Developing an Object-Oriented Framework for Solving Problems Using Ant Colony Optimization

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 a robust Object-Oriented C# framework for developing Ant colony systems that we have developed. Our framework is named GRAF-ANT (Graphical Framework for Ant Colony Optimization) and we present in detail the design guide-lines of developing GRAF-ANT and implemented features. We solved several important problems: implementation of individual ants, or ant colonies, connection between visualization and problem space, leaving the opportunity for hybridization of ACO (Ant Colony Optimization), creation of a multithread application in which multiple ant colonies can communicate, creation of a graphical user interface (GUI) that could be used for a variety of problems. We also present a concept of escaping ACO stagnation in local optima, named suspicious path destruction, that is a part of GRAF-ANT. We also analyze the effect of this hybridization on different variations of Ant colony systems.

Key-Words: - Ant colony system, Evolutionary computing, C# Framework, Optimization

1 Introduction

A large number of problems necessary to be solved by industry and business are in there 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 optima solutions exist, like the use of simple heuristics, a Monte Carlo approach up to more complex methods like Genetic algorithms(GA)[2][3][12] and Simulated annealing(SA)[6]. The observation of ant colony behavior has inspired a metaheuristic method called the Ant Colony Optimization (ACO) [4], which has in some cases given better results than GA and SA [7] 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. Some software implementations of this type like ACOTSP, ANTNET, GUIAnt-Miner with their source are available under a General Public License (GPL). They can be used as guide lines for creating new specialized software. They can even be modified for new problems, but in most cases it is easier to create completely new software. ANT-C is framework developed in the C that could be used as a base for new applications, but has a significant drawback of not being object oriented. An Object oriented framework created in Java by Ugo Chirico (Java Framework for Ant Colony Systems JFACS) is a better solution but 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 colonies systems.

In this paper we present a new object oriented framework for ant colony systems (Graphical Framework for Ant Colony Optimization GRAF-ANT) 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 hybridization implemented in GRAF-ANT that is used for escaping from ACO stagnation. In the final Section 5 we show results of using the hybridization from section 4.

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 an intersection where it decides which path it will take, in the beginning it seems as a random choice but after some time the majority of ants are using the optimal path (Figure 1). This is because the colony works as a group and not just as individual ants and this is achieved by using *pheromone.* Each ant deposits pheromone while walking which marks the rout 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 of it because it will have less time to evaporate before it is deposited again. The colony behaves intelligent because each ant chooses paths that have more pheromone.

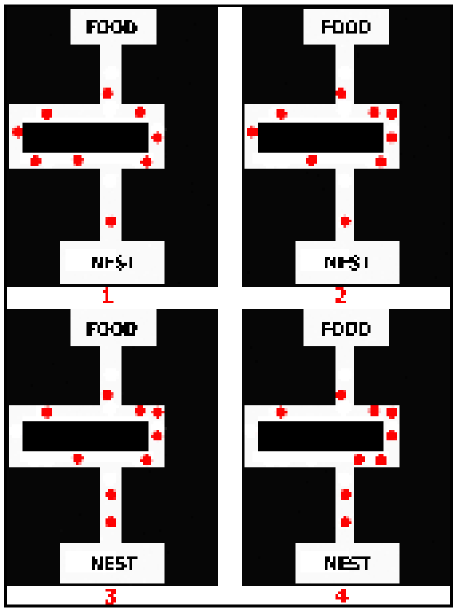

Figure 1: Ant colony behavior over time

There are many different ways of converting the presented behavior into a computational system. We except the one presented by Marco Dorigo and Luca Maria Gambardella[5] 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, 2) give us the probalistic decision method that artificial ant *k*, currently in 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] and 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 created 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, this is obtained using the following formula for

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

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 edge is chosen by an ant it loses some pheromone by the following formula where τ_0 is a predefined constant.

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

3 GRAF-ANT Framework Analysis

ACO can be applied to a large number of different problems. Do 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 [13].

When developing this type of system, the aspect of creating a useful GUI and the possibility of easy visualization of the problem being solved and the progress of ACO is usually neglected. ACO algorithms are very sensitive to input parameters that define the ant colony behavior [9]. 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 do to avoiding of redundant programming. A separate abstract class named *RAntGraphAbstract* is used as a base class for keeping information about problem and calculation variables. The connection between visualization and these variables is very strong and is not only visualizing the solution. In some cases it is easier and even necessary to determinate the problem variables from given visualization [15], our frame work 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(),CalcProbabilityDistribution()*. 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.

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 frame work we have abstracted variations of the basic ACO in to the base class *RAntColonyAbstract*. These variations are explained in detail in article [9] 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*]. TMax, TMin are user defined constants
- 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 when all ants have completed there paths to sort them by the quality of there path, and depending on there rank to decide the amount of pheromone they will leave.

Hybridization of ACO algorithms in many cases results in significant improvement of their efficiency[14]. *RAntColonyAbstract* has the possibility of hybridization in two directions adding local searches to elevate the quality of paths found, and 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* that will be explained in detail in the following section.

When developing ACO the idea of parallelization has at first been exploited in the direction of using different processors for calculating the movement of different ants but this was not the most powerful approach. Similarly to GA which is more efficient when having *N* islands of populations of *M* members than one with population of *NM* members ACO is generally more efficient when it has more small colonies of ants than one big colony [10]. 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[8]. Do to that, multithreaded applications are becoming more effective which means multiple colonies can be calculated in separate threads. GRAF-ANT implements a multi colony approach through the base class *RAntColonyClusterAbstract*. When working with multiple colony system the communication between the colonies is very important [11]. 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 there performance in this type of systems is needed. For this reason we decided to emulate the following network topologies presented in [12].

- *Fully connected* in which the best over all 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 problem in different speeds and some times 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 separate colony behavior can greatly increase the efficiency of the whole system. In GRAF-ANT the search for this optima has been lighten with good visualization and an easy way to test different parameter values.

Because of the significant dependence of ACO performance from 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. The groups are the following 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
- 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

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 while be needed for specific problems, and C# makes this easy through its property grid component and its relationship properties in classes.

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 Sales Problem (TSP). When the algorithm got 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 where two simple criteria very long edges and intersecting edges are very unlikely to be a part of the best path. The next step was to find a way to without of 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 let the colony resume its search. This was a good approach because in the case a suspicious edge was a part of the optimal path ants would come back to it after testing alternative routs. This hybridization improved the quality and calculation time of ACO for 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. This distribution has a correction factor *ExSuspect* that is used to avoid selecting the same edges constantly. When solving particular problems there could be a need for adding alterations to this distribution. Like in the case of 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.

$$\tau_{xy} = \delta \tau_{xy} \quad , \forall xy \in Suspects \qquad (6)$$

In the case of 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, when using the logic of ‘Better safe than sorry’. This is wrong for a couple 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 some times 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

5 Tests and Results

We performed an experimental analysis of the effect of suspicious path destruction (SPD) to different ACO variations. We have used the following parameters for defining the ACO algorithm $q0 = 0.3$, $\alpha = 2$, $\beta = 0.1$, $e = 0.1$ 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 preformed tests for TSP with 50, 100 and 150 cities.

Variation	Best Value	Best Value Iteration	Average
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

Variation	Best Value	Best Value Iteration	Average
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

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 SDP hybridization was added. It is important to notice that it does not only get a better solution but it also falls into stagnation at a higher number of iterations.

Variation	Best Value	Best Value Iteration	Average
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

6 Conclusion

It has been shown that C# is a good choice for creating an ant colony system framework because of 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 from 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 which 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. 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. Two types of hybridization are considered 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 that is why we added this possibility to GRAF-ANT. Multi colony systems are usually connected with network calculations sow 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. This method was tested in combination of all standard variations and gave good results when applied on the traveling salesman problem. 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 decries 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 application code to be compiled when using it as a frame work, 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] 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

[3] 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

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

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

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

[7] 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

[8] 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

[9]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.

[10]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

[11] 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.

[12] 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.

[12] 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.

[13] 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

[14] 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

[15] 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.