An Object-Oriented Framework with Corresponding Graphical User Interface for Developing Ant Colony Optimization Based Algorithms

RAKA JOVANOVIC
Institute of Physics
Belgrade
Pregrevica 118, Zemun
SERBIA
rakabog@yahoo.com

MILAN TUBA
Faculty of Mathematics
University of Belgrade
Studentski trg 16
SERBIA
tubamilan@ptt.rs

DANA SIMIAN
Department of Computer Science
Lucian Blaga University of Sibiu
5-7 dr. I. Ratiu str.
ROMANIA
d_simian@yahoo.com

Abstract: - This paper describes GRAF-ANT (Graphical Framework for Ant Colony Optimization), an object-oriented C# framework for developing ant colony systems that we have developed. While developing this framework, abstractions that are necessary for ant colony optimization algorithms were analyzed, as well as the features that their implementing classes should have. During creation of these classes, several problems were solved: implementation of individual ants and ant colonies, connection between visualization and problem spaces, creation of a multithread application in which multiple ant colonies can communicate, creation of a problem independent graphical user interface (GUI), establishing an opportunity for hybridization of ACO (Ant colony optimization). Effects of this hybridization to different variations of ant colony systems is analyzed. The use of the GRAF-ANT and its suitability is illustrated by few instances of the Traveling Salesman Problem (TSP). We also present a concept of escaping ACO stagnation in local optima, named suspicious path destruction, that is also a part of GRAF-ANT.

Key-Words: Ant colony system, Evolutionary computing, Combinatorial Optimization, Swarm Intelligence

1 Introduction

A large number of problems necessary to be solved by industry and business are in their source combinatorial. Problems like truck routing, facility positioning, production scheduling, fall into this group, and in some cases, they are even NP-complete. There are no known algorithms for finding the optimal solution of NP-complete problems in polynomial time [1]. When solving these problems in real life situations, like production planning, it is not necessary to have the best overall solution, but a near optimum is adequate in many cases. A large number of different methods for finding near optimal solutions exist, like the use of simple heuristics, greedy algorithms, a Monte Carlo approach, up to more complex local search methods like Tabu Search [2], and Simulated Annealing (SA) [3]. Another concept is using population based metaheuristic like Genetic algorithms (GA) [4], [5], [6] or swarm algorithms for solving this type of problems.

Swarms are collective systems capable of accomplishing complex tasks in dynamic system without any external guidance. The observation of ant colony behavior has inspired a metaheuristic method called the Ant Colony Optimization (ACO) [7], which has in some cases given better results than GA and SA [8], especially in dynamic systems.

A large number of different problems have been solved by using ACO, and in most of the cases new specialized software was created for this task. Some software implementations of this group like ACOTSP, ANTNET, GUIAnt-Miner with their source code are available under a General Public License (GPL). They can be freely downloaded at the IRIDIA research group website in the section for ACO public software at the following address http://iridia.ulb.ac.be/~mdorigo/ACO/aco-code/public-software.html. They can be used as guidelines for creating new specialized software. Their source code can even be modified for new problems, but in most cases, it is easier to create completely new software. ANT-C is an implementation of ACO developed in the C programming language [9] that could be used as a base for new applications, but has a significant drawback of not being object oriented. It is well known that an object-oriented approach is a good methodology for creating reusable code and modular software when solving a group of similar problems. Ugo Chirico has created an object-oriented framework for solving problems with ACO

in Java (Java Framework for Ant Colony Systems JFACS)[10]. but this package has some major disadvantages. There is no available graphical user interface (GUI) and the concept of threading is implemented by attaching different ants to threads instead of the more efficient approach of multiple ant colony systems in which a different colony is simulated in a different thread.

In this paper, we present a new object oriented framework for ant colony systems, GRAF-ANT (Graphical Framework for Ant Colony Optimization) that we developed in C#. It solves some of the existing problems of developing new ACO algorithms. This paper is organized as follows. In Section 2 we show the basics behind the AC algorithm. In Section 3 we give the guidelines for creating abstractions that are needed for this type of framework. In Section 4 we explain a hybridization implemented in GRAF-ANT that is used for escaping from ACO stagnation. In Section 5 we give an example of using our framework for the traveling salesman problem (TSP). In the final Section 6 we show results of using the hybridization from Section 4 on the TSP and do analysis of the effects, combined with different variations of ACO.

2 Ant Colony Optimization

The basic idea of ACO is to imitate the behavior of ants in a colony while gathering food. Each ant starts from the nest and walks toward food. It moves until it reaches an intersection, where it decides which path to take. In the beginning it seems as a random choice, but after some time the majority of ants are using the optimal path (Fig. 1). Another important property of ant colony behavior is the ability of reconnecting a broken line after a sudden appearance of an unexpected obstacle has interrupted the initial path. This is possible because the colony works as a group and not just as individual ants, and the way it is achieved is by using pheromone as a collective memory for the ants in the colony. The pheromone is used in the following way. Each ant deposits pheromone while walking, which marks the route taken. The amount of pheromone indicates the usage of a certain route. Pheromone trail evaporates as time passes. Due to this, a shorter path will have more pheromone because it will have less time to evaporate before new pheromone is deposited. The colony behaves intelligently because each ant chooses paths that has more pheromone.

There are many different ways of converting the presented behavior into a computational system. We

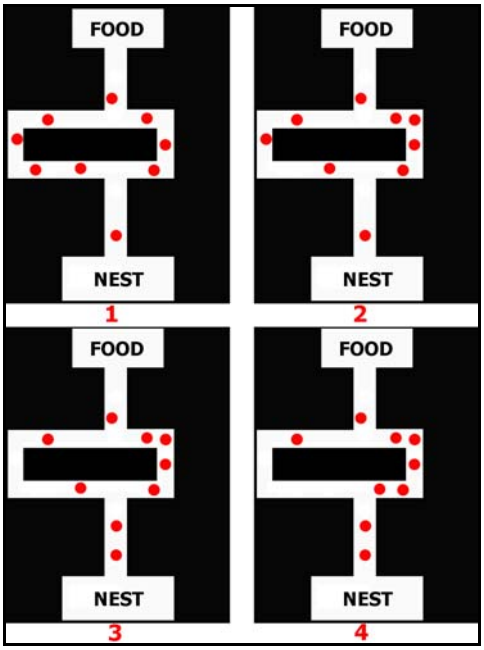

Fig. 1: Ant colony behavior over time

accept the one presented by Marco Dorigo and Luca Maria Gambardella [11], with small modifications

$$s = \begin{cases} \arg\max_{u \notin M_k} \left\{ {\tau_{rs}}^{\alpha} {\eta_{rs}}^{\beta} \right\}, & q \le q_0 \\ S & , q > q_0 \end{cases} \quad (1)$$

$$p_{rs}^{k} = \begin{cases} \dfrac{{\tau_{rs}}^{\alpha} {\eta_{rs}}^{\beta}}{\sum_{u \notin M_k} {\tau_{ru}}^{\alpha} {\eta_{ru}}^{\beta}}, & s \notin M_k \\ 0 & , s \in M_k \end{cases} \quad (2)$$

Equations (1) and (2) describe the probabilistic decision method that an artificial ant *k*, currently at node *r*, after visiting nodes in M_k uses for choosing the next node *s*.

- *q* is a random variable chosen uniformly from [0, 1]
- q_0 is a predefined parameter that gives us a balance between exploitation (use of known good paths, $q<=q_0$) and exploration (search for new paths, $q>q_0$).
- In the case of exploitation, the next node is selected by the highest value of ${\tau_{rs}}^{\alpha}\ {\eta_{rs}}^{\beta}$ where τ_{rs} is the value corresponding to the amount of pheromone deposit on edge connecting *r* and *s*, and η_{rs} is the value of some heuristic for the same edge.

- α, β are predefined parameters that specify the influence of pheromone and heuristic. In the case of exploration, the next node is chosen at random with a probability distribution given by Equation (2), where p_{rs} is the probability of choosing edge *rs*.

The pheromone trail is maintained using two types of updates. Global update is used to reward good paths, or in other words, more pheromone should be deposited on better paths, which is obtained by the following formula

$$\tau_{ij} = (1-e)\tau_{ij} + e\Delta\tau^k \quad , \forall ij \in B^k \qquad (3)$$

B^k is the set of all edges in the path ant *k* used, $\Delta\tau^k$ is the quality of that solution, and *e* is a predefined constant .The local updating is used to avoid creation of a very strong edge used by all ants, and it emulates pheromone evaporation. Every time an ant chooses an edge, it loses some pheromone by the following formula where τ_0 is a predefined constant.

$$\tau_{ij} = (1-e)\tau_{ij} + e\tau_0 \qquad (4)$$

The presented AC heuristic has the following desirable characteristics:

- It is versatile, in that it is easy to be applied to similar versions of the same problem; for example, there is a straightforward extension from the traveling salesman problem (TSP) to the asymmetric traveling salesman problem (ATSP).
- It is robust. It can be applied with minimal changes to other combinatorial optimization problems such as the quadratic assignment problem (QAP) and the job-shop scheduling problem (JSP).
- It is a population-based approach. This is interesting because it allows the exploitation of positive feedback as a search mechanism. It also makes the system amenable to parallel implementations

3 GRAF-ANT Framework Analysis

A framework is a special kind of software library that is similar to an *application program interface* (API) in the class of packages that make possible faster development of applications. Big difference is that, while an API consists of a set of functions that user calls, a framework consists of a hierarchy of abstract classes. The user only defines suitable derived classes that implement the virtual functions of the abstract classes. Frameworks are characterized by using the inverse control mechanism (also known as the Hollywood principle:" Don't call us, we'll call you") for the communication with the user code: the functions of the framework call the user-defined functions and not the other way round. The framework thus provides full control structures for the invariant part of the algorithms and the user only supplies the problem-specific details.

ACO can be applied to a large number of different problems[12][13] [14]. Due to this, the idea of creating a framework that can be used for different types of problems appears naturally, and was exploited a number of times [15] [16].

When developing this type of system, the aspects of creating a useful GUI and the possibility of easy visualization of the problem being solved as well as the progress of the ACO are usually neglected. ACO algorithms are very sensitive to input parameters that define the ant colony behavior [17]. The optimal values are obtained from a large number of tests. This process is more efficient when a good visualization and GUI are accessible. To answer these needs, GRAF-ANT is being developed in C# because of its powerful GUI development tools. In our framework, we have created a base abstract class that implements some basic visualization for graph problems. It is named *RAntVisualiserAbstract* and is inherited when different or more precise visualization is needed for specific problem. This greatly decreases the time of developing ACO applications due to avoiding of redundant programming. The other purpose of this class is to keep problem information that is in not directly associated with the ACO algorithm. An example would be the positions of nodes in the TSP, because for the implementation of ACO we only need the distances between cities. A separate abstract class named *RAntGraphAbstract* is used as a base class for keeping information about problem and calculation variables that are important for ACO like heuristics, pheromone trail, and cost. The connection between visualization and these variables is very strong and is not only graphical presentation of the solution. In some cases it is easier and even necessary to determinate the problem variables from given visualization [18]. Our framework leaves this direction of communication open.

The next important part of the ACO is implementation of individual ant behavior and this task is done through the specification of the class *RAntAbstract*. This class has some methods that

need to be implemented for each specific problem like *GetNextHeuristicStep(), CalcProbability-Distribution()*. It also implements some common needs for these types of classes like local path update, getting a value from a probability distribution, keeping track of a path. In many cases, the path that ants create is not just a simple array of node indices, but a structure with some properties, and constraints. Several different steps of the algorithm implementation use the path created by ants, and it should be possible to visualize a path in the GUI. That is why we created a new abstraction *RAntGraphPathAbstract* that implements some of the common path manipulation methods and is used for communication between different classes in GRAF-ANT.

Class *RAntColonyAbstract* implements standard colony methods like control of the best path found so far, iteration counting, checks for stagnation of the algorithm. The basic ACO system presented in Section 1 has large number of improvements and variations that are used to enhance performance and avoid falling into local optima. Some of them can be implemented independently to a certain level of the problem being solved, and in other cases just some parts of the program code should be edited. In our framework, we have abstracted variations of the basic ACO into class *RAntColonyAbstract*. These variations are explained in detail in article [17] for the TSP problem. GRAF-ANT implements the following

- *Elitist Ant System* in which only the best path is reinforced in ant colony iterations.
- *Elitist Reinforcement Ant System* in which the best path is reinforced with more pheromone.
- *Min-Max Ant System (MMAS)* in which the strength of the pheromone trail is confined to the interval [*TMin*, *TMax*]. *TMin* is calculated at the beginning of the search depending on the problem data. *TMax* is calculated dynamically depending on the best path found so far.
- *Rank Based Ant System* in which the amount of pheromone being deposited by an ant does not only depend on the quality of the solution. The idea is that when all ants have completed their paths to sort them by the quality of their path, and depending on their rank to decide the amount of pheromone they will leave.

Hybridization of ACO algorithms in many cases results in significant improvement of their efficiency [19]. Hybridization usually consists of combining ACO with other methods for solving combinatorial problems mentioned in the introduction. Regardless of the hybridizations being used, the effect on the basic ACO, can be divided into two groups. *RAntColonyAbstract* has the possibility of hybridization in the directions of adding local searches to elevate the quality of paths found, or the possibility of pheromone trail correction when the algorithm has reached stagnation. We have implemented, to our knowledge, a novel concept to pheromone trail correction that we called *suspicious path destruction*. The following section explains this method in detail.

When developing an ACO the idea of parallelization has at first been exploited with the concept of using different processors for calculating the movement of different ants, but this was not the most powerful approach. Similarly to GA, where it is more efficient to have *N* islands of populations of *M* members rather than one island with population of *NM* members, ACO is generally more efficient when it has more small colonies of ants rather than one big colony [20]. Creating an ACO system with a multiply colonies today is especially powerful when multiprocessor machines are becoming a standard and parallel computing is more and more available[21]. Due to that, multithreaded applications are becoming increasingly effective. This means that multiple colonies can be calculated in separate threads. GRAF-ANT implements a multi colony approach through the base class *RantColony-ClusterAbstract*. When working with multiple colony system the communication between the colonies is very important [22]. There is a variety of different approaches used, from what is going to be exchanged between them, to the topology of the communication. We have chosen to use the best-so-far path instead of the whole pheromone trail matrix for communication between colonies. It should be understood that GRAF-ANT is not a high performance application, but a system for developing new ACO. ACO applications are often used in networks for parallel processing and some analysis of their performance in this type of systems is needed. For this reason, we decided to emulate the following network topologies presented in [6].

- *Fully connected* in which the best overall solution has been found and is distributed to all other colonies
- *Replace worst* in which the best solution is only distributed to the worst colony
- *Ring* in which the colony *i* exchanges its best solution with colony *((i+1)%k)* and colony *((i-1) % k)*
- *Parallel independent* runs in which colonies run independently and the best solutions is the best one over all colonies.

Ant colonies with different behavior advance towards the solution of the problem by different speeds and sometimes even in different directions. The behavior of an ant colony is defined by the variant of ACO being used and the calculation parameters, and in different steps of the search different behavior is more desirable. When we have a multiple colony system, some types of colonies can act complementary to each other. For example, ones with high and low exploration levels are complementary. Good combination of different colonies behavior can greatly increase the efficiency of the whole system. In GRAF-ANT the search for this optima has been enhanced with good visualization and an easy way to test different parameter values.

Because of the significant dependence of ACO performance on behavior parameters, it was important to create an effective GUI. To achieve this effectiveness the GUI had to have the following characteristics:

- Easy input and adjustment of parameters in several different groups. These groups are: ant colony decision parameters, ant colony variation and hybridization parameters, problem visualization and characteristics parameters, multi-colony system parameters.
- A possibility of having control over random seeds for better evaluation of the effect of parameter changes.
- Easy approach to each colony in the multi-colony system for observing current state of search progress and changing parameters
- Changing parameter values for several colonies simultaneously in multi-colony systems
- Possibility of viewing the state of pheromone trail
- Real time reaction to parameter changes
- Possibility of comparing visual results of search for different colonies in the multi-colony system
- Adaptability of the GUI for a wide range of problems possibly solved by ACO
- Adding the possibility of enlarging GRAF-ANT GUI with new dialogs created by the developers of ACO Modules.
- Selecting different ACO modules
- The possibility of loading and saving parameter settings
- Saving ant colony search results for further analysis.

In GRAF-ANT we have implemented these features as a part of the GUI. The GUI is connected to a particular problem through the abstract classes mentioned earlier. We choose to develop GRAF-ANT in C# because it was obvious that augmentation of the basic GUI will be needed for specific problems, and C# makes this easy through its property grid component and its relationship properties in classes.

In Fig. 2 and 3 we can view the GUI for TSP. We notice that parameters for colonies and the multi-colony system are located in different tab controls. In each of them groups of parameters are placed in separate tab-pages. On the left side of Fig. 2 we see the best path in red and the current iteration path in blue.

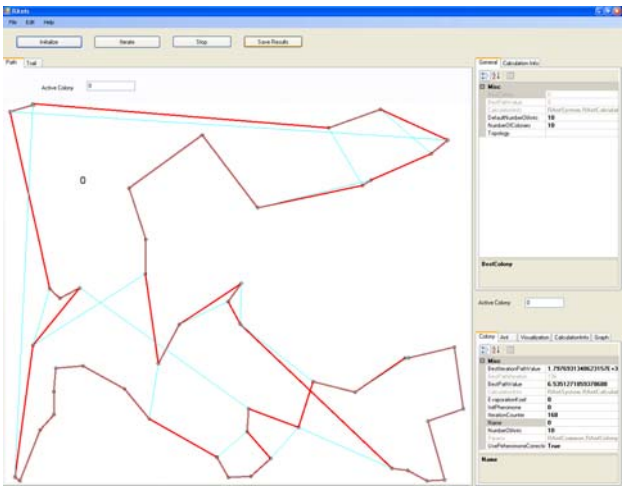

Fig. 2: A screenshot of GRAF-ANT application for the TSP

This screen is animated in real time for observing the progress of the algorithm. Real time animation can be turned of in case that greater speed is needed. The GUI also has basic control over threads with colonies for stopping, pausing, restarting, initia-lizing and saving current results. File operations are positioned in the standard way in the main menu.

Strictly speaking, GUI is not a part of the framework, because it would greatly restrict the possible uses. On the other hand, the mentioned GUI features are sufficient for the majority of users. Even in the case of users for which this type of visualization is not adequate or is superfluous, it will be useful in the early tests of the algorithm being developed. Designing and programming of a GUI takes a large part of development time for an application. That is why it is very useful to use already developed ones.

At the current stage of development the GRAF-ANT framework and its corresponding GUI are a Microsoft Visual Studio 2005 C# project. The project itself consists of a set of files in which the

abstract framework classes are defined together with classes that define the GUI. The project also has as a template an example of the TSP implementation. When creating a new application of an ACO the easiest way is to change the example template.

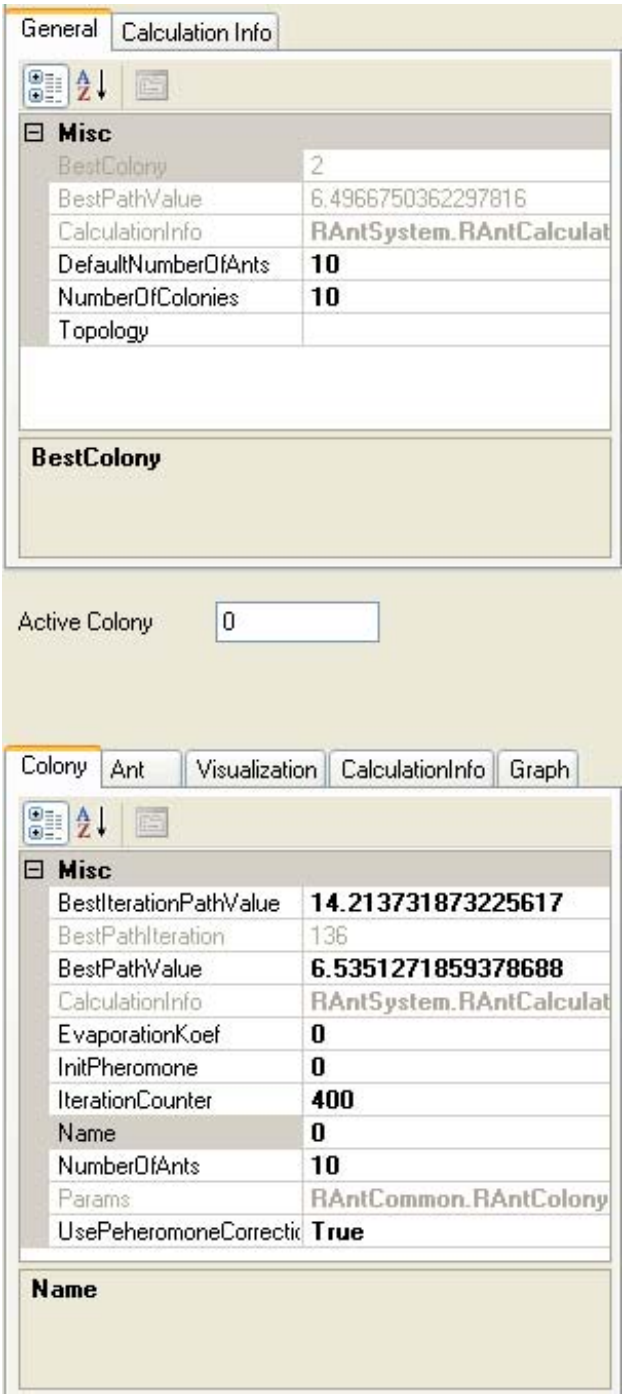

Fig. 3: Part of GRAF-ANT GUI for editing colony parameters and for viewing numeric results

4 Suspicious Path Destruction

In this section we present a concept of ACO hybridization for escaping local optima or best path search stagnation. The idea appeared from observing the progress of ACO on the Traveling Salesman Problem (TSP). When the algorithm was trapped in local optima in many cases it was obvious from visual observation, which corrections should be made, or more precisely, what should not appear in the shortest path. There were two simple criteria: very long edges and intersecting edges are very unlikely to be a part of the best path (Fig. 4).

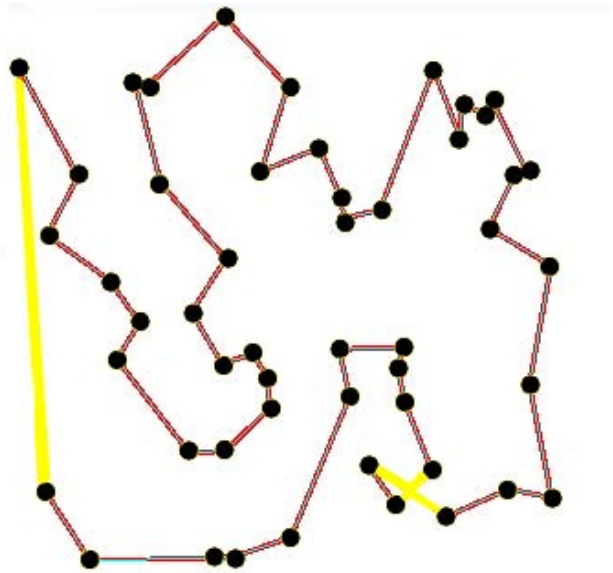

Fig. 4: Example of local optima found with ACO for TSP with suspicious edges on path

The next step was to find a way to, without major corrections to the ACO algorithm, remove them from the ants search path. The solution was to significantly lower the amount of pheromone on randomly selected highly suspicious edges belonging to the best path and to let the colony resume its search. This was a good approach because in the case that a suspicious edge was a part of the optimal path ants would come back to it after testing alternative routes. This hybridization improved the quality and calculation time of ACO for the TSP in most of the tested cases, which will be shown in the following section. A similar concept could be used in other applications of ACO. More formally we need to define a heuristic function *Sus(edge,path)* that indicates the level of undesirability of an edge in a path. Using that function, we create a random variable for selection of edges using the following probability distribution

$$p_{xy} = \frac{Sus(xy, Bp)\, ExSusepect(xy)}{\sum_{xy \in Bp} Sus(xy, Bp)\, ExSusepect(xy)} \quad (5)$$

Bp is the best path, *xy* is an edge. Just using the heuristic function for calculating the distribution was not good enough for a large number of iterations because the same group of edges would be selected repeatedly. That is why we added a correction factor *ExSuspect* that is used to avoid selecting the same edges repeatedly. This correction factor is initially set to 1 for all edges. If an edge *xy* is selected, we update *ExSuspect* as in Equation (6)

$$\begin{aligned} &Pass < 1 \\ &ExSuspect(xy) = ExSuspect(xy) * Pass \end{aligned} \quad (6)$$

In the case that the best path has improved, we also need to update correction values

$$\begin{aligned} &1 < NewEv \\ &\forall A \in Edges \\ &ExSuspect(A) = \min(1, ExSuspect * NewEv) \end{aligned} \quad (7)$$

This correction maintains the starting logic of suspicion because, like in the real world, when a suspect passes interrogation he becomes less likely that he has committed a crime, and in the case of new evidence, suspicion can return. When solving particular problems, there could be a need for adding alterations to distribution from Equation (5). Like in the case of the TSP, a pair of intersecting edges would be considered as one *suspicious edge*. The next step is to select up to *N* edges from the best path for the set *Suspects* and apply the Equation (6). δ is a predefined parameter.

$$\tau_{xy} = \delta \tau_{xy} \quad , \forall xy \in Suspects \quad (8)$$

In the case of the TSP, the appearance of intersecting edges could have been avoided by using a local search or some correction method for newly found paths. This could mislead to the idea that a good local search could be a much better solution for this type of problem in the general case also, using the logic of 'Better safe than sorry'. This is wrong for a number of reasons.

- Correction of suspicious edges in most cases is a more complex problem to be solved than just recognizing them
- When using this approach, ants in fact perform a guided local search with a minimal change to the original code
- Suspicious edges are sometimes a part of the best solution, and totally avoiding them with local search could lead to local optima
- More information is passed to all the ants by adding a new heuristic that analyses the characteristics of the solution independently from the rest of the problem
- The path correction is conducted only when stagnation appears and not for all newly found good paths like it would be done with a correction method used by individual ants.

5 Implementation of the TSP

In this section, we illustrate the use of the GRAF-ANT framework with the TSP problem. Fig. 5 shows an UML model of GRAF-ANT and its expansion with classes for implementing TSP. First, we wish to point out that there is a large number of connections between classes in our framework through composition, aggregation, and dependence due to the complexity of the multi-colony system. The user of the framework is isolated from this complexity because he only needs to specialize certain classes. When creating the TSP module we need to create new inherited classes as seen in Fig. 5. First, we create *RAntVisualiserTSP* from *RantVisualiserAbstract;* in this case we do not need to override function *Visualise* for drawing the problem and *ShowPath* for path found by ants graphical representation.

The base class has implemented drawing nodes at certain positions and drawing lines between them as default. In some cases it is very useful to view the state of the pheromone trail and to do this we override the method *ShowTrail*, This is not an obligatory step, and we did not do it for the TSP. We also create class *RAntGraphTS*P in which we need to implement calculating of cost and distance from *RAntVisualisationTSP;* in our case it will be simple Euclidean distance. This is done by overriding a constructor that uses as a parameter *RAntVisualisationAbstract*. *RAntGraphPathTSP* is the same as its base class because it has no constraints except nodes in the path being unique, a check for more constraints is done in the virtual member function *CheckNewNode()*. *RAntTSP* needs some extra code in the constructor to create properties for graph, path and visualization of class types corresponding to the TSP. The initialization of the search path for each ant is another problem, specific point of ACO. We divided this into two virtual methods: one in the path abstraction *Reset*, and *ResetPath* in the ant abstraction class. In the

case of our problem, we just set the first element of the path to a random city. Classes *RAntColonyTSP*, *RAntColonyClusterTSP* are the same as their base classes, except for overriding methods like *CreateAnts* and *CreateColonies* for creating arrays or array objects of a procreate classes. When we have created all of the classes for our problem, the connection to the GUI needs no extra lines of code. The class *RAntColonyClusterAbstract* links the colonies, parameter groups to the GUI.

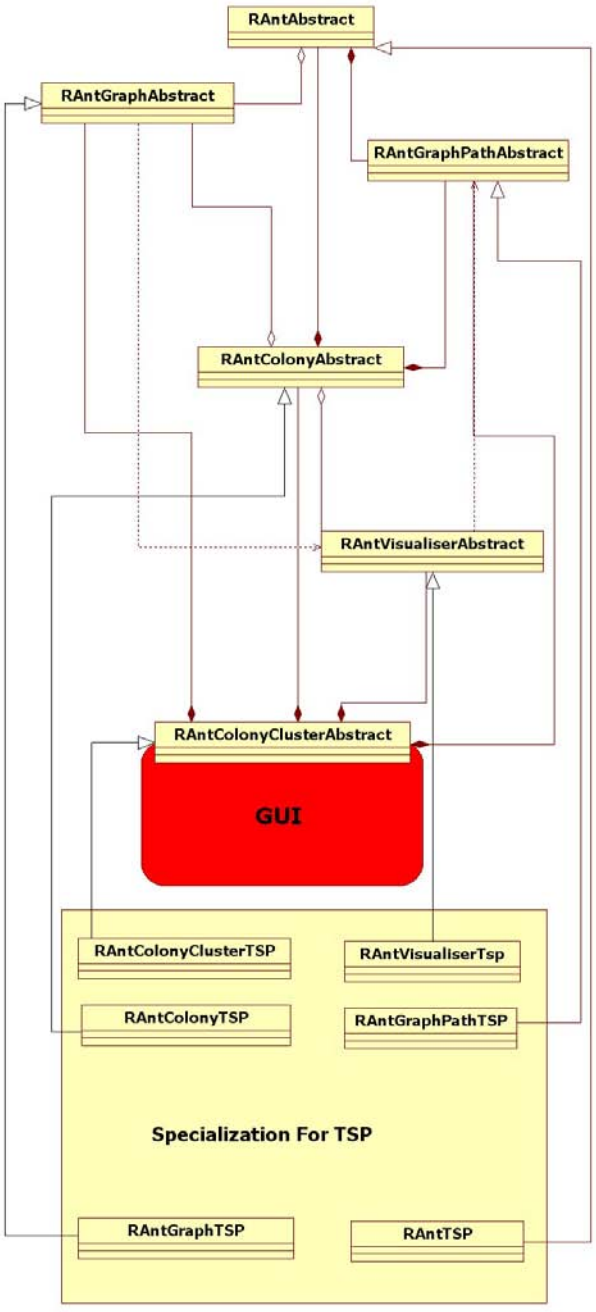

Fig. 5: UML diagram of implementation of TSP

6 Tests and Results

We performed an experimental analysis of the effect of suspicious path destruction (SPD) to different ACO variations. It is important to test the effect of SPD on all the standard variations because none of the variants can be the considered as the most effective. We used our framework to create an application for testing the effects of SPD. To achieve this we needed to implement virtual function *ConstantGraphCorrectection()* in *RabstractAntColonyTSP*. We considered long edges and pairs of interesecting edges as suspicious. We used the SPD through two separate methods, one for long edges and one for intersecting ones. In the case of length suspision, the heuristic function was the rank of the edge in the best path, and in intersection suspision it was a 0,1 depending if intersections occures or not. It is interesting to mention that information needed for calculating intersecting edges (positions of nodes) was inside *Rvisualiser*, which proves ones again that strong connection between visualisation and ACO algorithm is more powerful than it seems at first glance.

In our tests we have used the following parameters for defining the ACO algorithm $q_0 = 0.3$, $\alpha = 2$, $\beta = 0.1$, $e = 0.1$ and τ_0 is calculated as suggested in [5], and the colony had 10 ants. For the Rank based variation of ACO the number of significant ants was 5. The criterion for stagnation was the absence in improvement of the best path for more than 20 colony iterations, and the value of $\delta = 0.01$. For each variation with or without DSP, we preformed 10 different tests and recorded the best path length for all the tests, the iteration on which it was found, and the average path value. We performed tests for the TSP with 50, 100 and 150 cities on randomly generated city positions .

Variation	Best Value	Best Value Iteration	Avg
Basic	6.751	631	6.802
Basic SPD	6.155	1302	6.459
Elitist	6.376	402	6.500
Elitist SPD	6.191	1628	6.413
Elitist Reinforce	6.595	1409	6.670
Elitist R SPD	6.448	519	6.544
Rank Based	6.621	279	6.637
RB SPD	6.256	362	6.445
MMAX	6.307	401	6.573
MMAX SPD	6.448	1302	6.532

Table 1. TSP for 50 cities the maximum possible number of iterations was 2500

We can see from Tables 1, 2 and 3 that, independent from the problem size, in the majority of cases the same variation of ACO would give better results if SPD hybridization was added.

Variation	Best Value	Best Value Iteration	Avg
Basic	9.005	2959	9.136
Basic SPD	8.754	984	9.041
Elitist	8.734	1172	8.908
Elitist SPD	8.656	2751	8.813
Elitist Reinforce	8.764	533	9.029
Elitist R SPD	8.869	2963	8.975
Rank Based	9.154	3322	9.202
RB SPD	8.852	843	8.924
MMAX	8.869	2401	8.942
MMAX SPD	8.718	1709	8.951

Table 2. TSP for 100 cities the maximum possible number of iterations was 3500

The effect is greatest for the basic ACO. It is important to notice that when using SPD, it does not only get a better solution but it also falls into stagnation at a higher number of iterations. SPD could have been improved if more criteria where added, like the case of 4 consecutive nodes in a path forming a non-convex quadrangle, but in our case we just wished to show its effectives, even with simple and easy tests.

Variation	Best Value	Best Value Iteration	Avg
Basic	10.932	1372	11.181
Basic SPD	10.284	360	10.371
Elitist	9.932	742	10.698
Elitist SPD	9.979	3611	10.068
Elitist Reinforce	10.844	959	10.972
Elitist R SPD	10.077	1503	10.364
Rank Based	10.509	3343	10.625
RB SPD	10.113	1624	10.317
MMAX	10.669	2179	10.902
MMAX SPD	10.017	3101	10.451

Table 3. TSP for 150 cities the maximum possible number of iterations was 4000

7 Conclusion

It has been shown that C# is a good choice for creating an ant colony system framework because of its good GUI development tools and the simplicity of creating multi-thread applications. A good GUI is very important when developing and using ACO algorithms. Result quality obtained by using ACO is highly dependent on colony behavior parameters. That is why we have to have an effective an easy way of testing numerous parameters for getting their best possible values and that can be achieved by having a good GUI. We have abstracted different types of ACO variations that are implemented regardless of the problem being solved. In the future we also plan to add the hyper-cube framework concept as a variation. While developing GRAF-ANT we have anticipated the possibility of some ACO hybridization and created classes with which this could be easily done when creating applications for solving particular problems. We considered two types of hybridization: adding local searches and pheromone trail correction in case of colony search stagnation. It has been presented in a large number of articles that a multi-colony approach for ACO greatly increases its efficiency when conducting parallel computing. That is why we added the possibility of experimenting with this type of system to GRAF-ANT. Multi-colony systems are usually connected with network calculations so we decided to emulate several standard network communication methods for testing purposes. A pheromone trail correction method based on the concept of destroying suspicious parts of the best-found path is presented. We used our framework to create an application for the TSP. With it, we tested the SPD method in combination with all standard variations of ACO and it gave good results. When performing these tests the convenience of a powerful GUI, combined with a multi-colony system that allows a large number of simultaneous colonies to run, has greatly simplified the process of retrieving results. A framework that implements all the previously mentioned qualities can greatly decrease the developing time for new ACO algorithms, both by avoiding programming of redundant code and by quick parameter testing which was the goal of GRAF-ANT. At the present stage of development, GRAF-ANT needs full GUI application code to be compiled when using it with the framework, but in the future, we plan

to turn it into a plug-in system to further simplify the creation of new ACO algorithms.

References:

[1] Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman, 1979

[2] L.Vogt,C. A. Poojari,J.E Beasley, A tabu search algorithm for the single vehicle routing allocation problem, Journal of the Operational Research Society, Vol. 58, No. 4, 2007 , pp. 467-480

[3] Kirkpatrick S. and Gelatt C. D. and Vecchi M. P., Optimization by Simulated Annealing, *Science*, Vol 220, No. 4598, 1983, pp. 671-680.

[4] Sancho Salcedo-Sanz, Xin Yao, Assignment of cells to switches in a cellular mobile network using a hybrid Hopfield network-genetic algorithm approach, *Applied Soft Computing*, Vol. 8 , No. 1, 2008, pp 216-224

[5] Der-Horng Lee , Hui Qiu Wang, Lixin Miao, Quay crane scheduling with non-interference constraints in port container terminals, *Transportation Research Part E*, Vol.44 ,No. 1, 2008, pp.124–135

[6] M. Manfrin, M. Birattari, T. St¨utzle, and M. Dorigo, Parallel ant colony optimization for the traveling salesman problem, *Proceedings of ANTS 2006, ser. LNCS, M. Dorigo et al., Eds.*, vol. 4150.Springer Verlag, 2006,, pp. 224–234.

[7] Dorigo M, Maniezzo V: Ant Colony system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics - Part B* Vol. 26, No.1, 1996, pp. 29-41

[8] Kwee Lim, Yew-Soon Ong,Meng Lim, Xianshun Chen, Amit Agarwal,, Hybrid ant colony algorithms for path planning in sparse graphs, *Soft Computing*, Vol. 12, No 10, 2008 , pp. 981-994

[9] Samdani Saurabh Arun, Ant Colony Optimization, http://www.geocities.com/saurabhsamdani/aco.html, Accessed November 2008

[10] Ugo Chirico, A Java Framework for Ant Colony Systems, http://www.ugosweb.com/Documents/jacs.aspx, Accessed November 2008

[11] Dorigo M, Gambardella LM: Ant colonies for the traveling salesman problem. *BioSystems* Vol. 43 No.2 ,1997, pp.73-81.

[12] Vlachos Aristidis, An Ant Colony Optimization (ACO) algorithm solution to Economic Load Dispatch (ELD) problem, *WSEAS Transactions on Systems*, Vol 5, No, 2006, pp. 1763-1771

[13] Bouktir T. and Slimani L., Optimal power flow of the Algerian Electrical Network Using Genetic Algorithms, *WSEAS Transactions on Circuit and Systems*, Vol. 3, No. 6, p.1478-1482, 2004.

[14] Usman Ali, Shahid Khan, Ant Colony System Vector Quantization, Effect on Image Data Hiding, *WSEAS Transactions on Information Science and Applications*, Vol. 1, No 6, 2004, p.1650-1655

[15] Blum, C. Dorigo, M., The hyper-cube framework for ant colony optimization, *Transactions on Systems, Man, and Cybernetics, Part B*, Vol. 34, 2004, No. 2, pp. 1161-1172

[16] A.Brocco, B.Hirsbrunner, M.Courant, Solenopsis: A Framework for the development of Ant Algorithms, *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE* Vo , Issue , 1-5 April 2007 pp. 316 – 323

[17] D. Asmar and A. Elshamli and S. Areibi. A Comparative Assessment of ACO Algorithms Within a TSP Environment, *In 4th International Conference on Engineering Applications and Computational Algorithms*, Guelph, Ontario, Canada, July 2005.

[18] Liu Nan, Huang Bo and Xiaohong Pan, Using the Ant Algorithm to Derive Pareto Fronts for Multiobjective Siting of Emergency Service Facilities, *Journal of the Transportation Research Board*, No. 1935, 2005, pp. 120–129.

[19] Feng Y J, Feng Z R, Ant colony system hybridization with simulated annealing for flow-shop scheduling problems. *WSEAS Transaction on Business and Economics*, Vol. 1, No. 1, 2004, p. 133-138

[20] R. Michels, M. Middendorf, An island model based Ant system with look ahead for the Shortest Super sequence problem, *Parallel Problem Solving from Nature — PPSN V*, Springer, 1998

[21] Laurentiu Rudeanu, Mitica Craus, Parallel Implementation of Ant Colony Optimization for Travelling Salesman Problem, *WSEASs Transactions on Systems*, Vol. 3, No. 3, 2004, pp. 1161 -1166

[22] I. Ellabib, O. Basir and P.H. Calamai, `A multiple Ant Colony System with different communication strategies', *World Scientific and Engineering Academy & Society (WSEAS) Transactions on Systems*, Vol. 6, 2005, pp. 663-670.

Research for this paper is part of the Project 144007, Ministry of Science, Republic of Serbia