# Visualising Ant Colonoy Optimisation

## Design

**Author:** Christopher Edwards

che16@aber.ac.uk

**Supervisor:** Dr. Neil Mac Parthalin

ncm@aber.ac.uk

An overview of design and design rationale for a Computer Science
Major Project

Department of Computer Science
Aberystwyth University
Wales
March 27, 2015

# 1  Language

The choice of implementation language for both the graphical user interface and the Ant Colony Optimisation algorithm is extremely important. The nature of the project suggests that an Object Orientated approach would best suit as the implementation language. One of the main reasons for this is because the manipulation and use of Objects in such languages allows for Object-based decomposition allowing for more logical modules system wide when compared to a non-Object Orientated approach. A non-Object Orientated approach may be more subject to functional decomposition (the application is split into modules grouping similar functions rather than representing separate Objects).

An Object Orientated approach has been identified as most suitable. Therefore there are two main languages which are at the forefront of the selection process. These two languages are C# (*C Sharp*) and Java. Both languages have the potential to achieve a high level of success when applied to the projects problem however; they both have different consequences depending on the environment and application in which they are used. C++ (*C plus plus*) is another popular Object-Orientated language. C++ has been discounted due to lack of language experience therefore using a more familiar language such as the two specified above (Java and C#) is far more appropriate in this instance.

One of the major factors in deciding on the implementation language is the suitability of the languages features when applied to the projects problem including any external resources or compatible libraries. Both Java and C# are very similar at an abstract level in terms of provided features by default. Both include everything that would be necessary to implement the proposed design for this application. Both languages provide the ability for Objective decomposition and allow for polymorphic behaviours (multiple entities of different types using the same interface) which enables effective use of inheritance to allow multiple Agent variations (Ants in this case) to be supported easily.

C# has been created and is continually being developed by Microsoft and is focused around the .NET framework which is also a product of Microsoft. As C# is heavily Microsoft orientated its cross-platform capabilities are significantly reduced. The .NET framework(s) have only recently been open sourced so they lack full support on all platforms reducing the applications cross-platform reliability. The project is being developed with a focus on educational value; therefore cross-platform reliability is very important as maximisation of potential consumers is important (more people using the application implies more people are learning). A standard build will not be specified or assumed so there must be necessary measures in place to accommodate as many environments as possible. C# is heavily coupled with Windows based systems therefore; If C# were to be used there is a risk of alienating Macintosh and UNIX users. There are attempts to port .NET to other architectures (for example, Mono[3]) but the implementations of such approaches are not exact replicas of Microsoft .NET framework(s), therefore they cannot be relied upon. The use of Java would eliminate the cross-platform support issues as Java applications execute inside the Java Runtime Environment (JRE) which is available across most platforms and behaviours can be accurately modelled and predicted in the vast majority of cases.

Little differs between Java and C# in terms of feature presence (abstractly, how each language achieves each task is very different) thus, Java will be used for this project. The cross-platform constraints that come with the use of C# are not balanced by any necessary exclusive key features. As a result Java is the most appropriate language for the project, this all but ensures cross-platform reliability without sacrificing any important libraries or features.

# 2  Architecure

There have been considerations as to what key elements will be present in the composure of a suitable underlying architecture for the application. The architecture must accommodate both major elements of the application (graphical user interface and the Ant Colony Optimisation algorithm) in a manner that enables the best possible expansion/modification opportunities to accommodate any additional features or unforeseen changes. Selecting relevant Design Patterns will enable the above goals to become reality however; design patterns should be respectfully and

must represent a general solution to a problem. The Overuse or misuse of such patterns can cause significant complexity issues through the system, this needs to be avoided.

## 2.1 Design and Architectural Patterns

### 2.1.1 Model-View-Controller

The Model-View-Controller (MVC) Architectural Pattern is designed to reduced coupling between system components, these are represented here as the user interface(s) (View) and the underlying data and its representation(s) (Model). The interaction(s) between the View and the Model "established using a subscribe and notify protocol"[1] (Controller). The Controller updates the View(s) based on the model(s) current state or vice-versa however; The View(s) cannot directly communicate with the Model, the Controller must govern such interactions.

This project will make effective use of Model-View-Controller in order to produce an environment which is much easier to maintain and has little coupling between the Model and the View(s). This allows the Model(s) and/or View(s) to be substituted or modified in order allowing different representations of the current algorithm, or in fact different algorithms altogether.
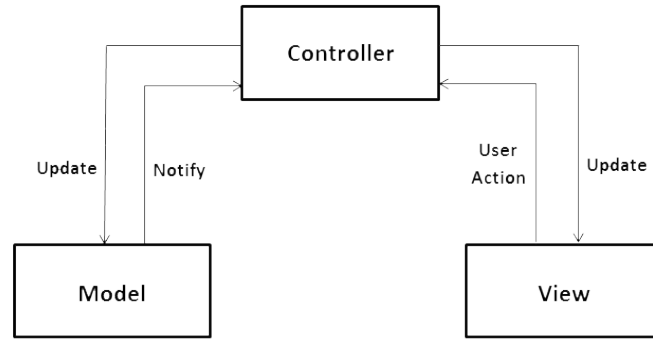


Figure 1: Basic abstract overview of the Model-View-Controller pattern

- **Model** represents the underlying Ant Colony Optimisation algorithm. The Model also contains the required arithmetic functions and any additional operations required to execute the algorithm correctly.

- **View** represents the graphical user interface which will not only display the algorithms execution, but also enable the user(s) to modify the algorithms parameters.

- **Controller** represents the Observer required to enable interactions between the Model(s) and View(s).

### 2.1.2 Singleton

In order to maintain simplicity throughout the application the Singleton design pattern will implemented for key Objects where one and only one instance of an Object must exist. The Singleton pattern prevents multiple instantiations of specific Object(s) as the Object itself is solely responsible for tracking the currently instantiated instance of itself[2].

The application will consist of a graphical user interface which in turn, will be composed of several different components. Such components must only be instantiated once in order to ensure correct interactions are performed. Without the presence of the Singleton pattern there exists the possibility of multiple instances of such components which could potentially cause unforeseen complications and undefined behaviours.
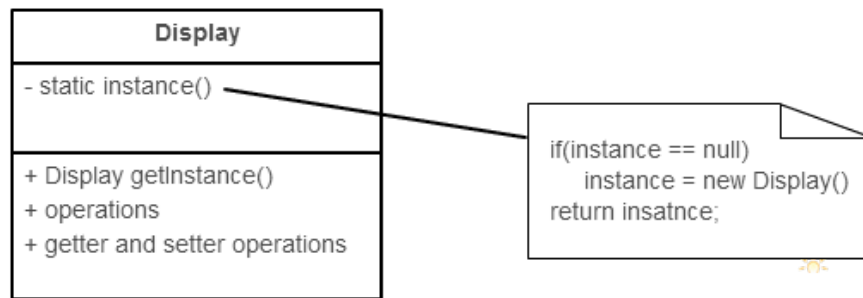
Figure 2: Proposed high-level implementation of the Singleton pattern demonstrating how the sole instance of the graphical user interface will be tracked.

## 2.2 Structure

Adhering to the concepts of the patterns described in sections 2.1.1 and 2.1.2 the following Class Diagram shows proposed application structure. This is not concrete and could potentially change throughout development, the Class Diagram in question is represented below by figure 3 and is represented using standard UML notation.
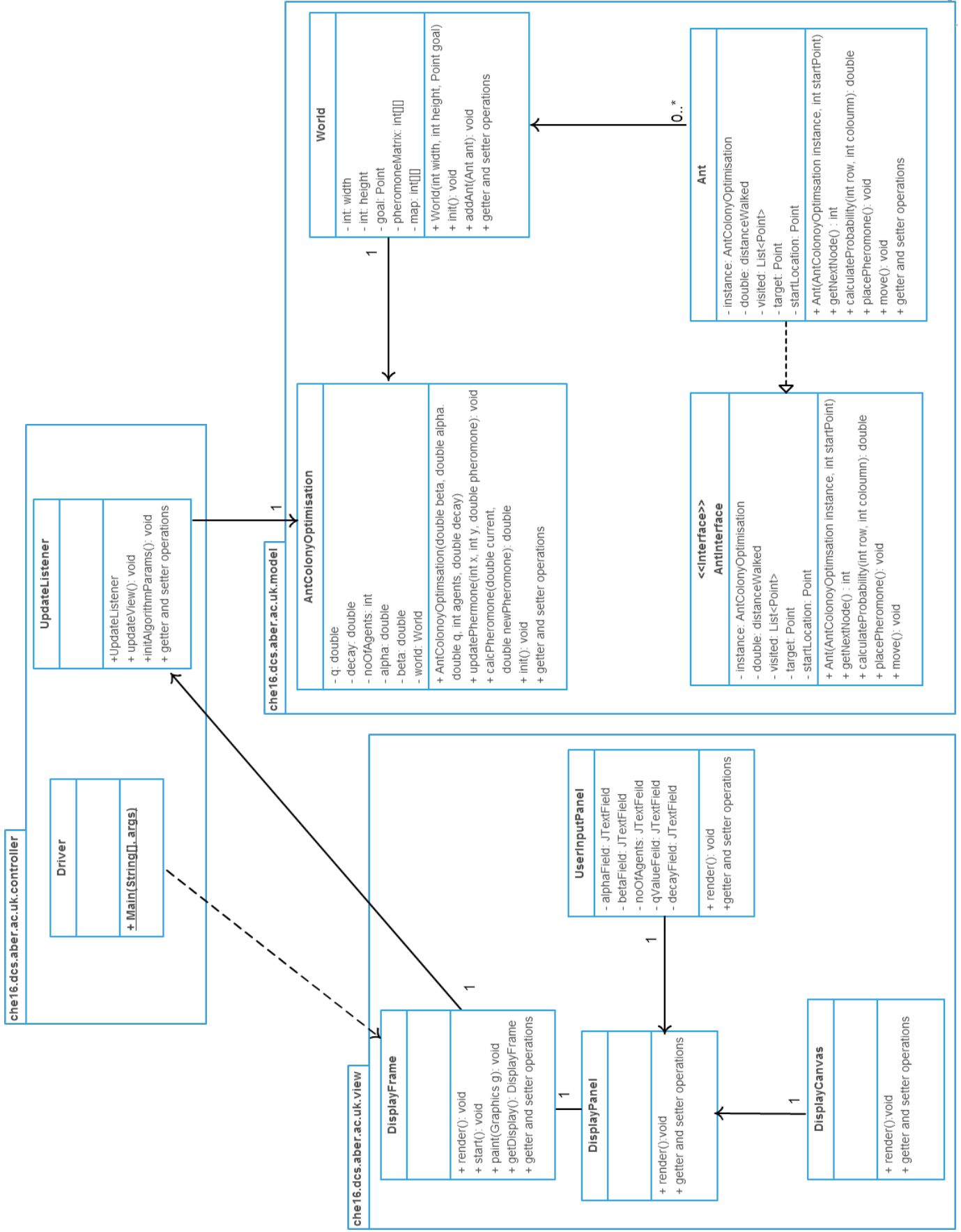
Figure 3: Initial Class Diagram representing the main modules, packages and system interactions using UML notation.

### 2.2.1 View Package

The *che*16.*dcs.aber.ac.uk.view* package contains the graphical user interface components. The main concept behind this is the use of nested components such as JPanel, JTextFields and the like. These will be contained inside top level JFrame. This allows modification of each component in isolation without impacting the other components behaviours and/or elements. This is the first module of the application which is initialised, and is done so directly from the *main* method. This package has no direct knowledge of any package members mentioned in 2.2.2. Instead the *che*16.*dcs.aber.ac.uk.controller* package members handle the communication(s) between the model(s) and view(s), adhering to the principles discussed in section 2.1.1.

### 2.2.2 Model Package

The che16.dcs.aber.ac.uk.model package contains the necessary elements and attributes required in order to accurately represent the algorithms state(s) and ensure correct algorithm execution. This package has no directl knowledge of any package members mentioned in 2.2.1. Instead the controller package members handle the communication(s) between these the model(s) and view(s), adhering to the principles discussed in section 2.1.1.

### 2.2.3 Controller Package

The initial concept for the controller is basic and will grow in complexity during the development lifecycle as more and more control based mechanisms will become prevalent. This initial concept is a simple Observer which notifies the view(s) should the model change in a significant way; the inverse of this is also true.

## 3 Interface Design

The interface design must accommodate all the necessary elements required to allow user defined values for each of the algorithm parameters. The interface must also be able to visually represent the current algorithms state including the locations of the agents and nodes without impacting on the usability of the application (the interface must not freeze or be negatively affected by the algorithms execution).

The interface will use a very neutral colour scheme which will maximise the usability and reduce the risk of complications which may arise from users being subject to difficulties understanding or identifying certain colours. Every option for the user will be textually defined and will not rely on any sound or visual prompts in order to user. As a result in addition to the above, users with visual impairments will still be able to use the application as intended.
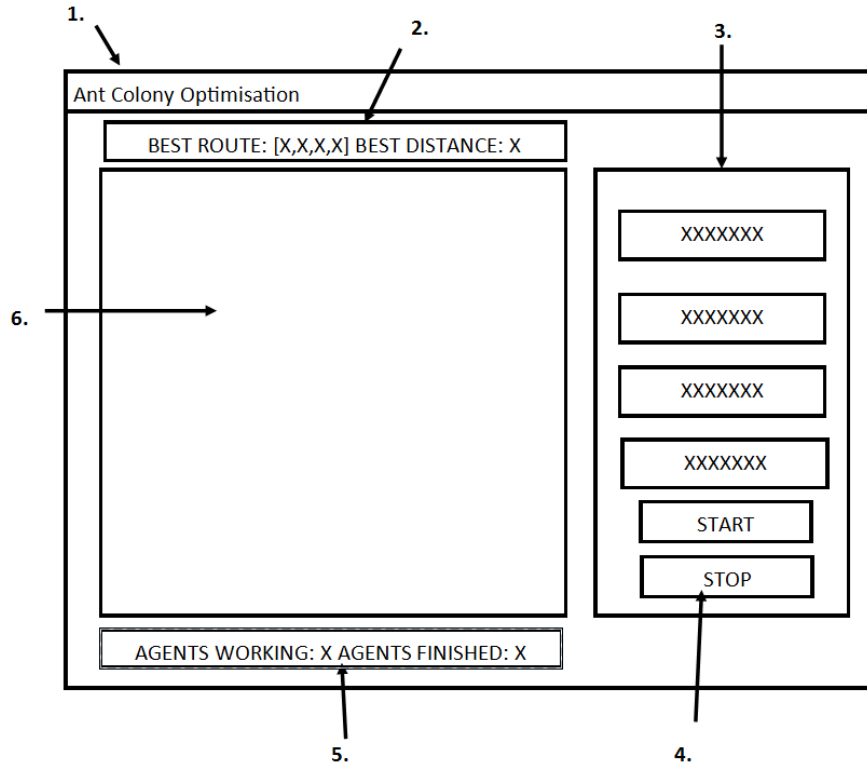
Figure 4: Proposed design for the graphical user interface, showing the key elements in their planned locations.

**1.** in figure 4 refers to the containing frame which will house the reset of the interface elements. This frame will not be resizable preventing complications such as the dynamic resizing of the interface having an impact on the observable range of the world.

**2.** in figure 4 refers to a text view containing information relevant to the algorithms current state of execution. This text view will contain the current best distance as well as the order of nodes visited to traverse the current best path. This will only be populated during the algorithms execution and only if the best path has been initialised.

**3.** in figure 4 refers to container which houses the interactive elements relating to the modification of the algorithms parameters. This will consist of several labels and text fields which can be modified informing the user of what they will be modifying as well as providing suitable error messages if the users enters an incorrect value for any of the parameters.

**4.** in figure 4 refers to the start and stop buttons. These are here to conform to the expectation that a user will expect some clear way to start and stop the application at their free choice, this is the simplest way to do this, and requires no hidden menus or hidden key bindings.

**5.** in figure 4 refers to a text view containing information relevant to the algorithms current state of execution. This text view will contain the number of agents currently working (which means the number of agents who haven't met their own stop conditions) as well as displaying the number of agents which are finished.

**6.**in figure 4 refers to the main canvas area which will display the algorithms current state of execution to the user. The contents of this will reflect the user defined values (elements in: **3.** in figure 4) as well as representing each agents current location, their movements between nodes and also the modelling of pheromone deposit and decay will be present in this canvas.

# 4   Algorithm

There are several adaptations of the Ant Colony Optimisation algorithm. Initially the project contain an implementation of Ant Colony Optimisation in its simplest form, without the presence

of any enhancements such as using *Elitist Agents* or similar. Once a working implementation is in place the next step will be to adapt the algorithm in various ways to further aid the teaching potential of the project.

## 4.1 How does it work?

The general premise is that each Agent (Ant) embarks and a pseudo random walk through the state space. The Agents movements are influenced by pheromone deposits placed on edges between vertices. This pheromone is deposited by other Agents in accordance with the equations stated in section 4.3.2. The Agent's next move is influenced by the result of the equation stated in section 4.3.1. However; there is still the probability of the Agent moving to a less attractive point so the Agent does not always travel to the strongest pheromone concentration.

Overtime the pheromone deposit concentration on $edge_{xy}$ is directly proportional to the quality of the candidate solution, and ultimately the ants will converge to find the shorted route between two or more points.

There are several algorithm requirements;

- **Suitable problem representation** the World and Agents must be represented in a suitable and logical manner allowing the algorithm to execute as expected.

- **Pheromone manipulation metrics** there must exists adequate ways to access the pheromone matrix as well as manipulate (deposit/remove) the concentration of pheromone on a given edge in order to model decay and deposits.

- **Probabilistic movement functions** there must exist functions that calculate the probability of the Agent moving to a specific vertex. This is based on the pheromone concentration and the Agents location (see section 4.3.1).

## 4.2 Pseudo-code

---
**Algorithm 1** Pseudo-code for Ant Colony Optimisation
---
1:  Initiate World and algorithm parameters
2:  Initiate *pheromone* values
3:  Initiate best distance to $\infty$
4:  **while** !stop condition **do**
5:      **for all** Agents **do**
6:          select a random starting vertex on the graph
7:          **while** !at completed solution **do**
8:              Calculate next move using probabilistic function
9:              Add moved point to Agent's memory
10:             Calculate and deposit pheromone on the path
11:         **end while**
12:     **end for**
13: **if** *local best solution < global best solution* **then**
14:     *globalbest = localbest*
15: **end if**
16: **end while**
17: output **global best** solution
---

Above is a somewhat simplified pseudo-code representation of the proposed Ant Colony Optimisation algorithm. The mathematical formulae required to achieve steps *7* and *10* are shown in section 4.3. The final algorithm may differ from the above depending on any additional feature present in the final release.

## 4.3 Metrics

### 4.3.1 Probabilistic function

$$p^k_{xy} = \frac{(\tau^\alpha_{xy})(\eta^\beta_{xy})}{\sum(\tau^\alpha_{xy})(\eta^\beta_{xy})} \tag{1}$$

Figure 5: Algebraic model of the probabilistic function used to calculate next move for any Agent [7]

The probability that an Agent moves to vertex $xy$ is described using the above. $p$ is the probability for any Agent $k$ to move through vertex $xy$. $\tau$ is the amount of pheromone deposited on vertex $xy$, which is raised to the exponent $\alpha$. $\alpha$ is an heuristic value representing how greedy the algorithm is in its path finding [5]. The result for $\tau^\alpha_{xy}$ is then multiplied by the edge $xy$'s evaluation($\eta$). Generally $\eta$ will be represented using $\frac{1}{Euclidean\ distance_{xy}}$[5][4]. This will then be raised to the exponent $\beta$ which like $\alpha$ is an heuristic parameter however $\beta$ describes the Agents path finding speed.$\sum(\tau^\alpha_{xy})(\eta^\beta_{xy})$ is the sum of all possible solutions.

---
**Algorithm 2** Pseudo-code for Probabilistic function - Each Agent - figure 5
---
1: read the *pheromone* level for vertex $xy$
2: raise the value from 1: to the exponent $\alpha$
3: Multiply the result of 2: by $(inverted\ distance_{xy})^\beta$
4: initiate a temporary double *cloumnTotal*
5: **for all** visted vertex **do**
6:     $columnToal$ += (read the *pheromone* level for vertex $xy)^\alpha \times (inverted\ distance_{xy})^\beta$
7: **end for**
8: divide the result of $3:$ by the result of $5:-7:$
---

### 4.3.2 Pheromone deposit

$$p^k_{xy} = (1-\rho)\tau^k_{xy} + \Delta\tau^k_{xy} \tag{2}$$

Figure 6: Algebraic model of the pheromone deposit function used to calculate the correct values for the pheromone matrix [8]

$\tau$ represents the pheromone deposit for an edge $xy$ by Agent $k$ [5]. $\rho$ is a value between $0-1$ which represents the decay rate *decay*. $1-\rho$ is multiplied by the existing amount of pheromone at $edge_{xy}$ to correctly account for decaying trails. The new amount of pheromone is then added using the equation from figure 7.

$$\Delta\tau^k_{xy} = \begin{cases} Q/L_k & \text{if Agent } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Figure 7: Algebraic model of the function used to calculate the how much new pheromone is deposited at $xy$ [6]. Pheromone is only updated if the Agent $k$ has visited point $xy$.

figure 7 represents the new amount of pheromone to be added to the existing concentration at $edge_{xy}$. This can be read as change in $\tau$ ($\Delta\tau$). Q is simply another heuristic parameter which is

divided by the distance $agentk$ travelled to get to $edge_{xy}$. If the result of this is $\leq 0$ return 0. This ensures that new *pheromone* is only added to the existing concentration at $edge_{xy}$ is used by $agentk$ in its tour.

---

**Algorithm 3** Pseudo-code for Pheromone function - figures 6, 7

---

1: **if** $pheromoneDeposit = Q\ value/totalDistanceWalked$ ¡ 0 **then**
2:     $pheromoneDeposit = 0$
3: $pheromone_{xy} = (1 - algorithm\ decay\ rate) \times currentPheromone_{xy} + pheromoneDeposit$
4: **if** $pheromone_{xy} \geq 0$ **then**
5:     $pheromoneMatrix_{xy} = pheromone_{xy}$
6: **else**
7:     $pheromoneMatrix_{xy} = 0$
8: **end if**

---

# 5 Requirements

The application must at least be able to effective perform these operations.

- **Modify algorithm parameters** as this in an educational project they user must be able to experiment with different values and see their results reflected accordingly. This will be met using the elements as described in figure 4 point **3.**

- **Display the Agents in the world** the users must be able to see the agents in the canvas otherwise they would not be able to understand or view the algorithms execution. This will be met at a high level using the canvas described in figure 4 point **6.**

- **Display the Nodes in the world** the users must be able to see the Nodes in the canvas otherwise they would not be able to understand or view the algorithms execution. This will be met at a high level using the canvas described in figure 4 point **6.**

- **Display the Agents at their current Nodes** the users must be able to see the agents at their current location (Node) in order to see how the agents are moving between nodes depending on the pheromone values and parameters they specified. This will be met at a high level using the canvas described in figure 4 point **6.**

- **Display the Agents moving between Nodes** The users must be able to see the agents at their current location (Node) in order to see how the agents are moving between nodes depending on the pheromone values and parameters they specified. To show the ants movement Linear Interpolation will be used. This will be met at a high level using the canvas described in figure 4 point **6.**

- **Display the pheromone deposit and decay operations** the users must be able to see how the wolds pheromone changes over the course of the algorithms execution. This will be a result of the paremters they specified. This will be met at a high level using the canvas described in figure 4 point **6.**

- **User can start or stop the algorithm whenever** the user will be expecting a clear way to start and stop the algorithms execution. There will be a simple way to achieve this using two buttons for start and stop respectively as described in figure 4 point **4.**

- **Display the best route to the user** the users must be able to see how current best found route develops over time. This will also show the user how the optimal paths differ depending on the parameters they specified. This will be met at a high level using the canvas described in figure 4 point **6.**

# Bibliography

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, Nov 1994. pp. 4-6, [Online]. Available: http://www.worldcat.org/isbn/020163361.

> This book is a very good reference point for design patterns in general, providing concise descriptions, examples and use cases. The book examples are implemented using C++, as this will be a Java based project the main concepts are still relevant as the languages aren't too diss-similar.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, Nov 1994. pp. 127-128, [Online]. Available: http://www.worldcat.org/isbn/020163361.

> This book is a very good reference point for design patterns in general, providing concise descriptions, examples and use cases. The book examples are implemented using C++, as this will be a Java based project the main concepts are still relevant as the languages aren't too diss-similar.

[3] Mono. Getting Started — Mono , [online] Feb 2015. Avaliable: http://www.mono-project.com/docs/getting-started/. Accessed: 13 Feb 2015.

> Mono provide an open source implementation of the Microsoft .NET framework enabling the development of C Sharp across multiple platforms and environments. The Framework is based on ECMA standards for C Sharp so there is some reliability when developing C Sharp in different environments. However, based on research the implementation of certain features, specifically the newer features in the latest releases of the .NET framework do not always behave as expected. This causes development problems which can easily be avoided for this project.

[4] V. E. Sjoerd. "Dynamic ant colony optimization for the traveling salesman problem. Master's thesis, Leiden University, The Netherlands, Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands, Jul 2012. pp.7 [Online]. Available: http://www.liacs.nl/assets/2012-08SjoerdvanEgmond.pdf.

[5] Thomas Jungblut. Ant Colony Optimization for TSP Problems , [online] Feb 2015. Avaliable: http://codingwiththomas.blogspot.co.uk/2011/08/ant-colony-optimization-for-tsp.html, Aug 2011. Accessed: 14 Feb 2015.

> Simplistic explantions of each function required. Break down of equations is also included in order to make them more digestable.

[6] Wikipedia. Ant Colony Optimisation New Pheromone Function , [online] Feb 2015. Avaliable: http://upload.wikimedia.org/math/6/d/b/6db065218c956a4a7af6da99aaeca5d1.png. Accessed: 14 Feb 2015.

> Simple graphic representing the algera representation of the pheromone fucntion required to update the pheromone matrix.

[7] Wikipedia. Ant Colony Optimisation New Probability Function , [online] Feb 2015. Avaliable: http://upload.wikimedia.org/math/6/d/b/6db065218c956a4a7af6da99aaeca5d1.png. Accessed: 14 Feb 2015.

> Simple graphic representing the algera representation of the pheromone fucntion required to select the Agents next movements.

[8] Wikipedia. Ant Colony Optimisation Pheromone Function , [online] Feb 2015. Avaliable: http://upload.wikimedia.org/math/e/1/3/e1320f5f72b21e5766dfa7e29b536883.png. Accessed: 14 Feb 2015.

Simple graphic representing the algera representation of the pheromone fucntion required to update the pheromone matrix.