights of the links in the path. This weight assignment problem will be solved by using the proposed GA solution.

To test the ability of their GA solution, a network topology with 20 vertices and 62 links was created as shown in Figure 22. Two vertices were assigned as source and destination then the objective was to find the lower cost between these two vertices (0, 19). The experiment results showed that GA found the shortest path in a limited time. Moreover the solution was confirmed to be used ideally with OSPF routing.

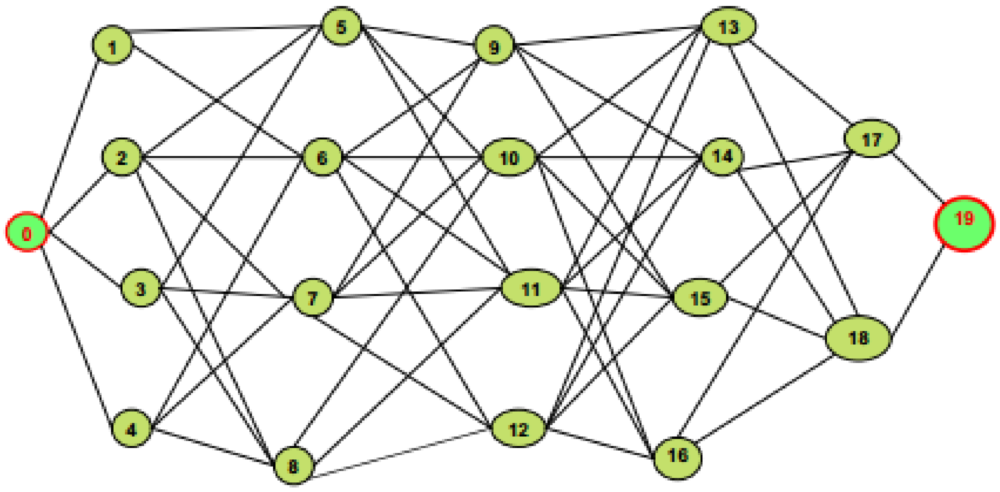


Figure 22: Example of network topology [54]

Another GA approach was introduced in solving the shortest path routing problem [55]. In this solution the basic GA operator such as crossover and mutation was applied. The crossover exchanged partial routes at independent crossing locations while mutation retains the genetic diversity of the population.

To avoid having infeasible solutions a repair function as shown in Figure 23 was proposed together with crossover and mutation to increase the quality of solution and enhance the rate of convergence.

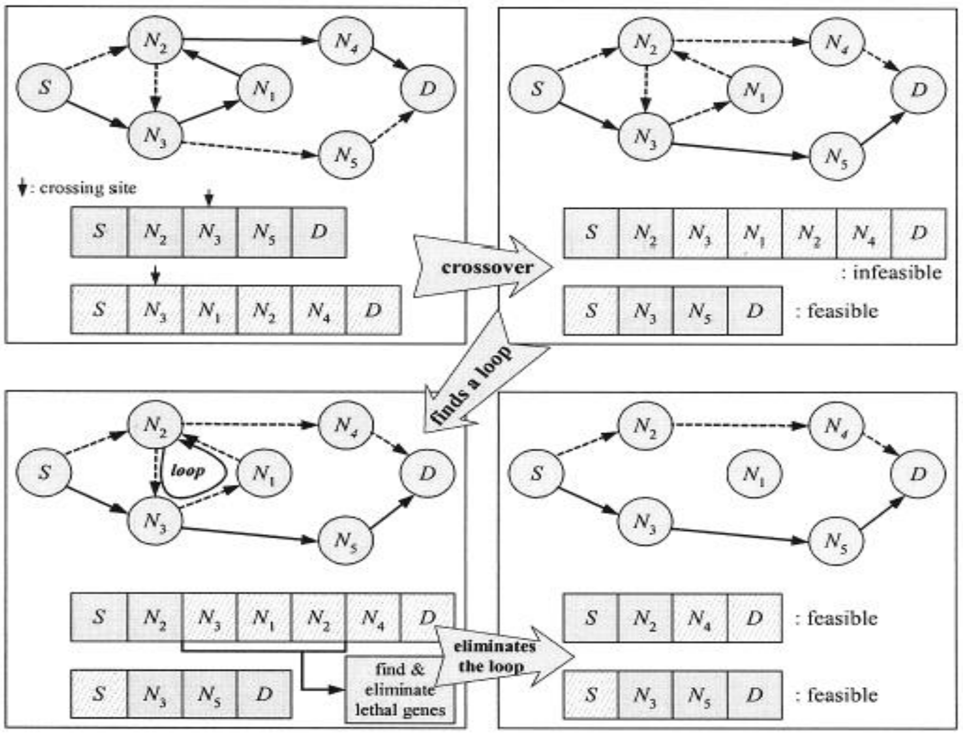


Figure 23: Overall procedure of the repair function [55]

The proposed repair function described in Figure 23 was able to find and eliminate loops in a routing path. After performing crossover one of the routes became infeasible because it has a loop N_2 -› N_3 -› N_1 -› N_2. The repair function detects and removes the loop that violates the constraint condition. The study shows that the quality of solution was better than other GA variations and the real computation time of the proposed GA was shorter than Dijkstra's algorithm.

3.3.3 Ant Colony Optimization (ACO) for Shortest Path

ACO metaheuristic have been shown to be effective in solving NP-hard problems, mostly by generating optimal solution in the shortest amount of time. A research study explored the ACO's capacity to evaluate length and cost in solving a unique shortest path problem [56]. Nevertheless their objective was not to search for all possible solutions but to look for good solution with low computation time. The proposed solution was based on AS metaheuristic called the shortest path ant colony optimization (SPACO). Each of the metrics that were used in SPACO involves cost, visibility and pheromone.

A simple transformation was applied to cost function $Desirability = 1/2^{cost}$. The equation alters the cost function into a desirability scale of range 0.5 to 1.15. This states that the edges with a higher cost will have a lower desirability to be selected while the edges with low cost will have higher desirability.

The visibility metric has an effect on each ant moving towards the target vertex. The visibility problem was based upon the distance between the current position and the length to target vertex as shown in Figure 24.

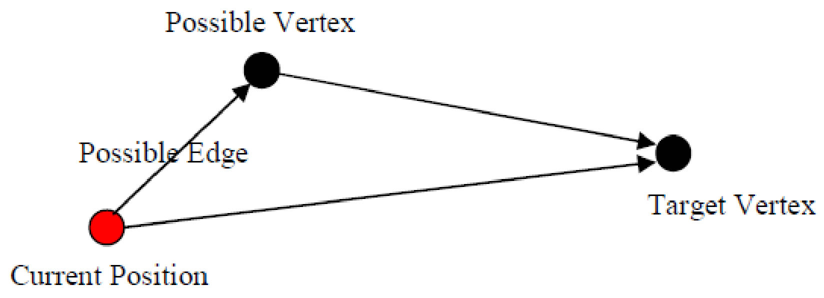
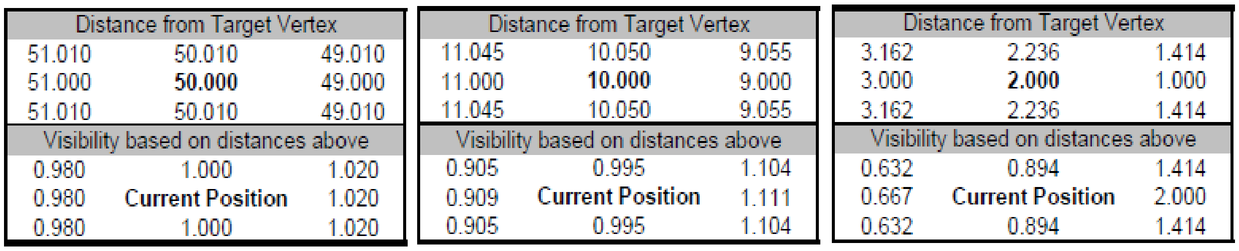


Figure 24: SPACO visibility definition [56]

The equation given by $Visibility = \frac{Distance\ from\ current\ position\ to\ target\ vertex}{Distance\ from\ possible\ vertex\ to\ target\ vertex}$ verifies the values that the visibility can carry. From the three simple cases shown in Figure 25 below it was certain how the visibility heuristic can give a stronger effect according to the distance from the target vertex. As illustrated in the same Figure the target vertex situated to the right while the calculated distance below was convert into equivalent visibility values.



Distance from Target Vertex		
51.010	50.010	49.010
51.000	**50.000**	49.000
51.010	50.010	49.010
Visibility based on distances above		
0.980	1.000	1.020
0.980	**Current Position**	1.020
0.980	1.000	1.020

Distance from Target Vertex		
11.045	10.050	9.055
11.000	**10.000**	9.000
11.045	10.050	9.055
Visibility based on distances above		
0.905	0.995	1.104
0.909	**Current Position**	1.111
0.905	0.995	1.104

Distance from Target Vertex		
3.162	2.236	1.414
3.000	**2.000**	1.000
3.162	2.236	1.414
Visibility based on distances above		
0.632	0.894	1.414
0.667	**Current Position**	2.000
0.632	0.894	1.414

Figure 25: Visibility calculations [56]

The pheromone metrics was the defining property of ACO techniques and also an endeavor in solving shortest path problem. The quantity of pheromone within the system will be distinguished from the initial pheromone density and the total amount of edges. The pheromone decay and pheromone update were constructed to manipulate the entire system pheromone constant.

In every iteration, the edges with higher pheromone concentration will lose more pheromone than the edges with lower concentrations by means of decaying every edges pheromone into a set percentage. As given by $Pheromone = (1 - Decay\ constant) * Pheromone$**.**

The pheromone update presented how ants deposits pheromone at a preset rate while looking for the problem solution. Consequently ants will deposit the preset amount of pheromone to the edge it crossed. In the same iteration, when two or more ants crossed an edge both ants will be given a relative amount of updates. This condition was defined from the equation if the edge has been traversed then $Pheromone = Pheromone + Update$. To reach the total system pheromone stability condition, the following equation must meet $total\ system\ pheromone = \frac{Number\ of\ ants * Update\ constant}{Decay\ constant}$

The SPACO was tested on smaller problems as flat and mound terrains to confirm the performance and to justify each metric used in this algorithm. Figure 26 displays the best path produced in solving a flat terrain with 9 pixels (3x3 grid) that consists of 235 possible paths.

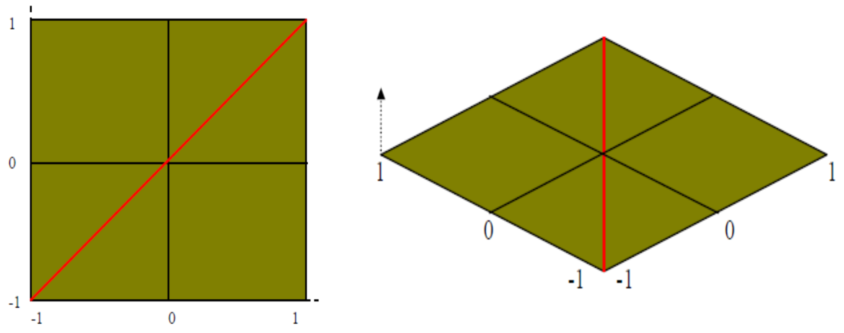


Figure 26: Flat terrain [56]

In mound terrain 25 pixels (5x5 grid), 81 pixels (9x9 grid) and 289 pixels (17x17 grid) have been tested. The valley terrain with 289 pixels in Figure 27 was the most difficult problem in this study that verifies the performance of SPACO while considering the cost and visibility.

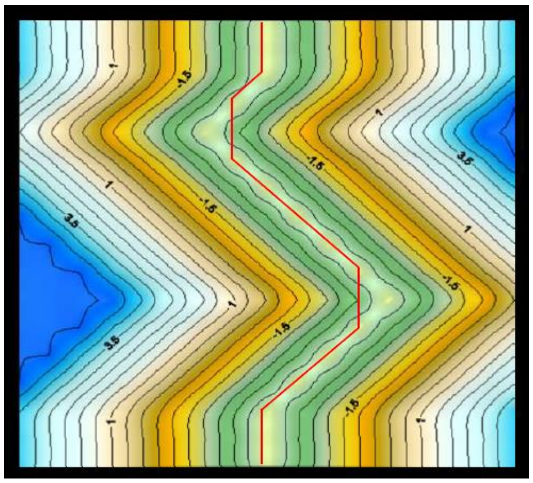


Figure 27: Valley terrain [56]

The smaller problems carried out in this study, SPACO have reached the optimum solution 100% of a time while in larger problems it found the optimal solution 90% of a time. For some cases of not finding the optimal solution, a near optimal solution was found instead as illustrated in Figure 27.

The possibilities and usability of ACO for solving the Shortest Path problem were presented [57]. The proposed solution was called ShortestPathACO algorithm which is based on metaheuristic of ant colony. There are two methods introduced in this study for finding paths which includes finding path for each ant one by one and finding paths according to the time list.

In finding paths for each ant one by one method the choice of succeeding vertex was made iteratively for each ant and another. Once an ant reached the target vertex the pheromone trail in that path will be strengthen. Therefore other ants will likely to choose the same path. In the second method, finding paths according to the time list they used this time list for recording durations in which an ant reaches a given vertex. As soon as an ant arrived faster in the target vertex it will be able to lay more pheromones on the same path. In that manner, it will be possible to strengthen the pheromone trail in order to keep ants on following iterations to choose this path. The evaporation of pheromones ensues with every transition to the next ant on the time list.

After the comparison of these two presented methods it was verified that both have influenced the choice of ants in every iteration. These conclude that using the first method of searching paths will favor paths that consist of a lower number of edges since in every lower number of iterations such paths can be found. However the redundancy effect of using time list can be compensated by stronger pheromone reinforcements of short paths and better adjustment. The proposed Shortest Path ACO algorithm is an optimization algorithm and does not guarantee to find optimal solution in all possible cases. However the proposed algorithm can be improved in order to reach optimal solutions for shortest path problem.

In previous section a Dijkstra method was proposed to find a safe escape routes in a fire scene. However an ACO based solution was presented to find the safest escape routes in five distinct realistic environments which involve static environment, dynamic environment, realistic environment with and without smoke, and realistic environment with ad-hoc networks [58] .

The first experiment done in static environment ACO was used in its simplest form with a slight adjustment. The Dijkstra's algorithm was used to calculate the optimal solution. And same experiment with adjustments of hazard probabilities which means there always a safe path that exists. These conclude that ACO was able to find the near optimal solution with a very few iteration. Whereas in dynamic environments a function called every 200th was used. Thus, per 200th iteration the route will change to the exact opposite of the current optimal solution. When the evaporation rate was set to 0, ACO failed to adapt to the new optimal solution. On the other hand, when the evaporation rate was set to 0.2, ACO was able to quickly adapt to new environments and was able to interact well with dynamic environments.

In realistic environments, a ship model consists of 290 rooms and 4 corridors in two floors were used to attain empirical results from ACO. The actual fires was generated and simulated by using the well-established tool fire dynamics simulator (FDS). Figure 28 shows the visual representation of the temperature with a color-coded scale.

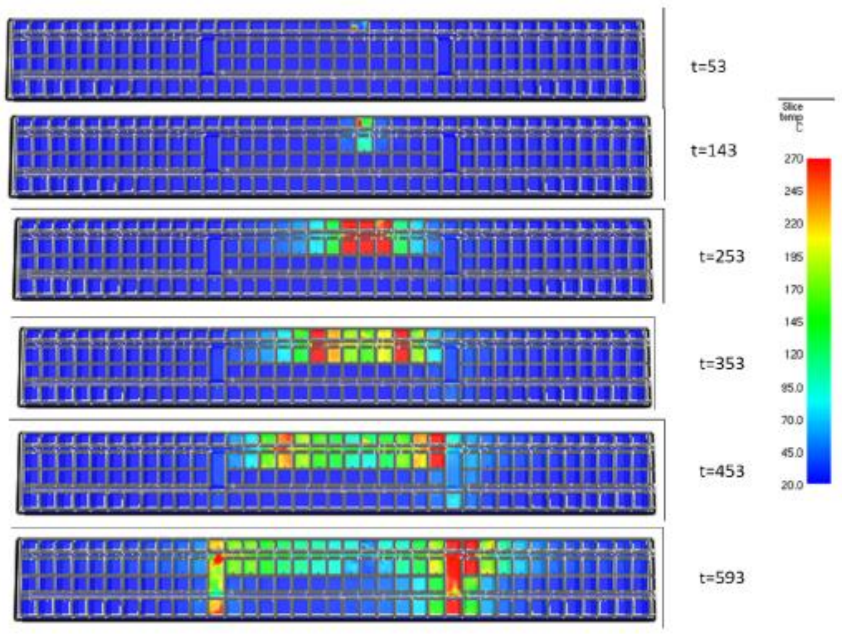


Figure 28: Heat map visualization illustrating the room temperature as the fire spreads over time [58]

In this experiment the fire spread out over a time frame of 600s. The fire reached one corridor in 150s and then both corridors at most 500s. For each simulated second, a certain number of ants were released. Three variants of ACO were used: with 10, 100 and 1000 ant for 600 different scenarios. In this experiment it shows that ACO can be able to attain safe escape routes even in difficult times however more ants are required in order to reach convergence.

Another experiment was done in realistic environment with smoke. This experiment produced very different results due to fatalities from much higher smoke. Therefore smoke had much impact on evacuation situation compared to fire. Even though having smoke affects the performance of ACO, by running enough ants to reach convergence will overcome this situation. Considerably, ACO with 1000 ants can still provide near optimal solutions.

The last experiment in realistic environment with ad-hoc networks using ACO with 100 ants managed to find a smaller value of shortest path. Even with a network of only 5 rooms, ACO was still able to find a very good solution. As a conclusion from all tested scenarios, ACO successfully produced the near optimal escape route solution in all cases.

3.4 Traveling Salesman Problem (TSP)

TSP is an NP-hard problem in combinatorial optimization (CO) problem that is commonly used in operations research since it provides a well-known application to evaluate different algorithms. TSP consists of a salesman and a set of cities. In the regular form of TSP, a map of cities is given to the salesman. The salesman has to visit all the cities starting from the initial city and returning to the same city to complete a tour. The challenge for the salesman is to minimize the total length of a completed tour. The length of the tour should be the shortest among all possible tours in the given map. This challenge also means finding a cycle passing through all the cities in the map while having a minimum total weight, also known as Hamiltonian cycle. In a TSP framework, the Hamiltonian cycle are usually called tours. For instance, the Figure 29 [59] below had given a graph with weights. The optimal solution in this Figure will be path {A, B, C, D, E, A} which has a total length of 24.

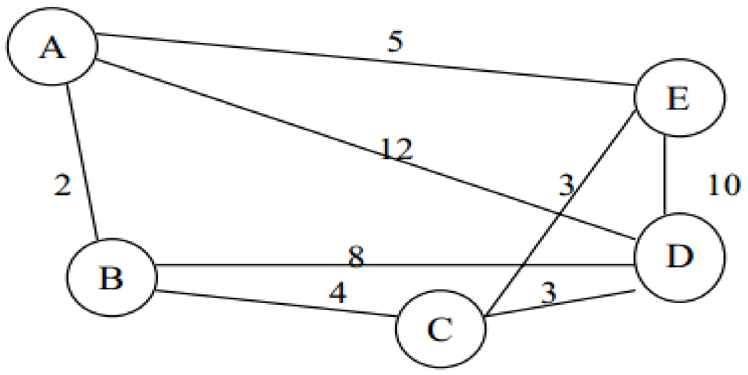


Figure 29: Example of the cyclic TSP problem [59]

The two basic types of TSP are called symmetric and asymmetric TSP. In symmetric TSP, the distance between each pair of cities is independent of the route in which the salesman travels. For instance in Figure 29 above, vertex A to B and vice versa will have the equal distance of 2 in symmetric TSP. While in asymmetric TSP, distances between two cities might be different and paths may not exist in both directions. This study will focus on symmetric TSP.

Many different well-known combinatorial algorithms applicable to many areas were first developed to solve TSP. Therefore TSP has been often served as a touchstone for new problem-solving strategies and algorithms. In the following, two different algorithms include GA and ACO will be discussed to show how these two algorithms improved the different parts of TSP solving process.

3.4.1 Genetic Algorithm (GA) for TSP

GA has been studied to solve many search and optimization problems, specifically NP-hard problems like TSP. In GA an individual is known as a candidate solution for a given problem. Each solution will be evaluated through a fitness value. The fitness value defines how close the candidate solution is to the optimal solution. Then set of solutions creates a population that evolves from generation to the next by using genetic operators such as selection, cross over and mutation. As stated by Darwin's survival of the fittest approach, we retain the best candidate solution from generation to generation while allowing the inferior ones to die off. The same process repeats until the best candidate solution in the population will fulfill the fitness criterion.

GA seeks the large and complex space of solutions towards the best answer. The effectiveness of genetic operators can increase the robustness of search process and has the ability to avoid being trapped in local minima. In 1991, a GA approach was presented to solve TSP with up to 442 cities [60]. The proposed solution insular genetic algorithm (IGA) was an improvement of the underlying basic operators of GA.

The order crossover operator consists of a part of parent1 and the other part of parent2. This approach was used to reach a tougher merging of the genes of the parents, which is suitable particularly for large problems. A well-known 2-opt local search measures up all possible valid combination and makes two changes to single route. While or-opt heuristic changes a single route by re-arranging more than two stops to reduce the total distance. These two heuristic approaches are used together with the neighboring restriction to provide a good local optimization operator for solving large problems.

In selection of survival of the fittest when greatest individual are chosen too often it will decrease the diversity of the population. For that reason the best individuals 4 times more often than the worst was chosen.

In the experimental investigation indicates that population were degenerated before the optimal solution was found specifically in large problems. As a result, the island model was developed to optimize the isolated population until they degenerate. The degeneration was eliminated by refreshing the population on every island through individuals of other island. Thus, the evolution can proceed. This study showed the performance of IGA in solving TSP with up to 229 cities in less than 3 minutes average runtime and TSP with up to 442 cities ideally in an acceptable time limit.

A sample application using GA demonstrate a solution for TSP [61]. In this solution the crossover also known greedy crossover which is not completely random was used. In Figure 30 represents a GA solution for TSP with 40 cities and 80 population size. In this solution the maximum number of cities was only up to 50. Apparently this solution cannot solve large problems.

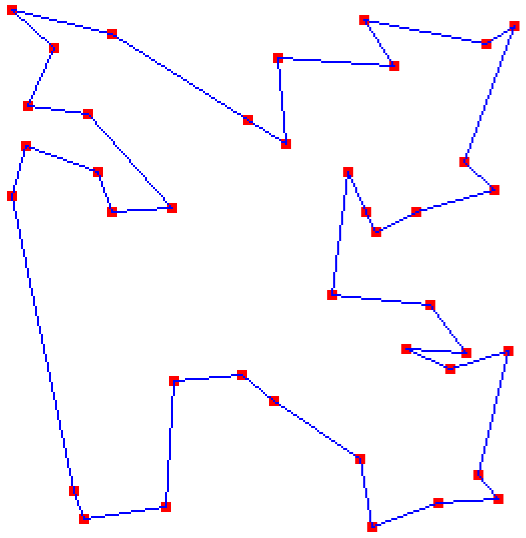


Figure 30: Example of TSP solution using genetic algorithm [61]

Some of traveling paths in the means of evolution process are shown in Figure 31 [62]. The population size = 40, number of cities = 40, rate of crossover = 0.6 and the mutation rate = 0.1. These paths were generated from the best individual in corresponding generation. The zig-sag paths will only fade out after a few generations which validate the effectiveness of GA. Nevertheless, in the greater part of trials, the algorithm can only obtain the local minima paths in the designated number of generations.

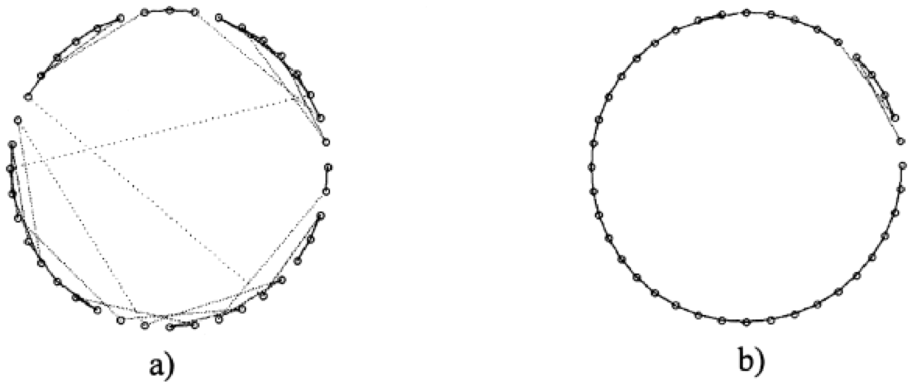


Figure 31: TSP using GA approach (a) after 3rd generation (b) after 18th generation [62]

The edge assembly crossover (EAX) performances for TSP were analyzed and examined [63]. EAX is a recombination operator that uses edges from two parents to build disjoint sub tours. One of the important traits of EAX is that it can present new edges into the offspring when connecting sub tours. Mostly edges in both parents and entire population were introduced into offspring. Since EAX become aware of the appropriate tradeoff between the ratios of edges inherited to offsprings from parents. This study showed that EAX can be used to generate large numbers of better offspring's from two parents.

3.4.2 Ant Colony Optimization (ACO) for TSP

The Ant System (AS) was the first ACO algorithm which proposed as a means of solving the TSP [64]. The goal is to find the shortest path to link a series of cities using artificial ants. Each ant will make possible round-trips along the cities while moving from one city to another. Some rules has to be considered such as ants must visit each city exactly once, a distant city will have less chance of being chosen due to visibility, and the more pheromone trail deposited between two cities the greater probability that it will be chosen. In addition when the path is short, ants will deposits more pheromones in the way it traversed and pheromones will evaporate in every iteration. Figure 32 below [65] describes how the AS sends out artificial ant agents to explore the possible routes on the map.

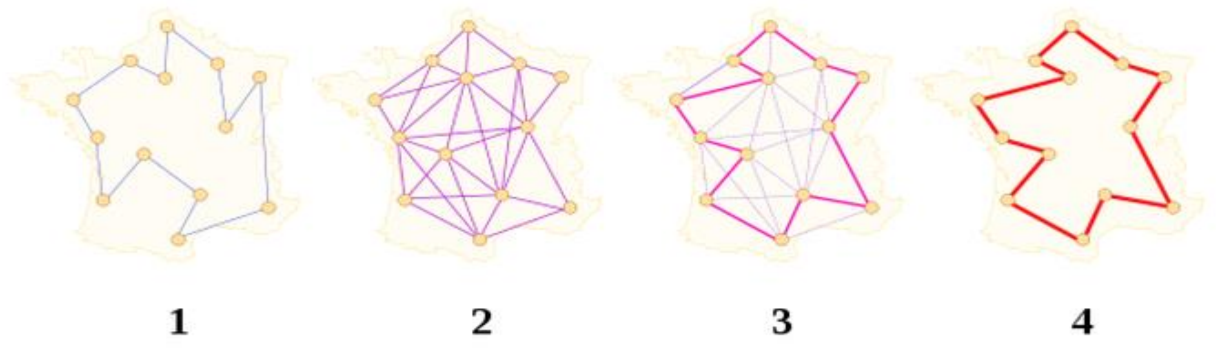


Figure 32: Trail pheromones [65]

The ants will explore the map while depositing pheromones between cities they have crossed until they completed the journey. Those ants that completed the shortest journey deposits more virtual pheromones. The amount of pheromones is based on the journey length: the shorter the journey, the more virtual pheromones it deposits.

There is still active research on improving the performance of ACO algorithms. Though there are many algorithms that solved TSP effectively, ACO algorithm costs too much in time to produce a nearly optimal solution which became a barrier. In a research study they presented a fast ACO algorithm for solving TSP [66]. According to this study the two basic ways for ACO improvement method are: 1) improvement of element information update mechanism for better optimization ability of the ant colony algorithm and 2) search mechanism improvement.

A pheromone increment model called ant constant for keeping the energy conversation of ants was proposed to represent the pheromone difference of different paths. At the same time a pheromone diffusion model based on info fountain of a path was established to strengthen the cooperation among ants and to reflect the strength field of pheromone diffusion. The experimental results in this study displayed that the proposed method was successful in improving the searching ability of ACO and can obtain better search capability in combining with existing ACO variations.

The same study was done and a new mechanism of pheromone increment and diffusion for TSP were presented [66]. A new ant-constant model was proposed to increase the purpose of ant in choosing path. Figure 33 illustrates the new diffusion model based on a path which was introduced to strengthen the communication among ants.

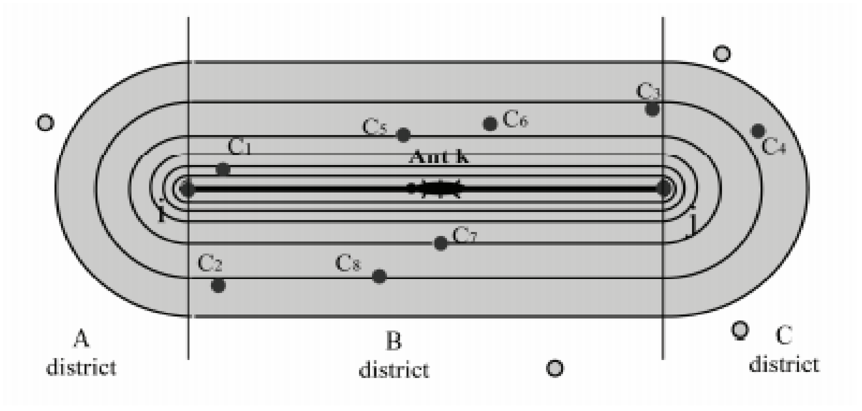


Figure 33: The info fountain of a path [66]

The ant traversing the Figure 33 above will go from city to depositing pheromones forming a path intensity field with much larger scale to influence the sets of the cities$[C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8]$. The new diffusion model reflects the intensity fields and increases the action scale of the pheromone diffusion. Another proposed solution in this study was mutation strategy with lower computational complexity. The reason for using this mutation strategy was to improve the solutions obtained by artificial ants. Moreover by using mutation operator will lead to much smaller computation complexity and higher search efficiency.

The population declining ant colony optimization (PDACO) algorithm is an improved Ant system algorithm which was proposed to solve TSP [67]. The PDACO algorithm was introduced for global searching capability improvement without increasing the computational complexity.

Many ACO algorithms can be combined with PDACO to improve its performance. For instance, a combination of PDACO with MAX-MIN ant system (MMAS) and ant colony system (ACS) for solving TSP was presented [68]. Though MMAS and ACS can overcome the shortcoming of AS, the searching range still becomes smaller with large population of ant colony. In PDACO algorithm the population of the ants is much bigger in the beginning. In this manner it can enlarge the searching range and as iterations carried on the population declines which can reduce computational complexity.

In Figure 34 shows the two presented solutions applied to TSP: PDMMAS by combining PDACO with MMAS and PDACS by combining PDACO and ACS. According to the experimental results, it showed that PDMMAS and PDACS have much better performance than the corresponding ACO algorithms and obtained the same computation complexity as PDACO algorithm.

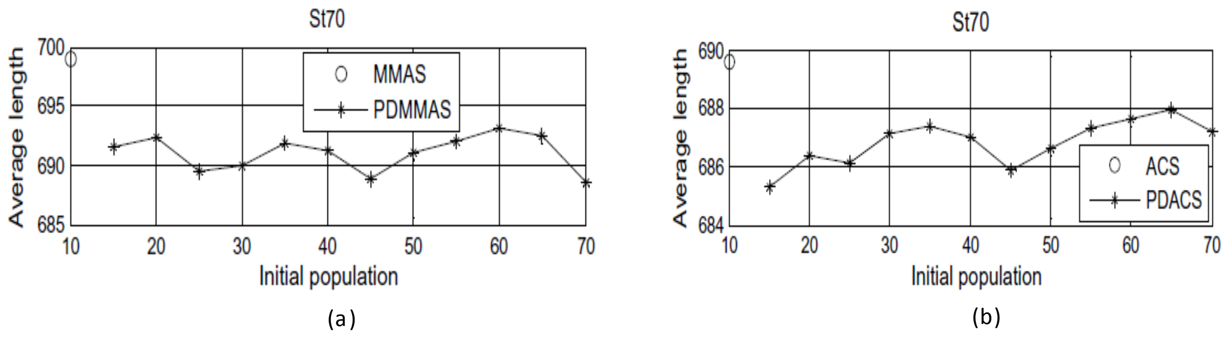


Figure 34: (a) Average length vs. initial population of PDMMAS in St70 (b) Average length vs. initial population of PDACS in St70 [68]

The pheromone diffusion model of PDACO algorithm which takes a city as info fountain is shown in Figure 35. The PDACO algorithm sets up a pheromone diffusion model to enhance the relations among neighboring ants. The main intention was the pheromone union effects among nearby paths when ants are searching for paths. The pheromones deposited on path will immediately influence the selection behaviors of neighboring ants located on the two cities connecting the path. Furthermore pheromone intensity will handle these two cities as two info fountains to diffuse around. Once the pheromone intensity increases on nearby paths, the behavior of neighboring ants will be easily influenced.

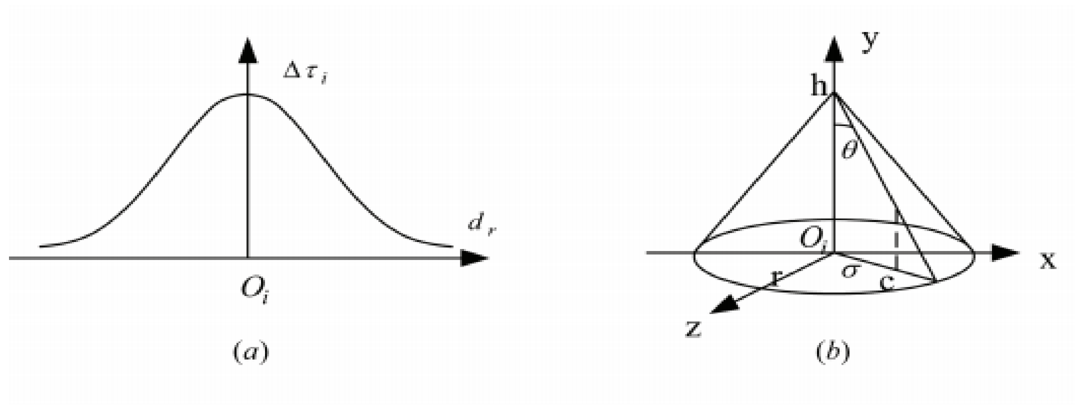


Figure 35: The pheromone diffusion (a) sketch map of the pheromone diffusion (b) reduction pheromone diffusion model [66]

As shown in Figure 35a the pheromone intensity $\Delta\tau_i$ and the distance d_r away the info fountain has more inverse ratio relationship. The reduction pheromone diffusion cone shaped model in Figure 35b helps the PDACO algorithm to accumulate pheromone influences on neighboring cities results from the info fountainO_i.

If the city C borders on the info fountain $O_{i,}$ ants on city C are able to experience pheromone intensity. The height of the cone model is represents by h while θ is an angle formed by the cone plane and cone axis. The diffusion radius is r and σ is the distance between city C and info fountain O_i. This model enhances the collaboration among ants by amending the pheromone intensity. Therefore this strategy improved the effectiveness of AS.

3.5 Limitations of State of the Art

Most of the existing approaches have been described in this chapter and some successful results regarding topics in solving shortest path and TSP have been summarized. In this section we will discuss the limitation or drawbacks of the existing solutions.

In computer program correlation matrix, adjacent matrix and distance matrix are used to calculate the shortest path based on network matrix of Dijkstra’s algorithm. When the number of vertices are very large, it occupies more CPU memory. For instance if the number of vertices are 3000 for one layer it needs $4 * 3000 * 3000 = 36000000$ bytes or 36MB memory, just as if the number of vertices are 6000 it will then need 144MB memory. Therefore it is very difficult to apply the Dijkstra’s algorithm in analysis for huge data. Moreover, the fact that it does blind search can consume a lot of time waste of essential resources.

GA can deal with combinatorial problems but it has a primary drawback which results from their flexibility. The design of GA requires an encoding schemes and one has to make sure that the evaluation function assigns a meaningful fitness values to the GA. It is often unclear how the evaluation function can be formulated for the GA to generate an optimal solution. Since it is computationally intensive consequently convergence can be a problem for GA. What may seem like simple changes in the algorithm will often leads to unexpected types of emergent behavior. Therefore they aim to increase the understandability of GA and to apply more advanced analytical approach.

Most of recent researches presented in this chapter were on the development of new and better performing variants of ACO. These algorithms are more effective in global search but still have a big chance to get stuck on local minima. As a result, attempts have been made to alleviate the movement of the ants through various models. ACO has been combined with mutation operator to enrich the diversity of solutions and to enhance the ability to get a nearly optimal solution. However, ACO algorithm costs too much in time to produce a nearly optimal solution. Though some existing ACO variants can overcome the shortcoming of ACO, the searching range still becomes smaller or chaotic with large population of ant colony.

Chapter 4 – Proposed Solution

The choice of ACO is motivated by the fact that there are so many existing variations of ACO algorithms. And the research on how to improve the performance of ACO algorithms is still an active topic.

For the shortest path system we had to make sure that the ants first will work as expected from a classical ACO solution before we could move on to multi-leveling. Then we expect that the ants favor the single level over the multilevel as they are to be built towards the single level and then applied to the multilevel.

It must be noted that in the results one will see the graphs make a jump at each level change for the TSP but not in the shortest path. It happens on both solutions but it can be calculated and removed in the shortest path due to the down resolution as each level is $1/_4$ the size of its parent level. Because distances are Euclidian this can be calculated by being 4 times larger than its parent.

On the TSP there had to be implemented a different solution as it its cost on the edges are determined by a number of different elements for an example distance. This shows that the same approach cannot be used for the shortest path and the TSP. For solving this we chose to cluster vertices in pairs then solve the new problem containing only half as many cluster-vertices and imprint the solution found onto the original problem. This has an impact on the data in the way that the graphs will look like they are worsening but that are only the performance on each individual level and different simulations must be ran and evaluated against the same level.

4.1 ACO

The ants themselves are simple and will be acting fairly similar in both the shortest path and the TSP problem which can be further investigated in Appendix A. Conceptually the ant from classical ACO approaches will outline an ant that has a certain amount of pheromones and disposes them evenly over the path between nest and food. The ant is also obliged to have a weighted random selection of the next movement where the pheromones of the previous ants acts as the weight, thus over time leading to a local optimal solution.

As the way an ACO algorithm is built is not a strict model but more of theoretical function model. It allows for the addition of modifications or additions to the algorithm as we used the common additions of random movement at random times to make sure that the ant has the possibility to leave the pheromone trail and discover or improve by modification to the trail. The occurrence odds are defined in 4.2 Shortest path and 4.3 Traveling Salesman Problem (TSP).

The world in which the ants are simulated also has a trick, for the shortest path the global amount of pheromones are decreased by a factor determined by the tests later in the chapter. While the TSP solution have the pheromones placed as lists instead of as a depot, removing a randomized pheromone variable at a randomized global vertex where the odds of removal is described later in the section 4.3 Traveling Salesman Problem (TSP).

4.2 Shortest path

Our contribution to the shortest path problem is to determine the advantage of a multilevel ACO solution over the classical ACO. The multilevel solution of the map is done in a multi grid approach as detailed in section 3.2 Multilevel Algorithms. The map selected for this problem was a simple empty room map where the ants spawn point is the upper left floor pixel and food is the lower right floor pixel, allowing for a simple way to calculate the shortest path. This is the straight line between the spawn and food. For the shortest path we decided to place the pheromone as a value on each pixel and for re-traversal of the pixels by the ants to make sure they do not get stuck in their own "locked" path, for the total path a traversed vertex is only counted once.

4.2.1 Shortest Path ACO – Exploration Factor

When tuning the ants for the problem we looked at what variables we had in hand and notice that both the exploration factor together with the pheromone dissipation were two good candidates for tuning. The first value to tune was the ant exploration factor on a set of variables denoted as $n = \left[0.1, 0.01,\ 0.001,\ 0.0001,\ \frac{0.1}{vc},\ \frac{10^6}{vc}\right]$ $vc = vertex\ count$ to gain an indication of how the freedom impacts the solution as shown in Figure 36.

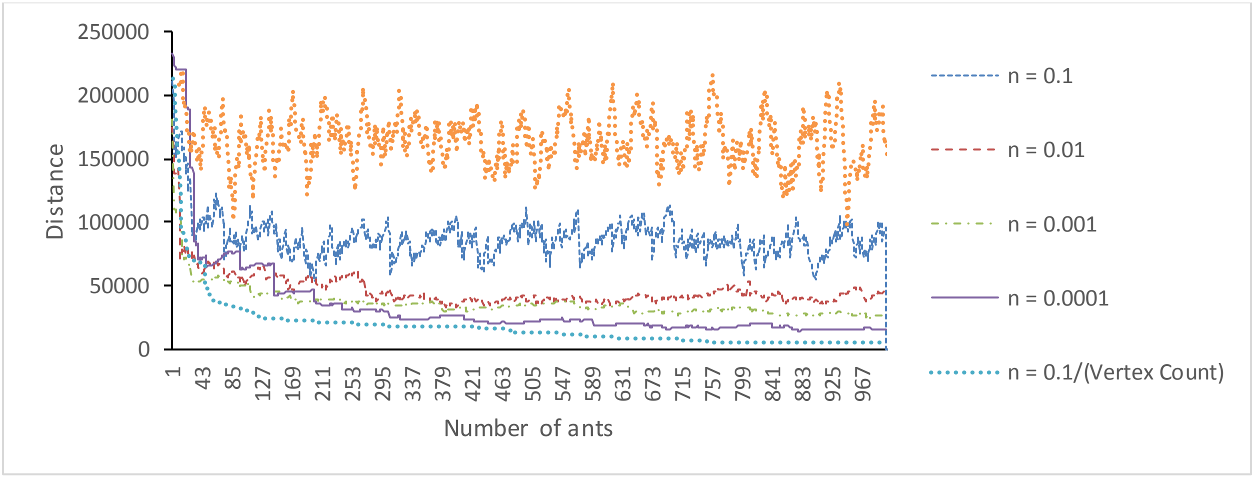


Figure 36: ACO exploration variable

Figure 37 shows that the lower the value the better it performs. Nevertheless it also impacts the discovery of new solutions therefore it cannot be set too low as it would decrease the usefulness of running the solution over time. And as we see that the multilevel approach does outperform or perform evenly with the single level in all simulations.

On one note, it is valuable to see that the variable of $\frac{1M}{n}$ performs almost without improvement (on average there is a minor improvement) within the single level. But when it was applied on the multilevel it performs on par with 0.0001, as with multi-leveling one achieves the initial drop at each level, where at a classical only one of these drops.

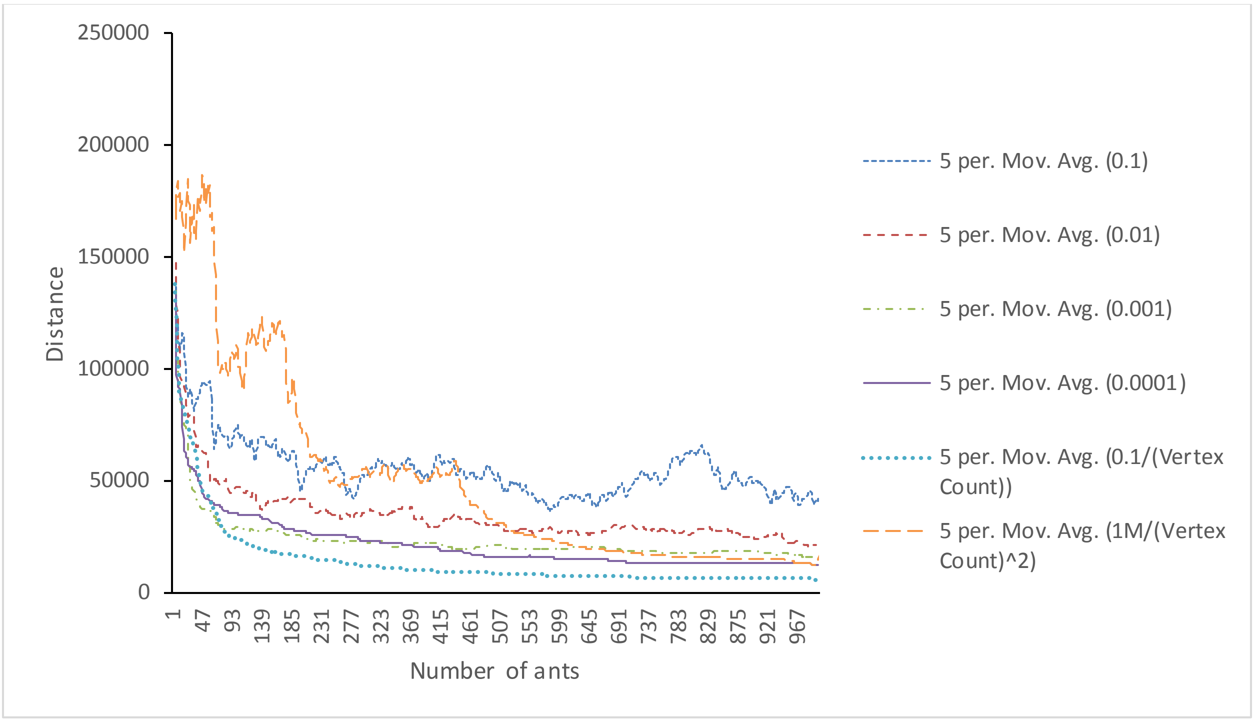


Figure 37: MG-ACO exploration variable (4 levels)

4.2.2 Shortest Path ACO - Pheromone Dissipation

For the variable of pheromone dissipation in MG-ACO it is notable that the higher the dissipation value is, the better the solution as shown in Figure 38. If we use 100% dissipation that would leave us at an initial condition (same as start) for each run with no improvement showing that we need some transfer of pheromone information, assumable only from ant$_{n-1}$ to ant$_n$ in a linked list style.

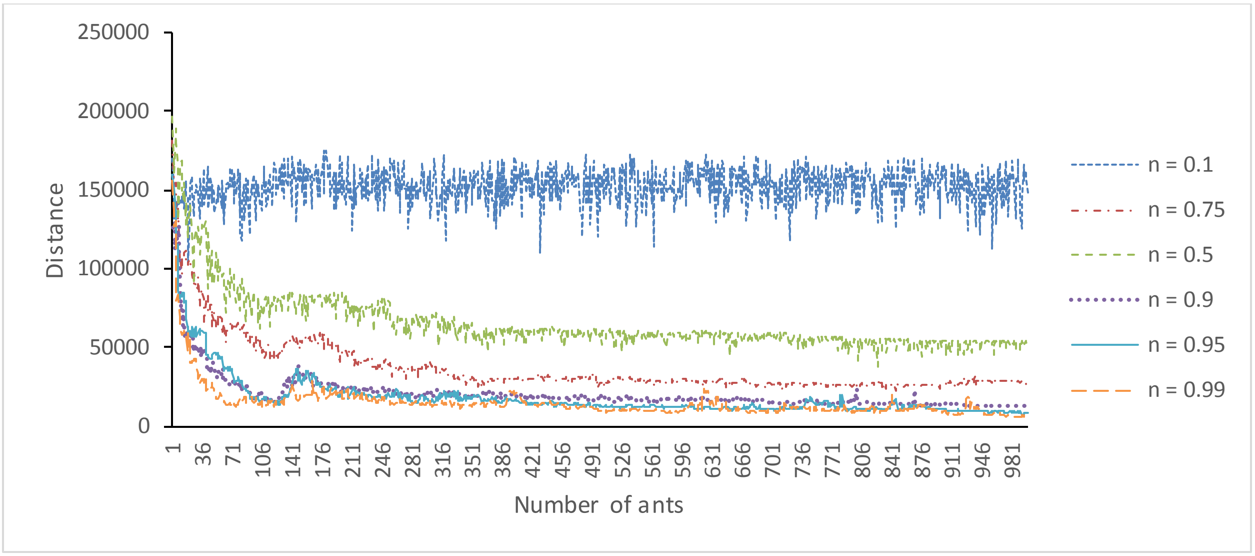


Figure 38: MG-ACO pheromone dissipation variable (4 levels)

4.3 Traveling Salesman Problem (TSP)

For the TSP we deployed a fairly classical ACO approach, where every road between vertices stores a list containing the pheromones dropped by passing ants. And the road selected by the ant is decided by performing a weighted random selection where the edge pheromone sum divided by distance will be the edge weight, or in some cases the road is picked at random dubbed exploration factor.

4.3.1 TSP ACO – Exploration Factor

Figure 39 shows different exploration factors we have tested to find out which value is appropriate in performing experiments. As 0.05 (5%) performs the best beating 0.1 by a minor margin.

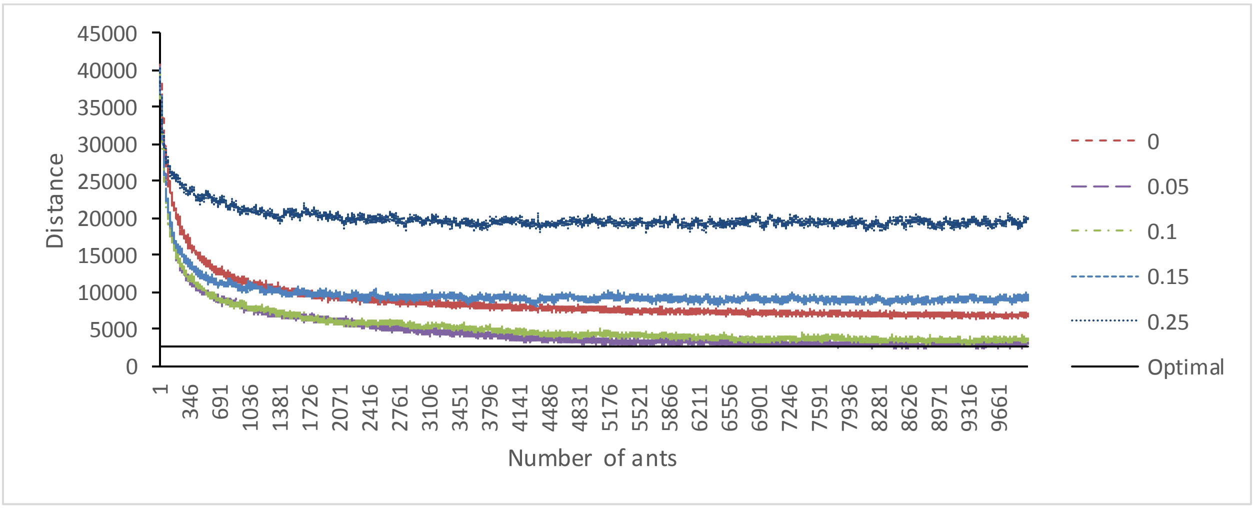


Figure 39: Exploration Factor

As the data can become somewhat overlapping, we added a moving average to the data making it easier to see the differences in performance. A little surprisingly 0% freedom outperformed 0.5% therefore we assume that this is due to the removal of vertices which creates connectionless vertices and selects edges at random.

4.3.2 TSP ACO – Pheromone Dissipation

We performed a range of tests to find the close to optimal variable for pheromone dissipation and exploration factor. In these tests we initially performed them on a simple system where every vertex is positioned on a radial allowing for a good baseline test to check against the optimal as it would be in an uninterrupted circle.

The pheromone dissipation simulations were ran with the exploration factor set to 0.005 and we see that when the pheromone dissipation goes well below 0.05 (5%) the performance of the ants improves extremely. As a side vertex the 0% performs somewhat worse than 5% and it is more expensive as the lists of pheromones become very large. The overall results of each pheromone dissipation values are shown in Figure 40.

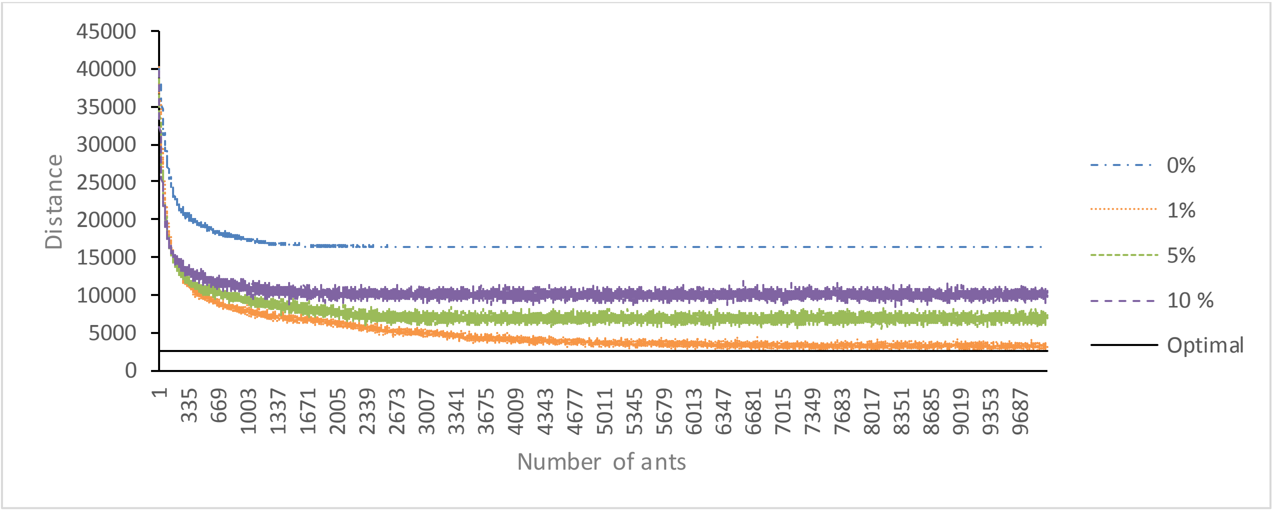


Figure 40: Pheromone dissipation distance per ant

4.3.3 TSP ACO – Cross comparison

As we now have a fairly good baseline for which variables to use for the classical approach, we also wanted to make sure that these variables performed expectedly in other solutions. Thus we performed the cross comparison on the problem described in Figure 41.

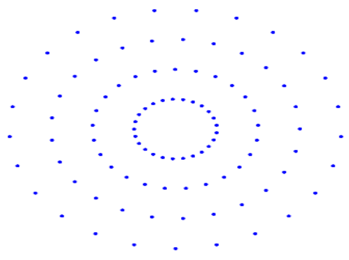


Figure 41: Vertex positioning for advanced TSP testing

We picked the best 2 solutions from both pheromone dissipation and exploration factor then combined these into 4 tests [Dissipation, Freedom] => {[0, 0], [0, 0.005], [0.05, 0], [0.05, 0.005]} as shown in Figure 42. This increased the complexion of the problem to solve and to make sure that we receive a solution capable of solving more than the simplest problems.

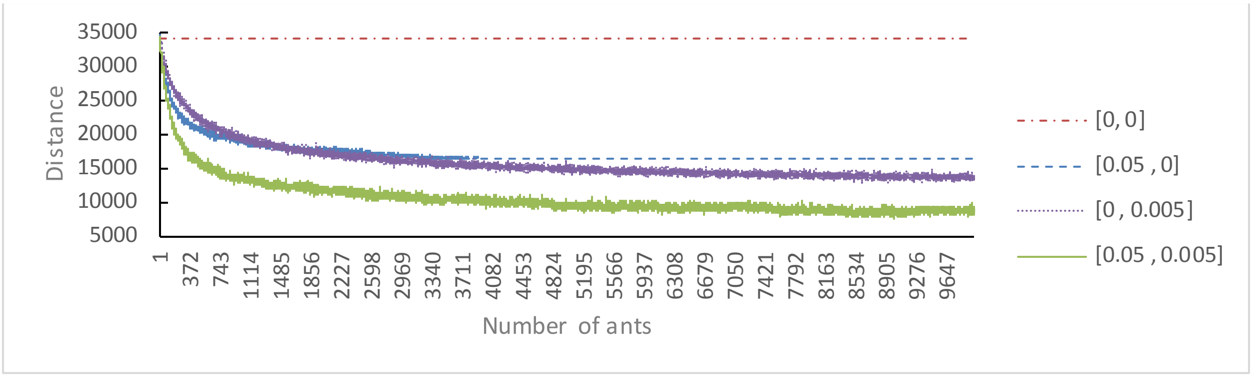


Figure 42: Cross comparison of variables

As we can see the difference between each type is fairly clear, we cannot drop both to 0 as there while be no change in the ant's movements over time. Also it shows that having a little of both is better than having nothing of one. If we only compare the freedom, we can see that 0% freedom outperformed 0.5% in the simple radial position although this does not hold up when the complexity increases. Thus we see that [0.05, 0.005] vastly outperforms the other candidates leaving us with pretty much only one good candidate for the ACO configuration.

4.3.4 TSP Multilevel ACO (ML-ACO)

Solution description

We notice that pairing would work out fairly well and was implemented based on the paper previously discussed [40] where we found that the amount of levels per set would be best suited by pairing the vertices one layer at a time until one had a level with 4 or less vertices in it as shown in Figure 43. This means that 100 vertices would become 5 levels containing [50 (pairs), 25 (pairs), 13 (12 pairs + 1 single), 7 (6 pairs + 1 single) and 4 (3 vertices + 1 single)].

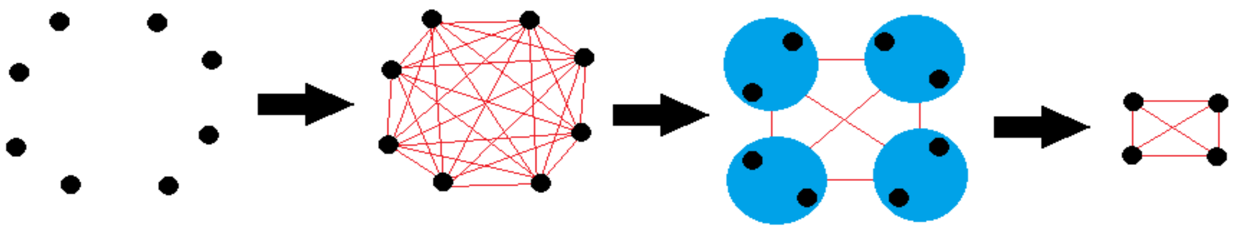


Figure 43: Leveling up from n to n+1

The single vertex is added in case of odd sets so that in theory the last set of 4 (3+1) could be grouped into 2 pairs. The vertices are paired into so called clusters where we assume that the cluster A's distance/cost to another cluster B is the shortest cost from the internal vertices of cluster A to the cheapest vertex within cluster B as seen above.

This simple technique for clustering can be performed over and over again on the end result to achieve a lowering of the problem complexity until the wanted complexity is achieved. Solving the problem leads to another problem, how do we add the solution to the level above?

We went through several solutions most of which limited the solutions in a sharp way by eliminating the edge of level n-1 based on the solution of level n. This meant that ants would need to have rules of movement defeating the purpose to use them as it could be solved with a simple rule based solution. So the solution we selected ended up on being a weighting of level n-1 based on the solution of level n as seen in Figure 44.

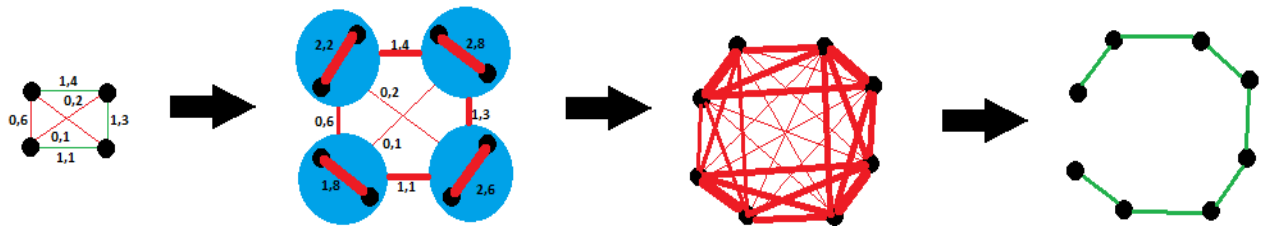


Figure 44: Propagating the solution when leveling down from level n to n-1

The solutions of the simplified levels are then propagated up to the level above and solved it again continuing that solution onto the initial problem. This will make the initial problem weighted towards specific local minima. The local minima will be refined towards global minima, which we propose to be performed quicker after having the local minima as an initial start solution for the initiated level. The code can be seen and checked in Appendix B.

For the next several experiments we performed a test to gain knowledge of how different leveling intervals performed. As we have seen that using a static interval performs fairly bad as well as using a too steep function makes for too many ants in total, as the top level would run indefinitely.

So the solution we picked was $\sum_0^{level} Multiplier * level^2$ as shown in Figure 45 where the multiplier M is used to scale the total amount of ants in a run and level n is the total amount of levels adjusted as previously described. Allowing for a more consise test enviroment and yet again it must note that only the same steps can be compared against each other.

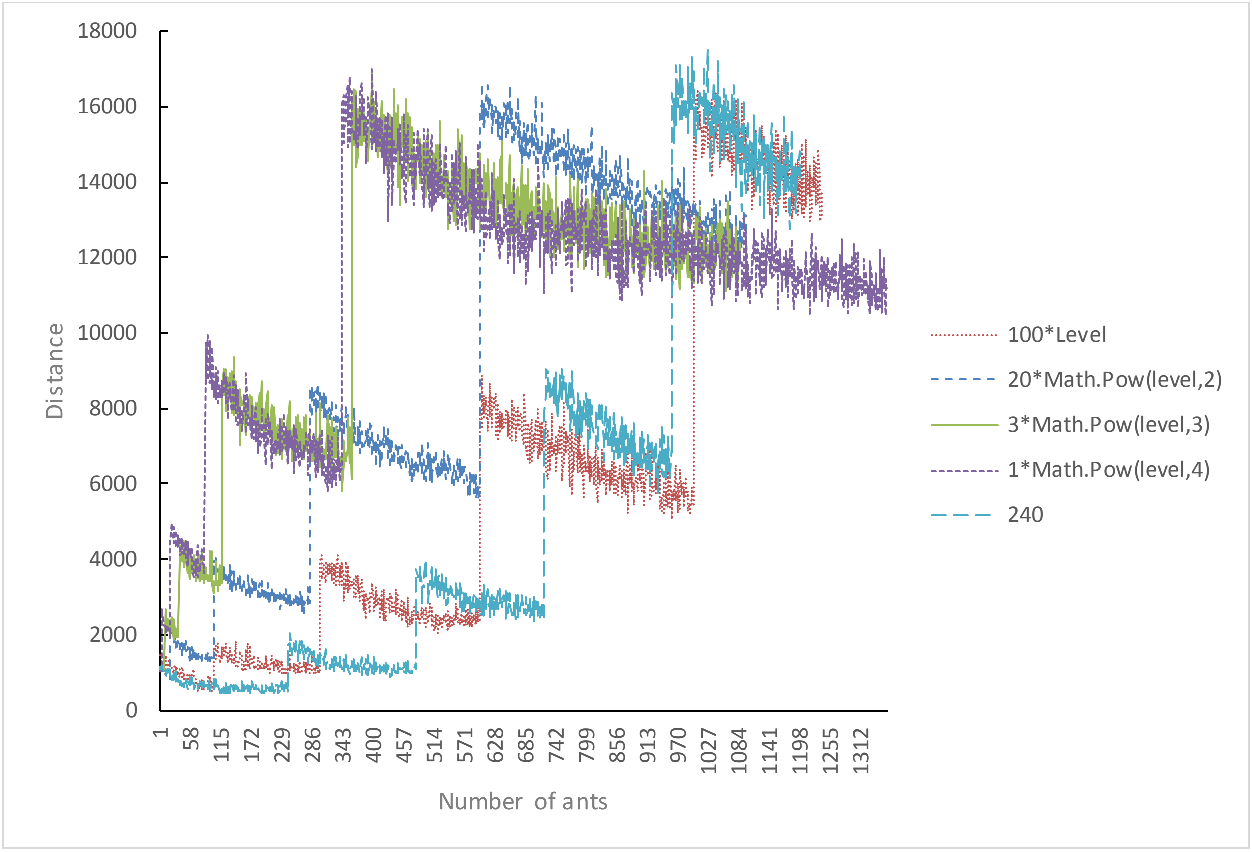


Figure 45: Comparison of different complexity increment formulas

4.3.5 Initial cluster testing

To test this idea we used a fairly small map running a test experiment on a very small scale. Initially we ran a series of small tests to attempt in achieving a concept of what was to be expected. The initial test was performed in a circular TSP map as shown in Figure 46 with 100 vertices, 4 levels and 300 ants versus a normal single level ACO.

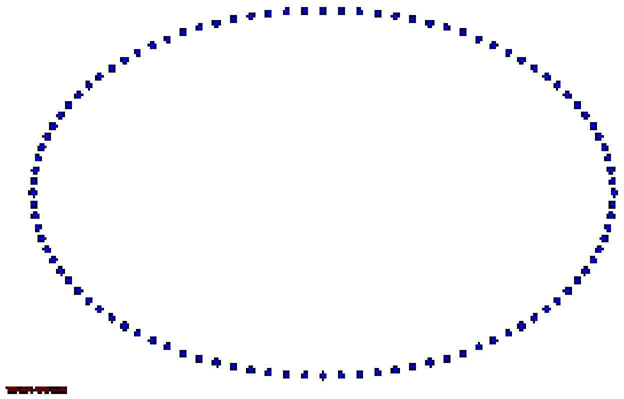


Figure 46: Vertex map in the circular experiments

The results in Figure 47 shows that the multilevel outperforms single in shear distance. It is worth noting that the steps in the multilevel is caused by the steps in which the multilevel ACO is leveled, thus only the last step is comparative to the classical single level.

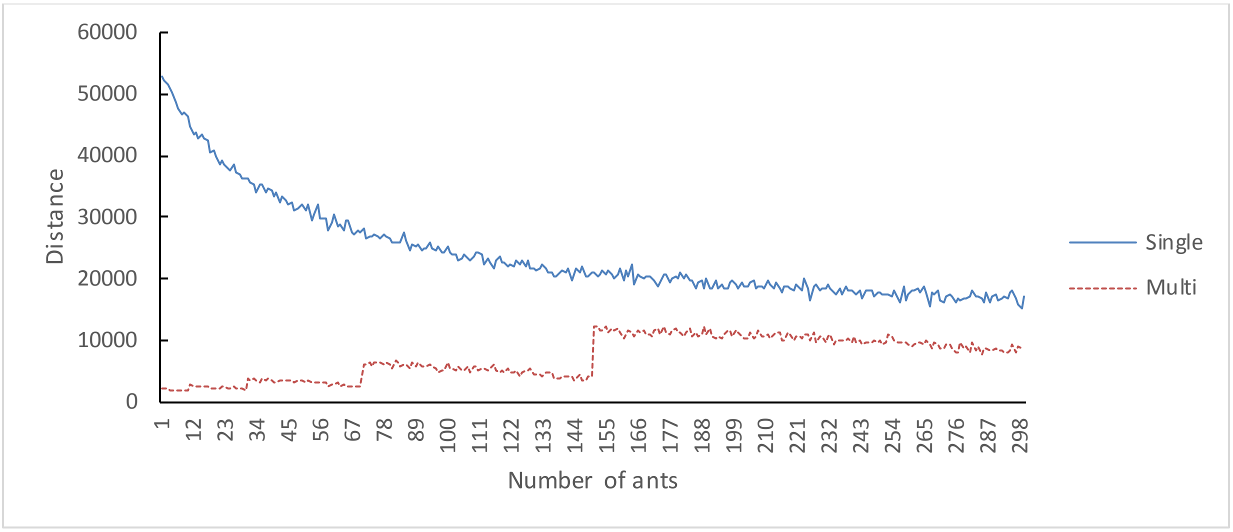


Figure 47: Single simulation on 100 vertices with 300 ants

Clustering variance

Another valuable experiment is by looking at the size of the clusters. Could it be that a larger cluster would solve to a better solution, or find one at an earlier point in time? So we began investigating the different ways to run clusters of 3 –> k nearest and started running the experiments seeing that the larger clusters at least on a small scale looks to be worsening the solution as shown in Figure 48.

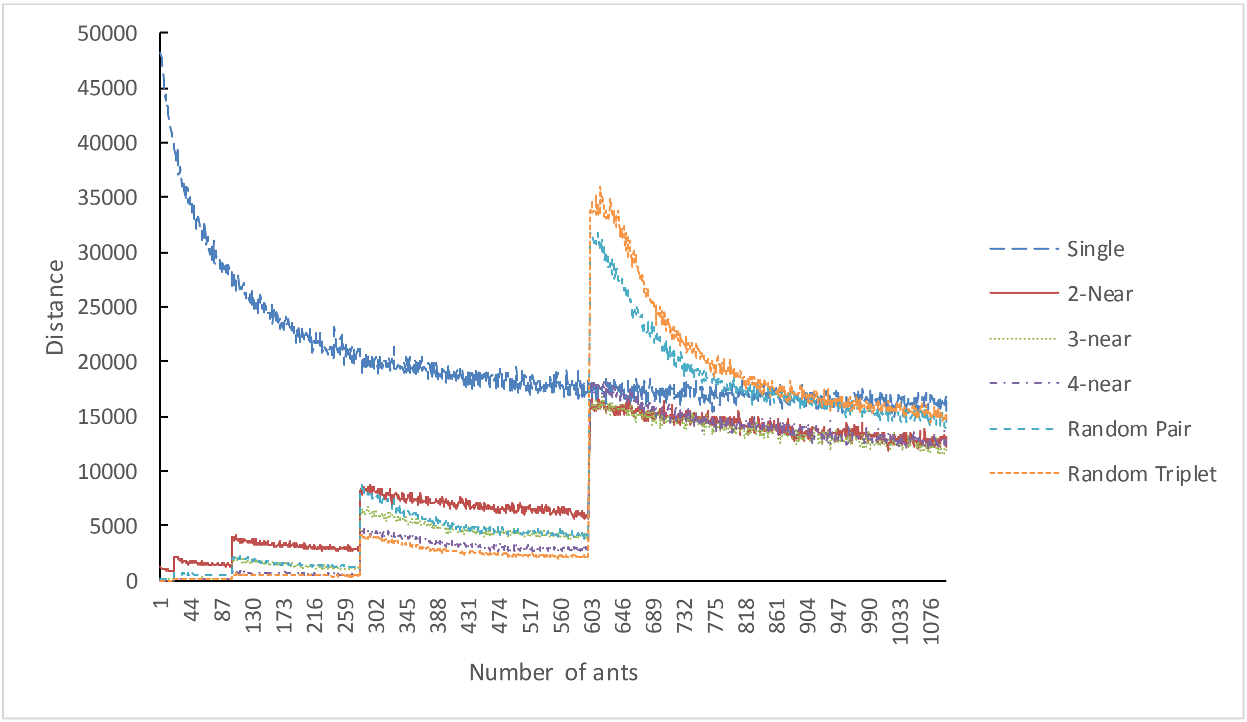


Figure 48: Comparison of different types and sizes of clusters

When expanding the random pairing up to random triplets, we see a worsening in the final solution. It is worth noting that in several of the datasets, different solutions reach lower distances in sub levels than the best at top level. This is due to the pairing method fetching the shortest distances between clusters and it becomes more visible as the vertex count in a cluster increases.

The solution is made by leveling down the problem to a simpler problem and thus solving that problem will bring the solution to the upper level as a base for which one can make a solution for the level. This allows the next problem not to start bare. Instead it can start with some knowledge of how the previous solution was solved to provide speedups and improvements in the final solution. Figure 49 shows how a solved levels solution is imprinted onto the upper level, giving as seen predefined local minima for some sets of vertices. This improved the speed of which ants improve the path as well as it eliminated many of the “dead” roads, leading to an overall better path.

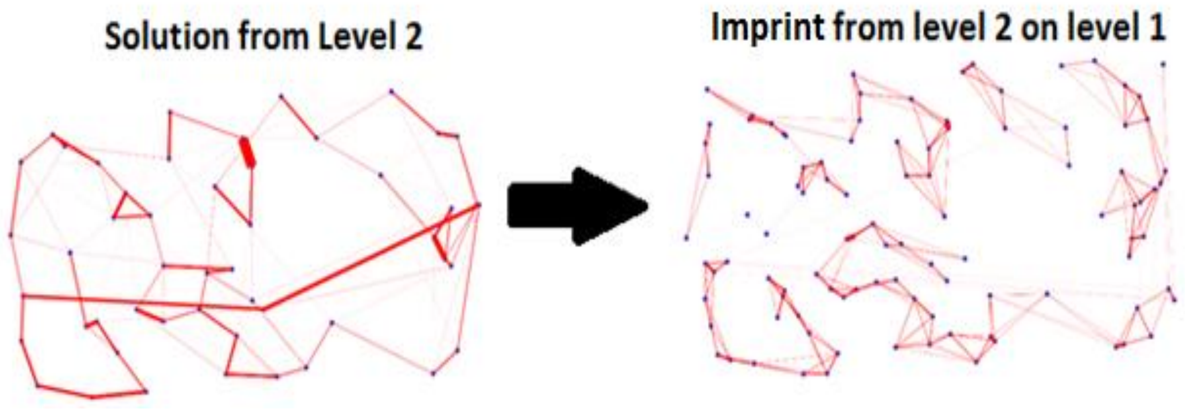


Figure 49: Imprint and improvement of path on per level increase

Chapter 5 – Experiments & Results

The experiments performed are designed to see how the different solutions act on different problems. Therefore we chose two very common problems – shortest path and TSP. Shortest path problem are problems designed solve movement in realistic environments. TSP on the other hand is a problem of combinatorics popularized by academia and in recent year's features like GPS navigation and Google Maps.

5.1 Shortest path

The initial experiments we performed were within the problem of shortest path in a uniform area. We made the setup as described in 4.2 Shortest path, making the map of simple flat room where we want to travel from corner to the opposite corner. The map was compromised of 565x428 pixels where every pixel is a vertex connecting to all surrounding vertices making 241, 820 vertices, 1,934,560 edges and we allow re-traversal of an unlimited amount of possible paths.

5.1.1 ACO vs MG-ACO

The ant colony (start) is placed at (3, 3) and food (finish) are placed onto (561, 424). Since the map we used as a baseline has nothing of interest in it, meaning that a straight line can be used as the baseline optimal path (Optimal = 421). The first ant runs on random and the initial path might be vastly different between individual runs as the first ant might find the 100% best path. For the first 5 levels the amount of ants and map size reduction is shown in Figure 50.

Level	Ants	Reduction
5	62	256
4	125	64
3	250	16
2	500	4
1	63	1

Figure 50 : Ants and resolution reduction *factor per level.*

As the ants are random in nature, they are allowed in this implementation to perform re-traversal on the map. This makes it possible for an ant to in theory never reach the goal within an acceptable time. For future experiments we would recommend for the pheromones to be stored in a way such that they also gives directions for the next vertex.

To attain the shortest path (Optimal), we ran an algorithm we developed (MT-Dijk) [69] which was built of Dijkstra's algorithm. It uses branched strands by running each of them on its own thread instead of doing the radial entailed in Dijkstra's algorithm. This was done in order to quickly calculate the shortest path between A and B in a 2D environment.

The simulation was performed and averaged over several runs comparing single level, multilevel and MT-Dijk (optimal) as shown in Figure 51. The figure shows that there are not so much changes in terms of single or multilevel results. However specific jumps might be seen at certain stages and this is caused by the level change at certain stages in the ant counter. We can see that both ACO and MG-ACO have the same performance on a per ant basis.

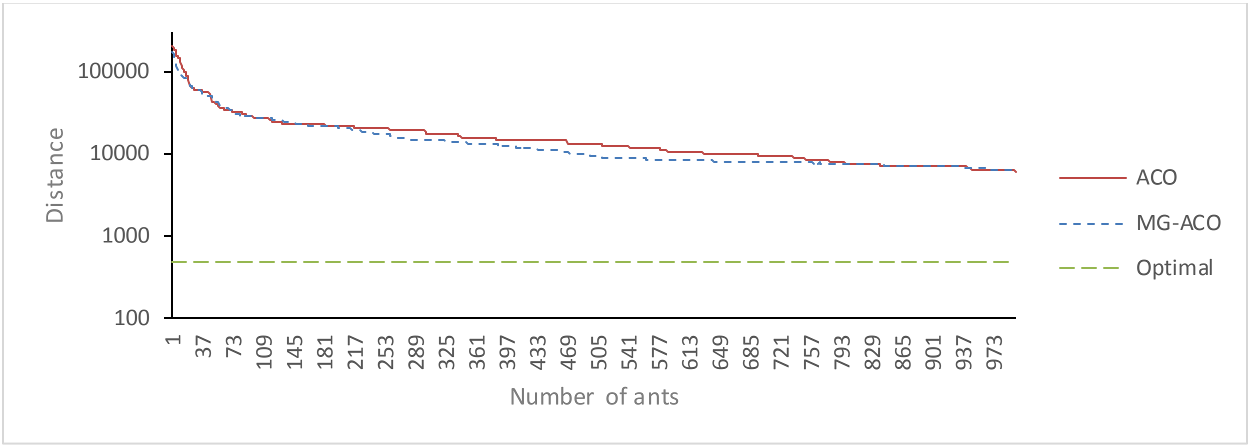


Figure 51: ACO vs MG-ACO vs Optimal (log scale) 10 simulations average

For the initial simulation map the MT-Dijk is somewhere in the order of 240x faster, solving the problem in 0.5s. A single threaded Dijkstra spent about 1.2s on the task. This allows us to assume that the MT-Dijk's performance increase is about half on one core but roughly doubles on 4 cores as it uses all threads. As the MT-Dijk is somewhat faster it will be used for the further tests.

5.1.2 Data comparison

The initial results points towards MG-ACO being both faster and cheaper in terms of memory and time as shown in Figure 52. Memory consumption was lowered in terms of megabytes per second averaged over total runtime. And the peak memory consumption is the same on both ACO and MG-ACO. The cost of running ants over the top level would be the same as running them as ACO. In theory the cost of the problem should be at least $\frac{32xy}{4^{level-1}}$bytes assuming that each vertex is a double and level 1 is top and n is the most downscaled one.

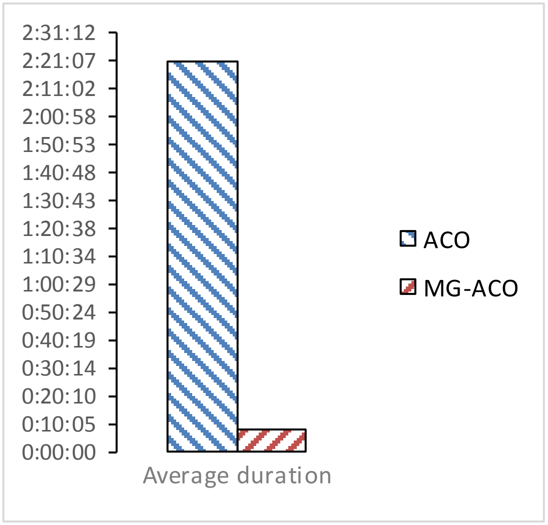


Figure 52: Average duration difference between ACO and MG-ACO

The other performance increases are time. The average time for solving the problem using the single level approach spent between 1,525s - 22,734s averaging at 8,434s. These gave us a good insight to the variable time spent solving as it is a probabilistic approach. The same time difference can be seen in the MG-ACO approach spending between 208s - 906s averaging at 479s. This leaves us with an average time improvement factor of 17.6x, ranging in the area of 1.7x for the worst case and 25.4x in the best case, when looking at it from the multilevel point of view.

This allows us to discern that the MG-ACO is better than ACO on several use-cases. However, for usage on shortest path problems, it does not perform better in terms of creating a path. Therefore MG-ACO is a viable alternative in problems un-solvable by the ACO within an acceptable timespan, as it is several times faster.

The multi grid solution is in earnest similar to decreasing the resolution of the problem and then solving the problem at the down resolved problem. The lower problems would always become $^1/_4$ of the previous problem as one combined 4 pixels into one. This made problems previously un-manageable by regular ACO into problems manageable by MG-ACO.

5.1.2 Large Maps

As an additional test we ran the MG-ACO on a map with 38,286,875 vertices making it roughly 160x larger. The map was run using 8 layers (MG8-ACO) expending 1700 ants in ~9000s. While the ants were distributed over each level making each level expending 225 ants.

As we ran the MG-ACO against MT-Dijk on small maps 561x424, the MT-Dijk outperformed the MG-ACO time wise in a factor of 240x as shown in Figure 53. As the complexity of the problem increases the difference between them decreased. The large map is 160x larger, but the time difference to MG-ACO was only 70x, showing as expected that the performance difference decrease when the size of the problem increases.

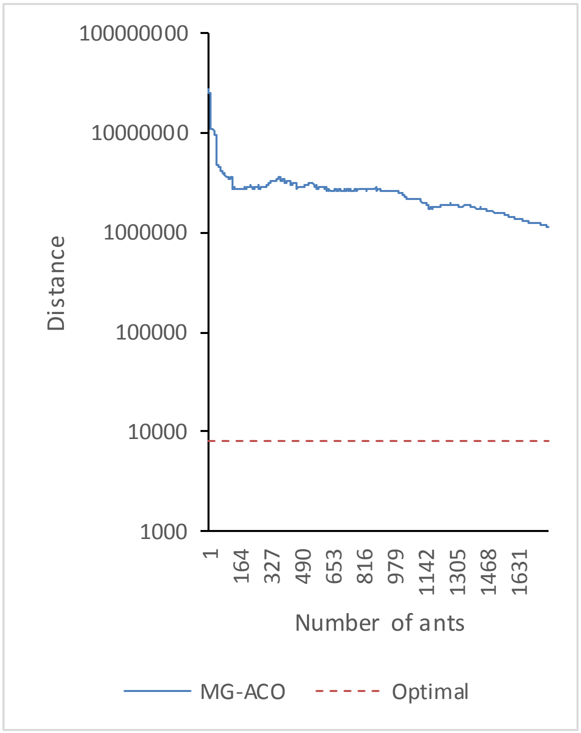


Figure 53: Large map simulation

As we know the MG-ACO is around 17x (ranging from 1.49x - 25.4x averaging at 17x) faster than ACO then we did not even try to run it over the large problem as we can assume it should be in the range of 9000s - 14,000s.

The MT-Dijk solves the XXL map in 129s (70x faster than MG-ACO) peaking at 8.5 GB of memory and freezing (fetching) during runtime compared to the MG8-ACO’s 1.6GB peak usage. Notably the setup alone without anything running just displaying and rendering the XXL image at 60 fps with double buffering used about 0.7 - 1.3 GB of memory idling. This means that MG-ACO is much cheaper than measured while MT-Dijk is only somewhat cheaper memory wise.

We compared a multithreaded variant of Dijkstra’s algorithm (MT-Dijk) that had been optimized and cleaned, against a single threaded MG-ACO that is in development. This was done on a 4 Core 8 thread Intel i7-3820 with 16 GB of memory being highly capable of handling heavy multithreaded work. The MT-Dijk spent 100% on the CPU while the MG-ACO only expended 20 - 40%. This should be taken into account in the MG-ACO’s favor as a side note rather than definite indication of anything.

5.2 Traveling Salesman Problem (TSP)

TSP is a well-known problem within decision problems as explained in earlier sections. For the experiments setup it is explained in the proposed solution, also the setup and fine-tuning is made for single level “Classical” ACO.

5.2.1 ACO

For reference we implemented a classical approach of the TSP, creating a good base to perform the multi-leveling upon. In our simulations we used 100 cities (vertices) yielding a total of 4950 possible edges as shown in Figure 54.

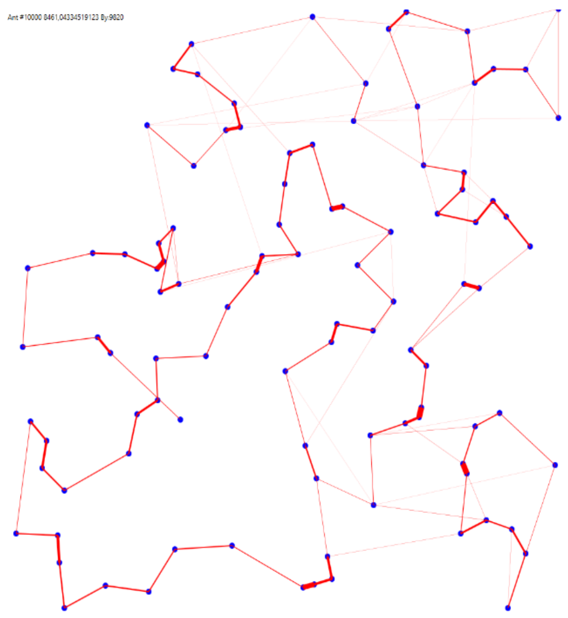


Figure 54: TSP 100 vertices (Blue) with road strength (Red) (After a classical ACO simulation)

The classical ACO approach is not the most interesting solution but it must be done as to create a sort of baseline for further experiment. The ant and variables are set to best suit the classical approach and to be ran on the multilevel in later experiments.

This means that we will tune the ants on the single level problem. In this way we will make sure that the multilevel ants are ran with the parameters optimized for single level, allowing for the evidence to be in favor of single level.

Setup wise we have performed experiments shows that the ants perform well under an exploration factor at 0.5% and pheromone dissipation rate at 5%. Running ACO on a problem set with static or random start vertex as shown in Figure 55 and Figure 56 which appears to have some impact on the chance to end up on global minima. This shows us that the set can converge on series of optimal paths even though ants have random start for simplicity and comparativeness. The future experiments will be ran with a static start vertex.

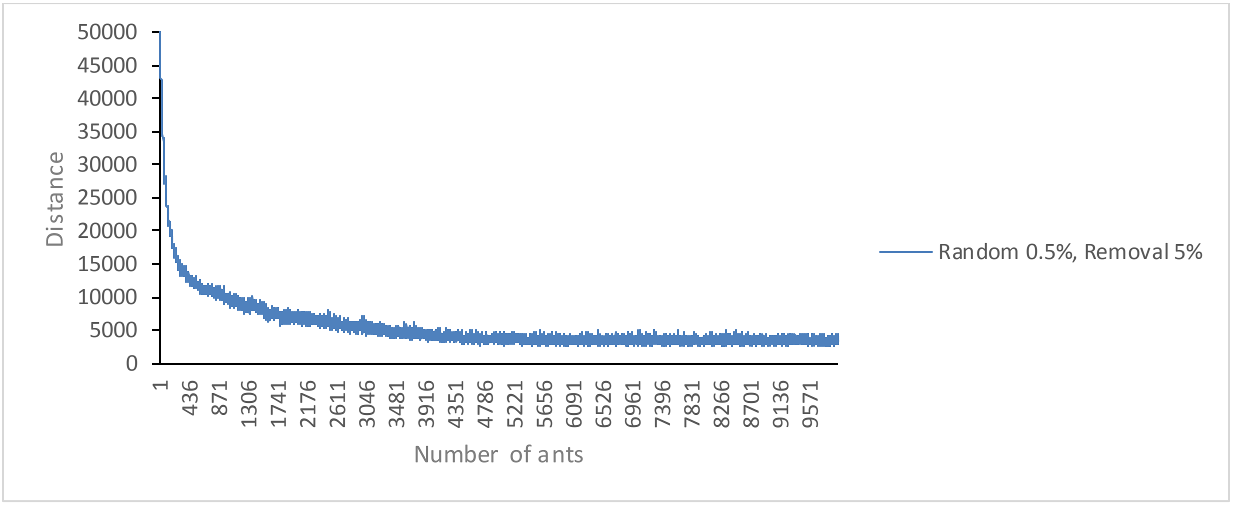


Figure 55: Static seed vertices

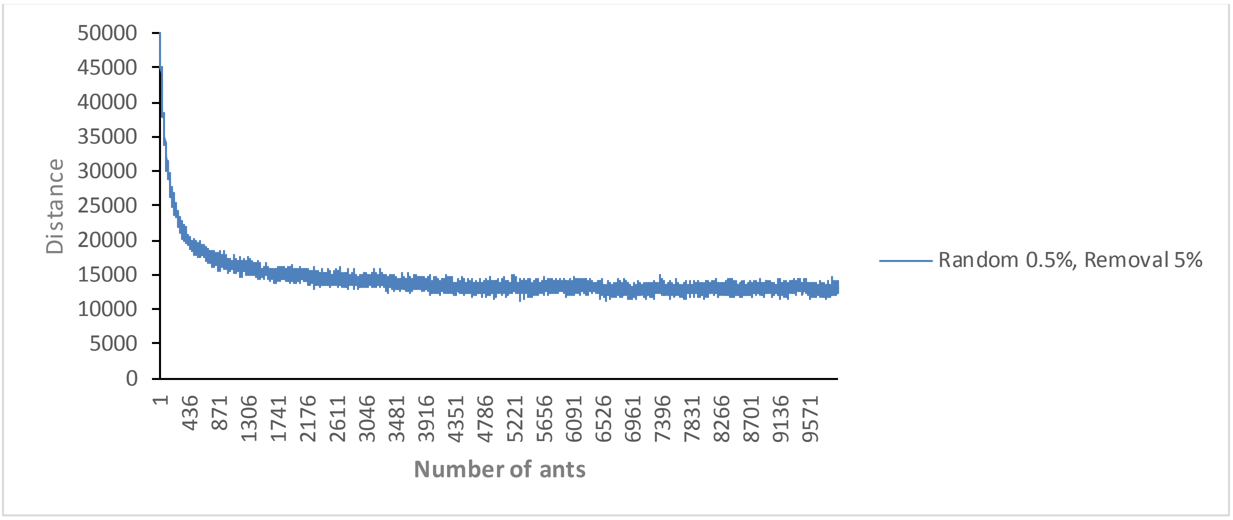


Figure 56: Random seed vertices

5.2.2 Multilevel-ACO (ML-ACO)

For the ML-ACO we started looking at different ways of clustering. One concept was to cluster the problem in large cluster as seen in Figure 57 where we have split the problem into 10 using k-nearest. But a different and simpler solution was performed where we only paired two and two, as explained in the experiment setup.

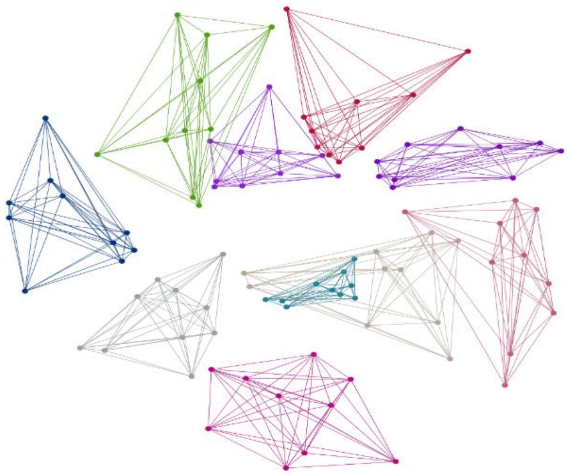


Figure 57: K-Nearest (10-Nearest) with all internal edge

This allows us to create a fairly good baseline to perform and compare against. We chose to run the multilevel by clustering two and two in a k-nearest fashion according to the research outlined [40] in 3.2 Multilevel Algorithms.

K-nearest vs Single-level

The experiments were performed with the multiplier set to 20 for 100 vertices and 9 for 500 vertices, totaling at 1100 and 1200 ants per run respectively. The first natural test was done to scale the initial test case with 300 ants up to the new ranges.

The first experiment performed was on the 100 vertices over 5 levels and 1100 ants as shown in Figure 58. This experiment shows that the multilevel approach achieves a better local minima than the single level. The same result can be seen repeated in the data collected on the 500 vertices problem in Figure 59.

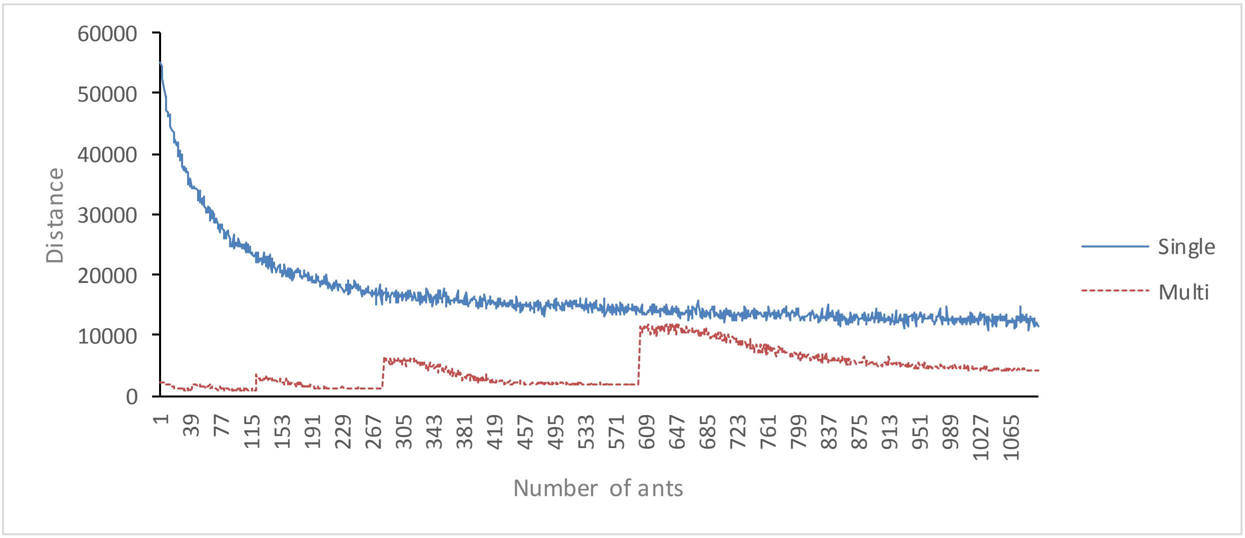


Figure 58: 25 simulations on 100 vertices with 1100 ants

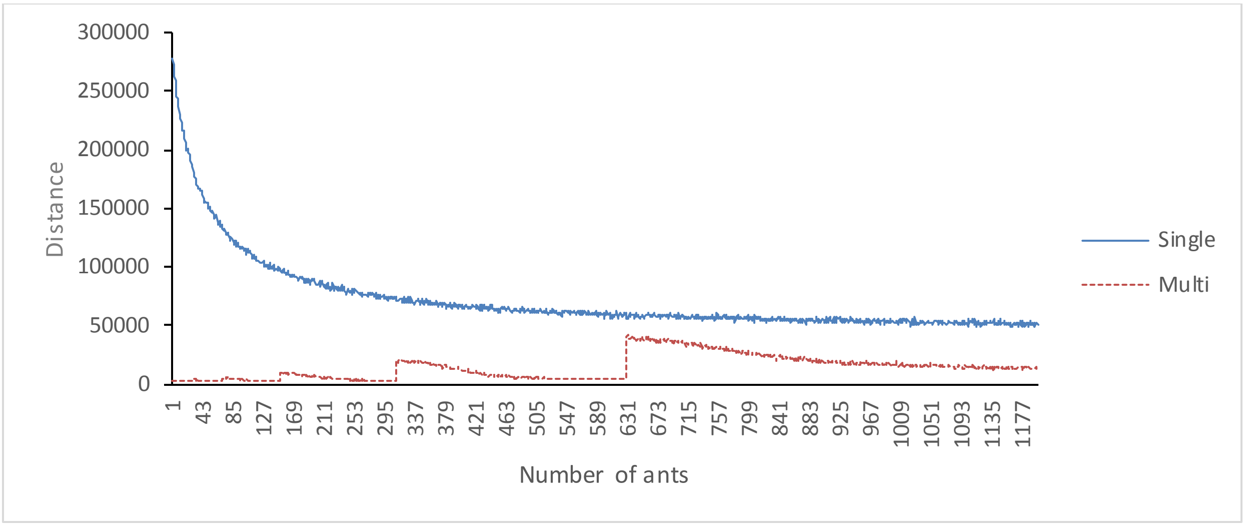


Figure 59: 25 simulations on 500 vertices with 1200 ants

The predictable solutions were successful, one could argue how this multilevel solution performs in a randomized environment. Do the k-nearest have anything to do with the outcome? Can we pair the vertices randomly?

K-nearest vs random pair

To further verify the evidence we performed more extensive test. The first environment is to do the tests on a randomized environment instead of the circular one. The outcome of running the multilevel in the randomized world is shown in Figure 60. Further inclusion of randomized pairing was also performed alongside the experiments on random maps. Displaying the performance difference and creating a randomized solution baseline.

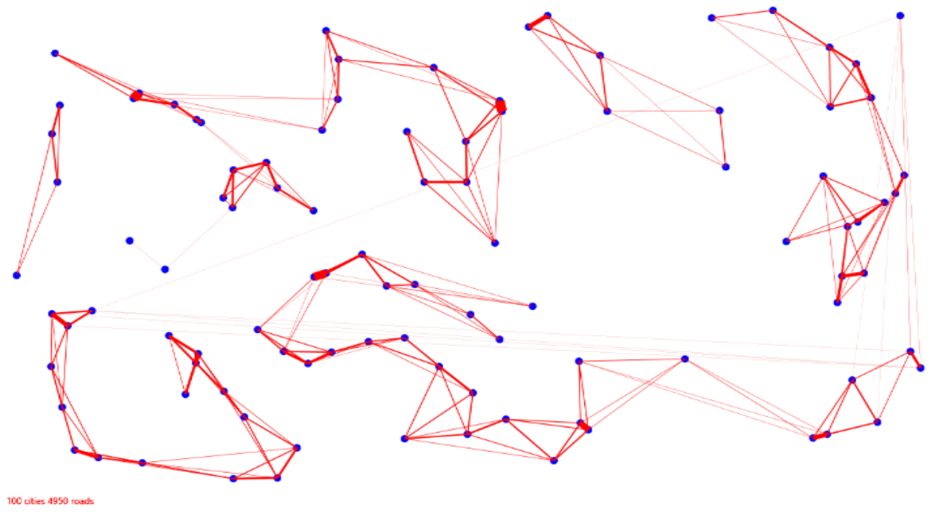


Figure 60: Outcome after a multilevel run in a randomized world. Pheromone strength/edge (Width, Red) and vertices (Blue)

As the results shown in Figure 61, performing the multilevel on a randomized environment does not give the same performance difference as the circular problem. It yields better local minima for both random pairing and k-nearest pairing over the single level. This displays an advantage of k-nearest over random pairing.

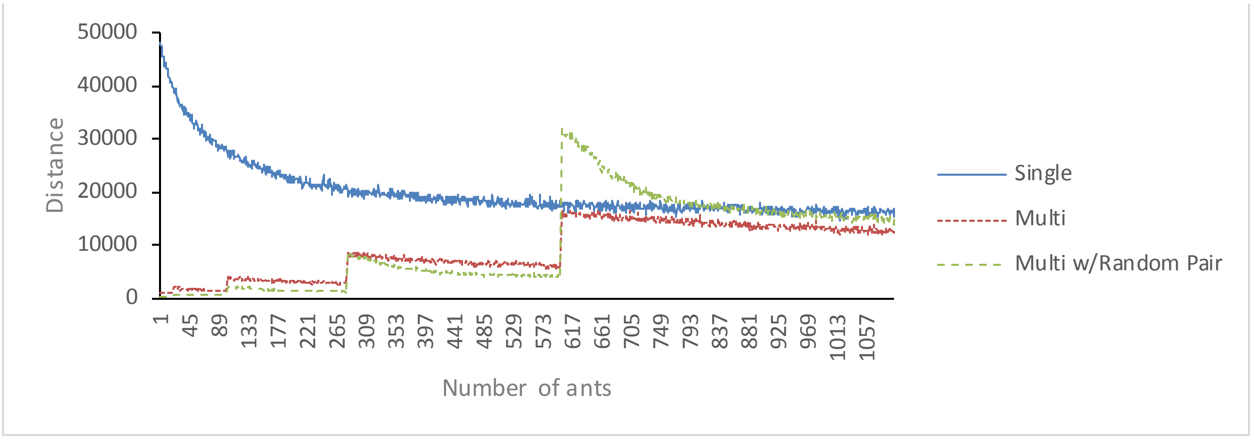


Figure 61: Nearest pairing vs random pairing (100 vertices)

The 500 vertices problem seen in Figure 62 shows that the same results are true for the 500 vertices problem as it is on the 100 vertices problem. Displaying a probable advantage of random and k-nearest over single level, where k-nearest outperform them both.

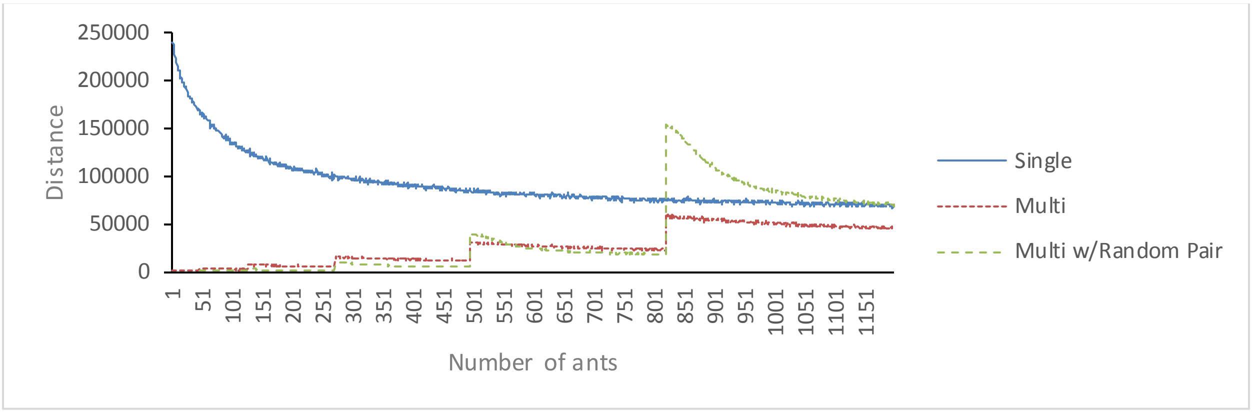


Figure 62: Nearest pairing vs random pairing (500 vertices)

Other experiments

To cover more basis some minor experiments was performed. This is because we wanted to make sure that the experiments done will remain true for more fringe cases, such as massive amounts of ants, visual conformation of solutions and improvements seen in graphs and the verification of nearest pair versus triplets and quads.

By comparing against a simple and naive algorithm, we chose to compare it against the Greedy Nearest Neighbor (NN), its counterpart Greedy Furthest Neighbor (FN) and random paths as shown in Figure 63. As we quickly noticed that the random path is as good as the first ant with no improvement and the FN is much worse. The NN is on average 25% worse than the best path [70]. But we do see that the multilevel converges towards the NN but the amount of ants is too small to actually display the crossing which requires a bigger run.

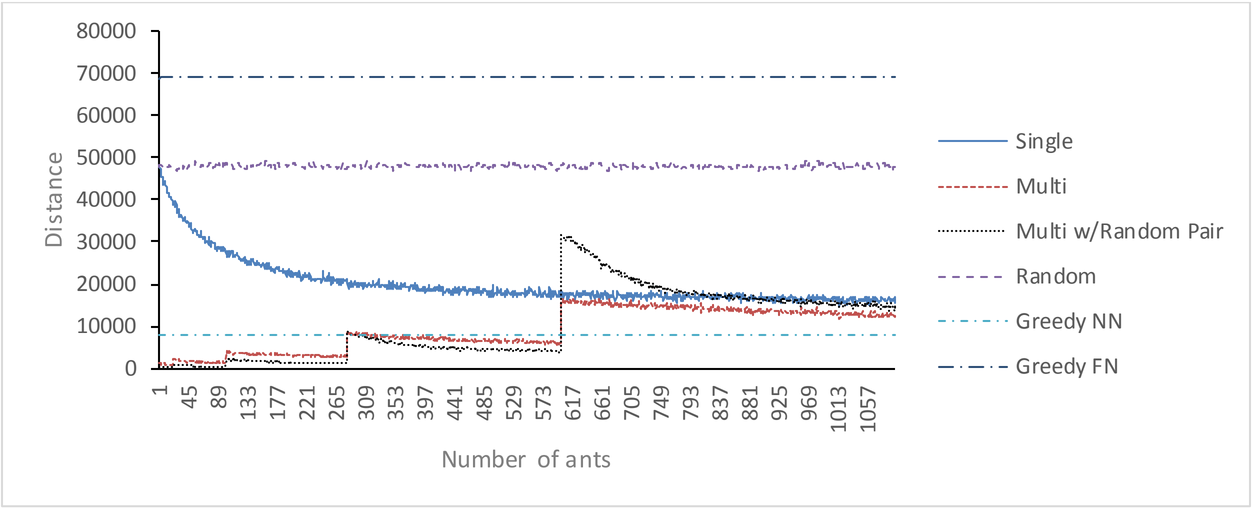


Figure 63: Comparison against random path and greedy

Large Number of Ants

To discover other features of the multilevel approach we decided on performing experiments using an immense amount of ants to confirm the convergence on a path. The multiplier was set to 1000 making the problem expend about 55,000 ants as shown in Figure 64.

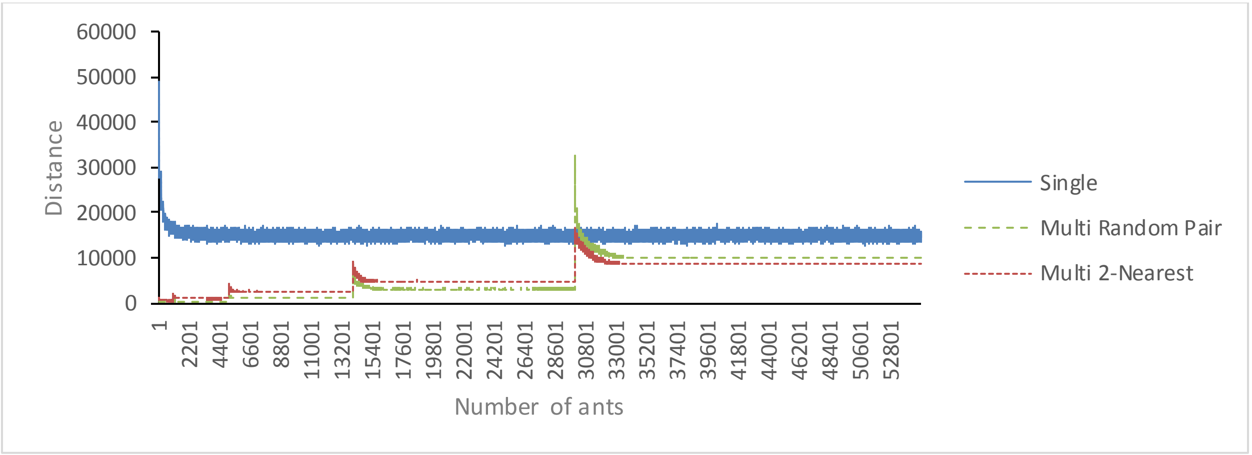


Figure 64: Simulation of 55,000 ants over 100 vertices comparing random pair against nearest pair

The experiment in Figure 64 displays the single level ACO becoming “stuck” on local minima. The randomized pairing ends up on another local minimum and nearest 2 pairing method produce an even better local minimum than random pairing. This result conforms to the results seen in this experiments that we can assume based on the data that the ant count at which the problem is leveled could have been done at earlier a more rapid rate.

Looking at the bigger picture allows us to achieve solutions which are seen in some cases only over a large number of ants. This displays problems only seen in large runs, for instance the leveling is not being optimally placed, for the reason that it converges on each of the level steps for some time before it performs the “jump”.

Performance wise we see that multi-leveling on TSP manages to perform at the level of worst solution from a greedy nearest neighbor solution as shown in Figure 65. Because ACO is a probabilistic and the greedy algorithm is deterministic, this means that the ant cannot be forced to give the worst or a bad solution, in opposition to greedy being forcible to deliver the worst path in a TSP [71]. This point towards ACO being a good alternative to the greedy algorithm, as the 2-nearest pairing and leveling can be used to gain a comparable result to that of an expected result of a greedy TSP solution.

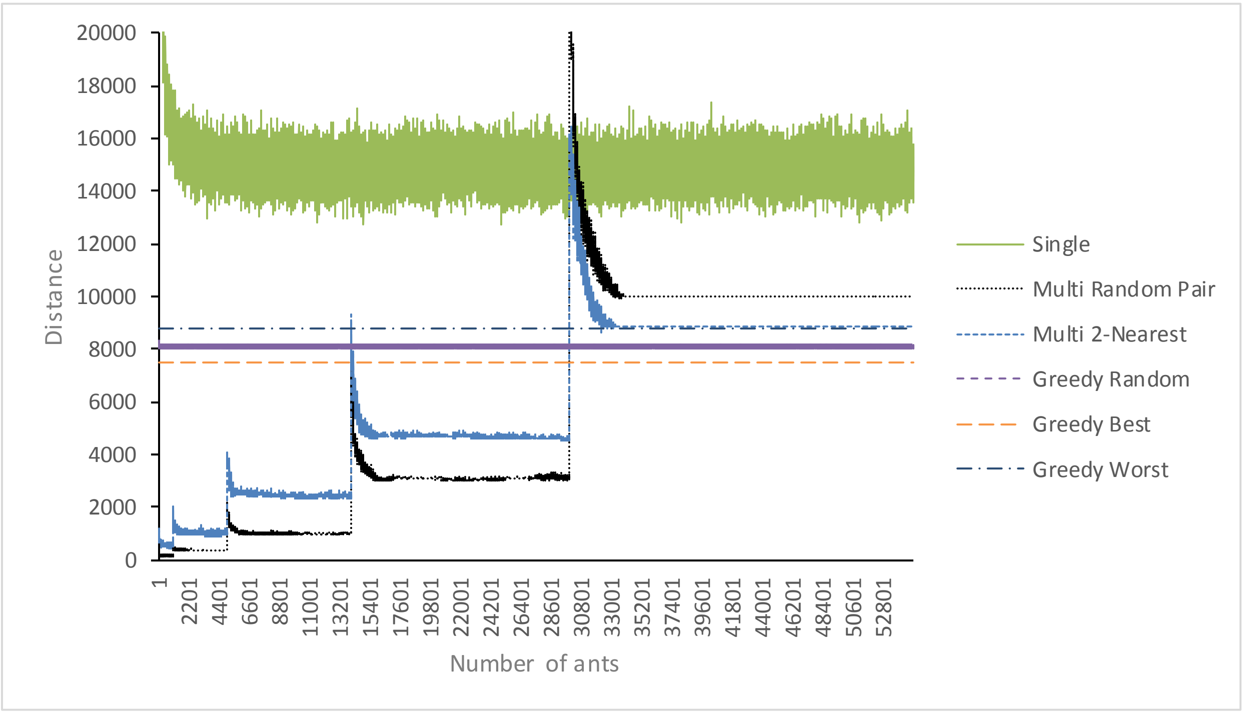


Figure 65: Simulation of 55,000 ants over 100 vertices, comparing random pair and nearest pair against greedy nearest neighbor

As the leveling is something we would assume at earlier times in the multi run and thus decrease the runtime spent. We ran a simulation with shorter intervals on the levels over 10,000 ants, showing that the leveling can be performed only within the initial drop in a regular ACO run. This shows that the ants can be forced to converge on a better solution in about the same time of the regular ACO as seen in Figure 66.

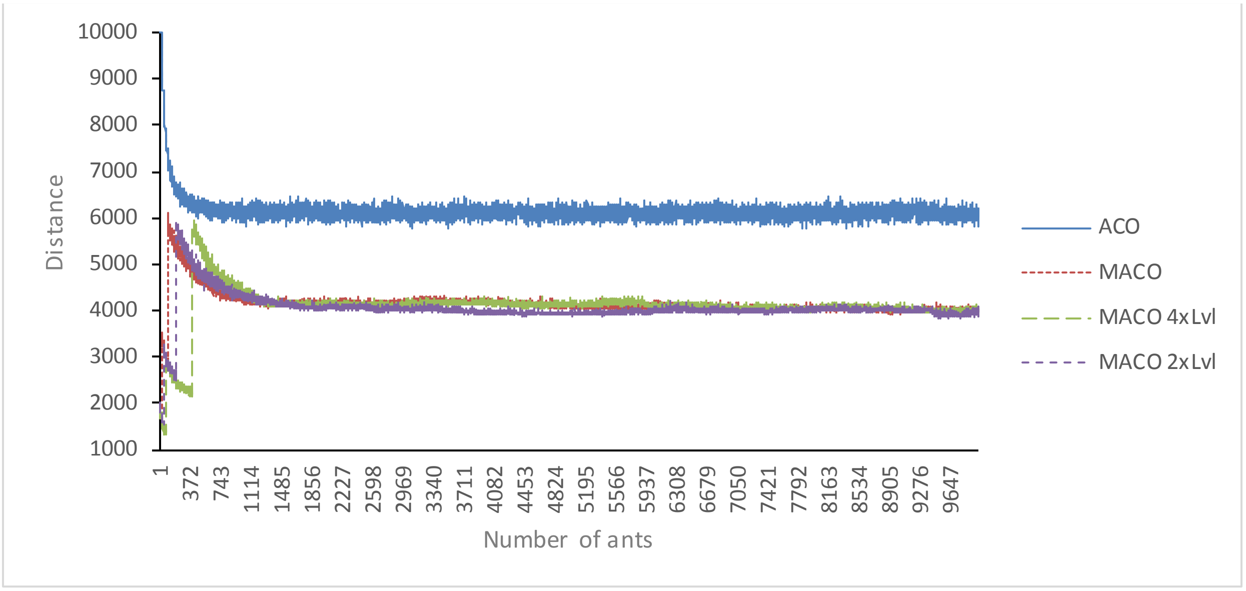


Figure 66: Comparison of the amount of ants used to perform the leveling

There are areas where one may have wanted to do things differently, most notably in the optimization of the ants for different solutions or levels as shown. We assume that one can achieve better solutions by using fewer ants with optimized exploration factor, pheromone dissipation, pheromone amount and/or keeping the levels until convergence before leveling up instead of using a rigid rule or in our case a formula.

That aside, as we used the same variables for single and multilevel ants, this allowed us to see clearly the difference in performance when doing multilevel versus single level. Not only did the multilevel outperform the single level in all experiments measured on the TSP in shear distance found. But it also performed in a less random “noisy” manner.

One can assume that the single level solution would get a head start while the multilevel actually performs the work by building the layers. However with smart structuring, the system should only have to be structured once and then the action of adding and removing would be worst case as big as adding k-vertices to the solution. Whereas k is the amount of levels and each level contains a cluster of the upper vertices. In best case it should be the same cost as adding one vertex to the problem.

Thus decreasing was the possible head start gained by the single level at initial clustering. And at each leveling up of the problem, the multilevel solution is multitudes faster on the lower leveled problems. As the complexity of the problem has been decreased to $\frac{vertices}{2^{level}}$, allowing for a problem of 1000 vertices containing 999! solutions. Edges to be solved as problems only contain 3+1, 7+1, 15+1, 31, 62+1, 125, 250, 500 and 1000 vertices where the +1 is a cluster that is not full for the TSP.

Graphical visualization of our Solution

A more direct way where one can observe the multilevel advantage is by direct observation of samples at certain intervals. Allowing for a quick overview of the results at certain stages, this provides an easy spotting of the differences between single level and multilevel. Figure 67 illustrates that the multilevel imprinted solution helps to solve the solution at a much quicker pace than the single level ACO solution. To achieve the ant count for the multilevel we adjusted the level multiplier accordingly.

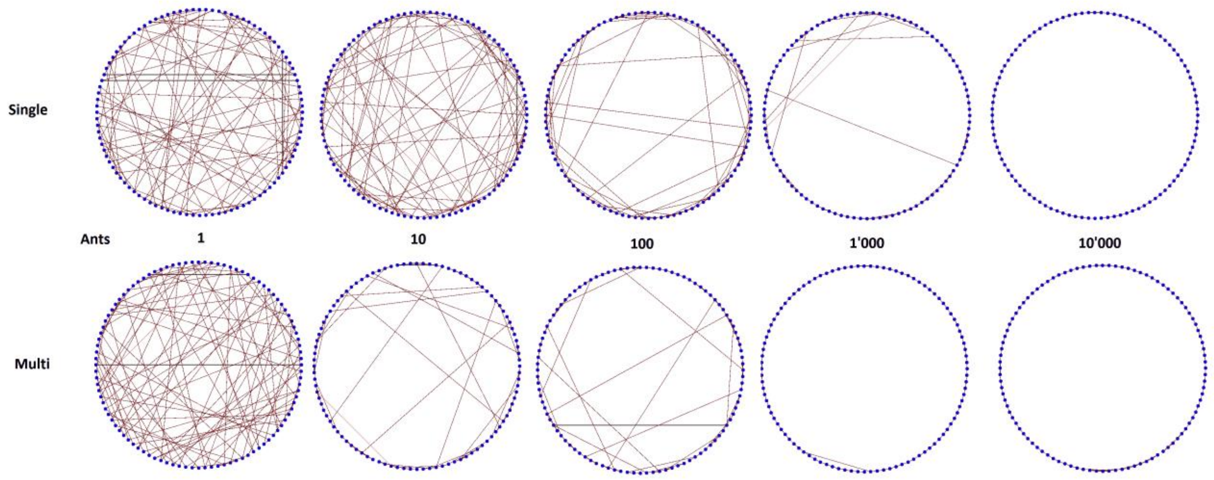


Figure 67: Visual comparison of multilevel vs single level on a symmetric TSP

This strongly implies that multilevel approaches will outperform single level approaches in most, if not all problems where one will prefer to use an ant colony optimization algorithm. By using a multilevel approach to ACO will give a range of advances within the solution albeit not all at the same time. Improvements range from improved local minima, lower memory consumption and shorter CPU time. This also allows us to make other assumed improvements within areas like Ethernet communications where one could assume that less packets would be needed to discover good paths through an Autonomous System.

Chapter 6 – Conclusion

Ant colony optimization metaheuristic is a method inspired by foraging behavior of real ants. Ant colony optimization has already shown a very good performance in many combinatorial problems and for some real applications, but research on multilevel variants of the algorithms is almost completely missing in the literature. In this master thesis, contrarily to most of research on shortest path and traveling salesman problem are on single level ant colony optimization approach. We have presented and studied some multilevel variants of ant colony optimization, where the searching ability and computation time are the objectives to optimize.

In this work we have proposed multilevel variants of ant colony optimization and discussed the different elements for multilevel algorithm from various researches which gave us a better understanding of the multilevel scheme. The main objective of the proposed algorithm is to increase the performance of the single level ant colony optimization algorithms in solving the two given problems – shortest path and traveling salesman problem.

The multilevel variant in solving shortest path problem have gained advantages within hardware cost with a drastically lowered overall run time by a factor of about 10-20x and decent lowering of the average memory consumption. Traveling salesman problem multilevel achieved an overall better solution than single level providing a solution matching with the greedy nearest neighbor algorithm.

Therefore indicating that the multilevel should provide a solution only 25% longer than the best path on randomized problems and would not, in contrast to greedy neighbor, cannot lured into providing the worst path. In our experiments the multilevel approach yielded a path about 20 - 50% shorter on random maps and on circular maps it expends about 10^{n-1} ants to perform what single level expends 10^{n} ants to do.

We therefore conclude that the performance of multileveled approach on both shortest path and traveling salesman problem have achieved great advantages and successfully increased the ability of single level ant colony optimization algorithm.

Chapter 7 – Future Work

With introduction of multilevel ant colony optimization, it opens up for additional research in the field. Future work includes uncovering what a distributed solution might add to the results. Other possibly more complex forms of clustering and/or leveling are also something one would assume being of some worth. For instance, the problem of shortest path, mesh refinement as seen in the world of graphics, the mesh in 3D graphics use triangles of varying size and complexity, depending on the interest of the users view point. This could be used as a means for a dynamical fine-grained multilevel approach.

For the clustering of TSP there are exists several other ways to perform it. Our approach of doing it pair wise is a fairly straightforward approach yielding good results. However more interesting and complex approaches such as the very similar approach of k-means to eliminate the outliers before clustering. This can be used in the final solution or even classify them at each level to solve them when they were in bound – at the level detected. This all shows that there are areas where other clustering approaches could be fitted, but we assume by looking at the collected data that multi-leveling attain better solutions.

Furthermore, the proposed multilevel ant colony optimization algorithm can enhance the exploration factor, pheromone dissipation, pheromone amount or other parameters to enrich the diversity of solution and increase the ability of finding an optimal solution.

Bibliography

[1] "Ant Colony Optimization," Wikipedia, The Free Encyclopedia, February 2014. [Online]. Available: http://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms. [Accessed 20 February 2014].

[2] M. Dorigo, "Optimization, Learning and Natural Algorithms, PhD thesis," Politecnico di Milano, Italy, 1992.

[3] C. Bourjot, D. Desor and V. Chevrier, "Stigmergy, in Self-organising Software," in *Springer-Verlag*, Berlin Heidelberg, 2011.

[4] J. Duan, Y.-a. Zhu and S. Huang, "Stigmergy Agent and Swarm-intelligence-based Multi-agent System," *Proceedings of the 10th World Congress on Intelligent Control and Automation (WCICA),* pp. 720-724, 6-8 July 2012.

[5] K. Khurshid, S. Irteza, A. A. Khan and S. I. Shah, "Application of Heuristic (1-Opt Local Search) and Metaheuristic (Ant Colony Optimization) Algorithms for Symbol Detection in MIMO Systems," *Communications and Network,* pp. 200-209, 15 August 2011.

[6] M. Dorigo and L. M. Gambardella, "Ant Colony System: A Cooperative Learning," *Evolutionary Computation, IEEE Transactions,* vol. 1, no. 1, pp. 53-66, April 1997.

[7] E. Tellez-Erniquez, E. Mezura-Montes and C. A. Coello Coello, "An ant system with steps counter for the job shop scheduling problem," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress,* Singapore, 2007.

[8] V. Maniezzo and A. Colorni, "The ant system applied to the quadratic assignment problem," *Knowledge and Data Engineering, IEEE Transactions,* vol. 11, no. 5, pp. 769-778, September/October 1999.

[9] R. J. Mullen, D. Monekosso, S. Barman, P. Remagmino and P. Wilkin, "Artificial Ants to Extract Leaf Outlines and Primary Venation Patterns," *Ant Colony Optimization and Swarm Intelligence Lecture Notes in Computer Science,* vol. 5217, pp. 254-258, 22-24 September 2008.

[10] M. Dorigo and L. M. Gambardella, "Ant colonies for the travelling salesman problem," *BioSystems,* vol. 43, no. 2, pp. 73-81, July 1997.

[11] Z. Li, P. Wuliang and Z. Zhongliang, "An Ant Colony System for Solving Resource Leveling Problem," in *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference*, Changsha, 2010.

[12] T. Stutzle and H. Hoos, "MAX-MIN Ant System and local search for the traveling salesman problem," in *Evolutionary Computation, 1997., IEEE International Conference*, Indianapolis, IN,

1997.

[13] Z. Wu, N. Zhao, G. Ren and T. Quan, "Population declining ant colony optimization algorithm and its applications," *Expert Systems with Applications*, vol. 36, no. 3, pp. 6276-6281, April 2009.

[14] G. M. Jaradat and M. Ayob, "An Elitist-Ant System for Solving the Post-Enrolment Course Timetabling Problem," in *Database Theory and Application, Bio-Science and Bio-Technology*, vol. 118, Springer Berlin Heidelberg, 2010, pp. 167-176.

[15] M. Tuba and Jovanovuc, "An Analysis of Different Variations of Ant Colony Optimization to the Minimum Weight Vertex Cover Problem," *WSEAS Transactions on Information Science and Applications*, vol. 6, no. 6, pp. 936-945, 6 June 2009.

[16] K. Khantanapoka and K. Chinnasarn, "Pathfinding of 2D & 3D game real-time strategy with depth direction A∗ algorithm for multi-layer," in *Natural Language Processing, 2009. SNLP '09. Eighth International Symposium*, Bangkok, 2009.

[17] P. J. Drews, D. G. Macharet and M. M. Campos, "A Terrain-Based Path Planning for Mobile Robots with Bounded Curvature," in *Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), 2012 Brazilian*, Fortaleza, 2012.

[18] A. A. Razavian and J. Sun, "Cognitive based adaptive path planning algorithm for autonomous robotic vehicles," in *SoutheastCon, 2005. Proceedings. IEEE*, 2005.

[19] L. Fu, D. Sun and L. R. Rilett, "Heuristic shortest path algorithms for transportation applications: State of the art," *Computers and Operations Research*, vol. 33, no. 11, pp. 3324-3343, 11 November 2006.

[20] R. Graham, H. McCabe and S. Sheridan, "Pathfinding in Computer Games," *ITB Journal*, 2004.

[21] NASA, [Online]. Available: http://www.carnegiemuseums.org/cmag/bk_issue/2004/janfeb/images/csc_image3.jpg. [Accessed 10 March 2014].

[22] "Mars Pathfinder NASA official," 10 August 2012. [Online]. Available: http://www.nasa.gov/mission_pages/mars-pathfinder/index.html. [Accessed 9 March 2014].

[23] Wikipedia , "Autonomous Car Wikipedia The Free Encyclopedia," Wikimedia Foundation, Inc., [Online]. Available: http://en.wikipedia.org/wiki/Autonomous_car. [Accessed 10 March 2014].

[24] J.-H. Ryu, D. Ogay, S. Bulavintsev, H. Kim and J.-S. Park, "Development of an Autonomous Vehicle for High-Speed Navigation and Obstacle Avoidance," *Intelligent Autonomous Systems 12*, vol. 1, pp. 101-109, 26-29 June 2012.

[25] A. Yahja, S. Singh and A. Stentz, "An Efficient On-line Path Planner for Outdoor Mobile Robots,"

Robotics and Autonomous Systems 32, 2000.

[26] C. Thiemann, F. Theis, D. Grady, R. Brune and D. Brockmann, "The Structure of Borders in a Small World," *PLoS One 5*, vol. 5, no. 11, 18 November 2010.

[27] M. Noto and H. Sato, "A Method for the Shortest Path Search by Extended Dijkstra algorithm," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on (Volume:3)*, Nashville, TN, 2000.

[28] [Online]. Available: http://qiao.github.io/PathFinding.js/visual/. [Accessed 19 March 2014].

[29] "A* search algorithm, Wikipedia, The Free Encyclopedia," Wikimedia Foundation, Inc., [Online]. Available: http://en.wikipedia.org/wiki/A*_search_algorithm. [Accessed 19 march 2014].

[30] "Heuristic, Wikipedia The Free Encyclopedia," Wikimedia Foundation, Inc., [Online]. Available: http://en.wikipedia.org/wiki/Heuristic. [Accessed 19 March 2014].

[31] S. Mertens, "Construction Heuristics," [Online]. Available: http://www-e.uni-magdeburg.de/mertens/TSP/node2.html. [Accessed 19 March 2014].

[32] L. G. Tavares, H. S. Lopes and C. R. Erig Lima, "Construction and Improvement Heuristics applied to the Capacitated Vehicle Routing Problem," in *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress*, Coimbatore, 2009.

[33] "Metaheuristic, Wikipedia, The Free Encyclopedia," Wikimedia Foundation, Inc., [Online]. Available: http://en.wikipedia.org/wiki/Metaheuristic. [Accessed 22 March 2014].

[34] "VANET/ITS Website," [Online]. Available: http://neo.lcc.uma.es/staff/jamal/portal/?q=content/genetic-algorithm-ga. [Accessed 9 May 2014].

[35] C. Blum, "Ant colony optimization: Introduction and recent trends," *Physics of Life Reviews*, vol. 2, no. 4, pp. 353-373, December 2005.

[36] L. M. Gambardella and M. Dorigo, "Ant-Q: A Reinforcement Learning approach to the travelling salesman problem," in *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[37] A. Brandt, "Multilevel computations: Review and recent developments," in *Multigrid methods: Theory, applications, and supercomputing, proceedings of the 3rd Copper Mountain conference on multigrid methods,* New York, 1988.

[38] C. Walshaw and M. Cross, "Mesh Partitioning: a Multilevel Balancing and Refinement Algorithm," *Society for Industrial and Applied Mathematics Journal on Scientific Computing,* vol. 22, no. 1, pp. 63-80, 2000.

[39] C. Walshaw, "Multilevel Refinement for Combinatorial Optimisation Problems," *Annals of Operations Research,* vol. 131, pp. 325-372, 2004.

[40] C. Walshaw, "A Multilevel Approach to the Travelling Salesman Problem," in *Operations Research,* 2002.

[41] S. T. Barnard and H. D. Simon, "Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Concurrency: Practice and Experience,* vol. 6, no. 2, pp. 101-117, 1994.

[42] B. Hendrickson and R. Leland, "A Multi-Level Algorithm For Partitioning Graphs," *Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference,* p. 28, 1995.

[43] P. Korosec, J. Silc and B. Robic, "Mesh Partitioning: A Multilevel Ant-Colony-oOtimization Algorithm," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003.

[44] K. Taskova, P. Korosec and J. Silc, "A Distributed Multilevel Ant Colonies Approach," *Informatica,* vol. 32, no. 3, pp. 307-317, 2008.

[45] "Princeton University Mathematics," [Online]. Available: http://web.math.princeton.edu/math_alive/5/Notes2.pdf. [Accessed 19 May 2014].

[46] "Wikipedia, The Free Encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Lin%E2%80%93Kernighan_heuristic. [Accessed 19 May 2014].

[47] P. K. Jimack, "Parallel and Multilevel Algorithms for Computational Partial Differential Equations," *International Symposium on Distributed Computing and Applications to Business, Engineering and Science Proceedings,* vol. 1, 2004.

[48] M. W. Dixon, G. R. Cole and M. I. Bellgard, "A neural network shortest path algorithm for routing in packet-switched communication networks," in *Communications, 1995. ICC '95 Seattle, 'Gateway to Globalization', 1995 IEEE International Conference* , Seattle, WA, 1995.

[49] Z. Chang and G. Gaydadijev, "On improved MANET network utilization," in *Wireless Communications & Signal Processing (WCSP), 2012 International Conference,* Huangshan, 2012.

[50] "Bowdoin.edu," [Online]. Available: http://www.bowdoin.edu/~ltoma/teaching/cs231/spring14/Lectures/15-shpaths/shpaths.pdf. [Accessed 10 April 2014].

[51] "Valvesoftware," [Online]. Available: http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf. [Accessed 9 April 2014].

[52] D. Fain and P. Shi, "Improvement of Dijkstra's algorithm and its application in route planning," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference*, Yantai, Shandong, 2010.

[53] Y. Xu, Z. Wang, Q. Zheng and Z. Hand, "The Application of Dijkstra's Algorithm in the Intelligent Fire Evacuation System," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2012 4th International Conference*, Nanchang, Jiangxi, 2012.

[54] B. Gonen, "Genetic Algorithm Finding the Shortest Path in Networks," in *The 2011 International Conference on Genetic and Evolutionary Methods*, Las Vegas, nevada, USA, 2011.

[55] C. W. Ahn and Ramakrishna, R.S., "A genetic algorithm for shortest path routing problem and the sizing of populations," in *Evolutionary Computation, IEEE Transactions*, 2002.

[56] D. Angus, "Solving a unique Shortest Path problem using ant colony optimisation," *Communicated by T. Baeck*, 2005.

[57] M. Glabowski, B. Musznicki, P. Nowak and P. Zwierzykowski, "Shortest Path Problem Solving Based on Ant Colony Optimization Metaheuristic," *Image Processing & Communications*, vol. 17, no. 1-2, pp. 7-18, December 2012.

[58] O.-C. G. J. R. Morten Goodwin, "Escape Planning in Realistic Fire Scenarios," Grimstad, February 5, 2014.

[59] [Online]. Available: http://www.csd.uoc.gr/~hy583/papers/ch11.pdf. [Accessed 5 April 2014].

[60] H. Braun, "On solving travelling salesman problems by genetic algorithms," *Parallel Problem Solving from Nature*, vol. 496, no. 0302-9743, pp. 129-133, 1991.

[61] "AForge.NET Framework," [Online]. Available: http://www.aforgenet.com/framework/samples/genetic_algorithms.html. [Accessed 7 April 2014].

[62] L. Qu and R. Sun, "A synergetic approach to genetic algorithms for solving traveling salesman problem," *Information Sciences*, vol. 117, no. 3-4, pp. 267-283, August 1999.

[63] Y. Nagata and S. Kobayashi, "An analysis of edge assembly crossover for the traveling salesman problem," in *An analysis of edge assembly crossover for the traveling salesman problem*, Tokyo, 1999.

[64] M. Dorigo, V. Maniezzo and A. Colomi, "The Ant System: Optimization by a colony of cooperating agents," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B*, vol. 26, no. 1, pp. 29-41, 1996.

[65] "ACO Travelling Salesman Problem," [Online]. Available: http://en.wikipedia.org/wiki/File:Aco_TSP.svg. [Accessed 26 March 2014].

[66] W. Zhao, X. Cai and Y. Lan, "A New Ant Colony Algorithm for Solving Traveling Salesman Problem," in *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference*, Hangzhou, 2012.

[67] J. Ji, Z. Huang, Y. Wang and C. Liu, "A New Mechanism of Pheromone Increment and Diffusion for Solving Travelling Salesman Problems with Ant Colony Algorithm," in *Natural Computation, 2008. ICNC '08. Fourth International Conference*, Jinan, 2008.

[68] Z. Wu, N. Zhao, G. Ren and T. Quan, "Population declining ant colony optimization algorithm and its applications," *Expert Systems with Applications*, vol. 36, no. 3, pp. 6276-6281, April 2009.

[69] T. A. Lian, "Feasibility study of autonomous flight for mapping of hostile areas.," Grimstad, 2013.

[70] D. Johnson and L. McGeoch, "The traveling salesman problem: A case study in local optimization," *Local search in combinatorial optimization*, pp. 215-310, 1997.

[71] A. Y. A. Z. Gregory Gutina, "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP," *Discrete Applied Mathematics*, vol. 117, no. 1-3, p. 81–86, 2002.

[72] R. Leigh, S. J. Louis and C. Miles, "Using a Genetic Algorithm to Explore A*-like Pathfinding Algorithms," in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, Honolulu, HI, 2007.

[73] "Particle swarm optimization, Scholarpedia," [Online]. Available: http://www.scholarpedia.org/article/Particle_swarm_optimization. [Accessed 24 March 2014].

[74] "An Ant's Rant, Daily Kos," [Online]. Available: http://www.dailykos.com/story/2010/04/11/856217/-An-Ant-s-Rant. [Accessed 24 March 2014].

[75] Z. C. S. S. Hliang and M. A. Khine, "An Ant Colony Optimization Algorithm for Solving Traveling Salesman Problem," in *2011 International Conference on Information Communication and Management*, Singapore, 2011.

[76] U. R. Craig C. Douglas, "What are Multigrid Methods?," [Online]. Available: http://www.mgnet.org/mgnet/tutorials/xwb/mg.html. [Accessed 08 03 2014].

If you see this then we would like to thank you for reading our master's thesis.

Appendices

Appendix A

Ant Code Shortest Path

```
namespace Masters.Algorithm.ACO
{
    public class Ant
    {
        private int ID;
        private int[] pos;
        private int[] posChange = new int[] { 0, 0 };
        private int distanceMoved = 0;
        private Random rand;
        bool discoveryAnt = false;
        public bool[,] beenAt;
        public int[,] Path;
        public Ant(int seed)
        {
            rand = new Random(seed);
            discoveryAnt = rand.NextDouble() < 0.1;
            ID = AntStore.getID++;
            pos = new int[] { Store.startX, Store.startY };
            Path = new int[Store.Width, Store.Height];
            beenAt = new bool[Store.Width, Store.Height];
            for (int y = 0; y < Store.Height; y++)
            {
                for (int x = 0; x < Store.Width; x++)
                {
                    beenAt[x, y] = false;
                    Path[x, y] = -1;
                }
            }
            StartAnt();
        }
        private void StartAnt()
        {
            //Set first node visited
            beenAt[pos[0], pos[1]] = true;
            //Find path
            while (Store.stopX != pos[0] || Store.stopY != pos[1]) //While not
finished
            {
                getNextDirection();
                if (isMovementValid())
                {
                    Move();
                }
            }
            WorkPhermone();

        }
        private void WorkPhermone()
        {
            distanceMoved = 0;
            foreach (bool a in beenAt)
                distanceMoved += a ? 1 : 0;
            //Store distance
            AntStore.distances.Add(distanceMoved);

            //Pheromones/distance * how much better it is than average
            double phermoneDeposit = (AntStore.PhermonePerAnt /
(double)distanceMoved);
            //phermoneDeposit = Math.Pow(phermoneDeposit, 2);
            if (distanceMoved < AntStore.shortest)//Store best
            {
                AntStore.bestPath = Path;
                AntStore.shortest = distanceMoved;

                //Some console info
                Console.WriteLine("\nBest dist: {0} ant no {1} dropped {2}
pheromones", distanceMoved, ID, phermoneDeposit);
            }
            else
            {
                //Some console info
                Console.Write("\rAVG dist: {0} @ ant {1}",
Math.Round(AntStore.distances.Average(), 1), ID);
            }
            Store.ResultListbuff.Add(new ResultListItem() { Ant = ID, Distance =
distanceMoved });
            if (distanceMoved <= AntStore.distances.Average()){
                AntStore.ReducePhermones(rand);
                AntStore.IncreasePhermones(phermoneDeposit, beenAt);
            }
            else if (discoveryAnt)
            {
                AntStore.IncreasePhermones(phermoneDeposit, beenAt);
            }
        }
        private void Move()
        {
            //Move
            pos[0] += posChange[0];
            pos[1] += posChange[1];
            //Memory
            /*
            if (beenAt[pos[0], pos[1]])
            {
                for (int y = 0; y < Store.Height; y++)
                {
                    for (int x = 0; x < Store.Width; x++)
                    {
                        if (Path[x, y] >= distanceMoved)
                        {
                            Path[x, y] = -1;
                            beenAt[x, y] = false;
                        }
                    }
                }
                distanceMoved = 0;
                for (int y = 0; y < Store.Height; y++)
                {
                    for (int x = 0; x < Store.Width; x++)
                    {
                        if (beenAt[x, y]) distanceMoved++;
                    }
                }
            }*/
            beenAt[pos[0], pos[1]] = true;
            //Path[pos[0], pos[1]] = distanceMoved++;

            //Update Position
            AntStore.antPos[ID] = pos;
        }
        private void getNextDirection()
        {
            //Check for possible directions
            double fermoneSum = getSurroundingSum();

            if (fermoneSum == 0 || AntStore.explorationFactor >= rand.NextDouble())
//first, "locked" or exploratory ants
            {
                posChange[0] = rand.Next(3) - 1;
                posChange[1] = rand.Next(3) - 1;
            }
            else
            {
                //do roulette selection of the connecting nodes
                RouletteSelection(rand.NextDouble(), fermoneSum);
            }

        }
        void RouletteSelection(double stopAt, double degreesInCircle)
        {
            double tmpCounter = 0;
            //Get new movement direction
            for (int y = -1; y <= 1; y++)
            {
                int y1 = y + pos[1];//Micro optimization
                for (int x = -1; x <= 1; x++)
                {
                    int x1 = x + pos[0];//Micro optimization
                    if (x1 < 0 || y1 < 0 || (x == 0 && y == 0) || (x == posChange[0]
&& y == posChange[1]) || x1 >= Store.Width || y1 >= Store.Height) continue;
                    double fermoneXY = AntStore.FermoneMap[x1, y1];
                    if (fermoneXY >= double.Epsilon)
                    {
                        double nextTotalFermoneXY = tmpCounter + fermoneXY;
                        if (tmpCounter / degreesInCircle <= stopAt &&
nextTotalFermoneXY / degreesInCircle > stopAt)
                        {
                            posChange[0] = x;
                            posChange[1] = y;
                            return;
                        }
                        tmpCounter = nextTotalFermoneXY;
                    }
                }
            }
        }
        bool isMovementValid()
        {
            try
            {
                return AntStore.FermoneMap[posChange[0] + pos[0], posChange[1] +
pos[1]] != -1 /* not wall */ && !(posChange[0] == 0 && posChange[1] == 0) /* not
standing still*/;
            }
            catch { }
            return false;
        }
        double getSurroundingSum()
        {
            double result = 0;
            for (int y = -1; y <= 1; y++)
            {
                int y1 = y + pos[1];//Micro optimization
                for (int x = -1; x <= 1; x++)
                {
                    int x1 = x + pos[0];//Micro optimization
                    if (x1 < 0 || y1 < 0 || (x == 0 && y == 0) || (x == posChange[0]
&& y == posChange[1]) || x1 >= Store.Width || y1 >= Store.Height) continue;
                    double fermoneXY = AntStore.FermoneMap[x1, y1];
                    if (fermoneXY >= double.Epsilon)
                        result += fermoneXY;
                }
            }
            return result;
        }
    }
}
```

Ant Code TSP

```
namespace TSM_ACO
{
    public class Ant
    {
        List<Vertice> path = new List<Vertice>();
        List<Node> NodePath = new List<Node>();
        Node At;
        int Name = 0;
        bool Rand;
        public Ant(Node A, int name, bool random = false)
        {
            Rand = random;
            Name = name;
            moveTo(A);
            Walk();
        }
        private void moveTo(Node A)
        {
            At = A;
            NodePath.Add(At);
        }
        private void Walk()
        {
            SS.Reduce();

            while (NodePath.Count <= SS.Nodes.Count())
            {
                Vertice tmpPath = At.getNext(NodePath, At, Rand);
                if (tmpPath == null)
                {
                    break;
                }
                moveTo(tmpPath.getOther(At));
                path.Add(tmpPath);
            }

            double distance = path.Sum(a => a.Distance);
            Console.Write("\rAnt {0} Moved {1}", Name, distance);

            if (distance < SS.ShortestDistances)
            {
                SS.bestAntNo = SS.TopAntCount;
                SS.ShortestDistances = distance;
                SS.bestPath = path;
            }

            SS.Increase(path, SS.Q / distance);
            SS.LastAntPath = path;
            SS.distances += distance + Environment.NewLine;
            SS.TopAntCount++;
        }
    }
}
```

Appendix B

```
namespace TSM_ACO
{
    class Cluster
    {
        public Node node;
        public List<Vertice> connection = new List<Vertice>();
        public Color color;

        public Node[] ParentNodes;
        public Cluster(Node[] nodes)
        {
            ParentNodes = nodes;
            color = ParentNodes[0].getColor();
            int X = 0, Y = 0;
            if (ParentNodes.Length == 1)
            {
                ParentNodes[0].setColor(Colors.Transparent);
                X = ParentNodes[0].X;
                Y = ParentNodes[0].Y;
            }
            else
            {
                ParentNodes[0].setColor(Colors.Transparent);
                ParentNodes[1].setColor(Colors.Transparent);
                X = ParentNodes[0].X - ((ParentNodes[0].X - ParentNodes[1].X) / 2);
                Y = ParentNodes[0].Y - ((ParentNodes[0].Y - ParentNodes[1].Y) / 2);
            }
            node = new Node(X, Y);
            node.setColor(color);
        }
        public bool nodeExist(Node n)
        {
            for (int a = 0; a < ParentNodes.Length; a++)
            {
                if (ParentNodes[a] == n) return true;
            }
            return false;
        }
    }
    class SuperCluster
    {
        List<Vertice> chosenPath = new List<Vertice>();
        List<Vertice> RoadList = new List<Vertice>();
        List<Cluster> Clusters = new List<Cluster>();
        public List<Node> LevelNodes = new List<Node>();
        public SuperCluster child = null;
        public SuperCluster(bool init = false)
        {
            if (init)
            {
                LevelNodes = SS.Nodes.ToList();
                RoadList = SS.RoadList.ToList();
            }
        }
        public void SolveSet()
        {
            SS.Nodes = LevelNodes.ToArray();
            SS.RoadList = RoadList.ToList();
            SS.ShortestDistances = int.MaxValue;
            foreach (Node a in SS.Nodes)
            {
                a.setColor(Colors.Transparent);
            }
            solveAnt();
            if (child != null)
            {
                child.SolveChild(SS.bestPath, SS.Nodes[0], this);
            }
        }
        public void SolveChild(List<Vertice> lastSolution, Node next, SuperCluster
parentCluster)
        {
            SS.RemovalFactor = 0.1d;
            foreach (Vertice a in lastSolution.Where(aa => aa.Phermones.Count > 0))
            {
                for (int b = 0; b < parentCluster.Clusters.Count; b++)
                {
                    if (a.A == parentCluster.Clusters[b].node || a.B ==
parentCluster.Clusters[b].node)
                    {
                        for (int c = b; c < parentCluster.Clusters.Count; c++)
                        {
                            if (a.A == parentCluster.Clusters[c].node || a.B ==
parentCluster.Clusters[c].node)
                            {
                                foreach (Node a1 in
parentCluster.Clusters[b].ParentNodes)
                                {
                                    foreach (Node b1 in
parentCluster.Clusters[c].ParentNodes)
                                    {
                                        foreach (Vertice abV in
a1.Connections.Where(g => g.getOther(a1) == b1 && g.getSum() == 0))
                                        {
                                            foreach (double tF in a.Phermones)
                                            {
                                                abV.AddPhermone(tF);
                                                abV.RemovePhermone();
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
            SS.RemovalFactor = 0.05d;
            Console.WriteLine();
            SS.Nodes = LevelNodes.ToArray();
            SS.RoadList = RoadList.ToList();
            SS.ShortestDistances = int.MaxValue;
            if (SS.Nodes.Count() > 0)
            {
                //Theese two lines decreases ant distance by about 30-35%
                string t = "0,0" + SS.ExplorationFactor.ToString().Split(',')[1];
                SS.ExplorationFactor = double.Parse(t);

                solveAnt();
                if (child != null)
                {
                    child.SolveChild(SS.bestPath.ToList(), SS.Nodes[0], this);
                }
                else
                {
                    Color selectedColor = new Color()
                    {
                        A = 255,
                        B = (byte)(SS.rand().Next(0, 205) + 50),
                        G = (byte)(SS.rand().Next(0, 205) + 50),
                        R = (byte)(SS.rand().Next(0, 205) + 50)
                    };
                }
            }
        }
        private void solveAnt()
        {
            double baseAntCount = 20; //5 = 300 ants, 10=600, 20 = 1200 etc
            SS.tmpInt3++;
            printImage("2");
            for (int a = 0; a < (int)(Math.Round((Math.Pow(SS.tmpInt3, 2) *
    baseAntCount)+0.2,0,MidpointRounding.ToEven)); a++)
            {
                new Ant(SS.Nodes[SS.rand().Next(SS.Nodes.Count())], a);
            }

            printImage("1");
            SS.tmpInt += 20;
            SS.tmpInt2--;

        }
        void printImage(string add = "")
        {
            foreach (Node t in SS.Nodes)
            {
                t.setColor(Colors.Blue);
                foreach (Vertice i in t.Connections)
                {
                    i.setColor(Colors.Red);
                    i.setWidth();
                }
            }
            CreateSaveBitmap(SS.canvas, "D://Level" + SS.tmpInt2 + add+".png");
            foreach (Node t in SS.Nodes)
            {
                t.setColor(Colors.Transparent);
                foreach (Vertice i in t.Connections)
                {
                    i.setColor(Colors.Transparent);
                }
            }
        }
        private void CreateSaveBitmap(Canvas canvas, string filename)
        {
            RenderTargetBitmap renderBitmap = new RenderTargetBitmap(
             (int)canvas.Width, (int)canvas.Height,
             96d, 96d, PixelFormats.Pbgra32);
            // needed otherwise the image output is black
            canvas.Measure(new Size((int)canvas.Width, (int)canvas.Height));
            canvas.Arrange(new Rect(new Size((int)canvas.Width,
(int)canvas.Height)));

            renderBitmap.Render(canvas);

            //JpegBitmapEncoder encoder = new JpegBitmapEncoder();
            PngBitmapEncoder encoder = new PngBitmapEncoder();
            encoder.Frames.Add(BitmapFrame.Create(renderBitmap));

            using (FileStream file = File.Create(filename))
            {
                encoder.Save(file);
            }
        }
        private void Text(double x, double y, string text, Color color)
        {
            TextBlock textBlock = new TextBlock();
            textBlock.Text = text;
            textBlock.Foreground = new SolidColorBrush(color);
            Canvas.SetLeft(textBlock, x);
            Canvas.SetTop(textBlock, y);
            SS.canvas.Children.Add(textBlock);
        }
        public void addCluster(Cluster a)
        {
            Clusters.Add(a);
        }

        public void BuildLevel()
        {
            //Cluster nodes into groups of 2
            List<Node> Nodes = SS.Nodes.ToList();
            while (Nodes.Count > 0)
            {
                Color selectedColor = new Color()
                {
                    A = 255,
                    B = (byte)(SS.rand().Next(0, 205) + 50),
                    G = (byte)(SS.rand().Next(0, 205) + 50),
                    R = (byte)(SS.rand().Next(0, 205) + 50)
                };
                List<Node> tmpCluster = new List<Node>();
                Node baseNode = Nodes[0];
                tmpCluster.Add(baseNode);
                if (true)
                {
                    //Cluster by using K-Nearest
                    Vertice[] v = baseNode.Connections.Where(b =>
Nodes.Contains(b.getOther(baseNode))).OrderBy(b => b.Distance).Take(1).ToArray();
                    foreach (Vertice b in v)
                    {
                        tmpCluster.Add(b.getOther(baseNode));
                    }
                }
                else
                {
                    for (int a = 0; a < 3; a++)
                    {
                        Vertice[] tV = baseNode.Connections.Where(b =>
Nodes.Contains(b.getOther(baseNode))).ToArray();
                        if (tV.Length > 0)
                        {
                            Vertice v = tV[SS.rand().Next(tV.Length)];
                            while (tmpCluster.Contains(v.getOther(baseNode)))
                            {
                                v = tV[SS.rand().Next(tV.Length)];
                            }
                            tmpCluster.Add(v.getOther(baseNode));
                        }
                    }
                }
                foreach (Node b in tmpCluster)
                {
                    Nodes.Remove(b);
                    b.setColor(selectedColor);
                }
                addCluster(new Cluster(tmpCluster.ToArray()));
            }

            //Init cluster values
            int c = 0;
            SS.RoadList = new List<Vertice>();
            for (int a = 0; a < Clusters.Count; a++)
            {
                for (int b = a + 1; b < Clusters.Count; b++)
                {
                    double distance = double.MaxValue;
                    foreach (Node n1 in Clusters[a].ParentNodes)
                    {
                        foreach (Node n2 in Clusters[b].ParentNodes)
                        {
                            double tmpDist = n1.getRoad(n2).Distance;
                            if (tmpDist < distance)
                            {
                                distance = tmpDist;
                            }
                        }
                    }

                    Vertice vert = new Vertice(Clusters[a].node, Clusters[b].node,
distance);
                    //vert.setColor(Clusters[a].color);
                    Clusters[a].connection.Add(vert);
                    Clusters[b].connection.Add(vert);
                    RoadList.Add(vert);
                    c++;
                }
            }
            Console.WriteLine("Reduced to {0} connections", c);
            for (int a = 0; a < Clusters.Count; a++)
            {
                LevelNodes.Add(Clusters[a].node);
                Clusters[a].node.setVertice(Clusters[a].connection.ToArray());
            }
            SS.Nodes = LevelNodes.ToArray();
            SS.RoadList = RoadList.ToList();
        }
    }
}
```