Katholieke
Universiteit
Leuven
Faculteit Toegepaste
Wetenschappen
Specialized advanced studies
Master in Artificial Intelligence

Visualizing Swarm Algorithms

Maarten VANKERKOM
Jin YU

Eindverhandeling aangeboden tot het behalen van de graad van gediplomeerde in de aanvullende studies Master in Artificial Intelligence

2003-2004

Promotor: Prof. Dr. Tom Holvoet

Readers: ir. Koenraad Mertens

Prof. Dr. ir. G. Janssens

Faculteit Toegepaste Wetenschappen K.U.Leuven
Master in Artificial Intelligence Academiejaar 2003-2004

Naam en voornaam studenten: Vankerkom Maarten
Yu Jin

Titel:

Visualizing Swarm Algorithms

Korte inhoud (Abstract):

Inspired by the collective foraging behavior of biological ants, several computational models (or ant colony optimization algorithms) are proposed which can be used to solve combinatorial optimization problems, such as the vehicle routing problem and sequential ordering problem, etc. In this article，a software framework is proposed based on which the software agents can be easily created and the dynamics of a swarm system such as the environment with changing pheromone distribution, the movement of agents, can be visualized.

We implement the framework in C++ and apply the proposed methodology to visualizing a swarm system which is used to handle the classical Travelling Salesman Problem (TSP).

Eindverhandeling aangeboden tot het behalen van de graad van gediplomeerde in de aanvullende studies Master in Artificial Intelligence

Promotor: Tom Holvoet
Begeleider: ir. Koenraad Mertens

Table of Contents

1. Introduction

Social insects, such as ants, termites, bees and wasps exhibit a collective problem-solving ability. In particular, many ant species have trail-laying, trail-following behavior when foraging: individual ants deposit a chemical substance called pheromone as they return from a food source to their nest, and foragers follow such pheromone trails and reinforce those trails by dropping pheromones. This reinforcement process results in the selection of the shortest path from the nest to a food source.

The fascinating behavior of ants has been inspiring researchers to study swarm algorithms or ant colony optimization (ACO) algorithms[6] which are computational models currently applied to applications, such as the Traveling Salesman Problem(TSP)[3,6], graph coloring problem[3], network routing problem[3] and a lot more.

The emergent behavior or emergent intelligence, which is the consequence of the self-organization[2] and indirect communication between the ants, is also inspiring from the perspective of computer science, because of the possibility to simulate and potentially exploit this behavior to solve real world applications. Based on the understanding of ant-based algorithms we propose a software framework, which can be employed to imitate the collective behavior of the ants and visualize the dynamics of a swarm. The framework is implemented in the object-oriented programming language C++, and all the elements of the artificial swarm system are modeled in C++ classes, which implement the ant algorithm in different levels of abstraction. As the illustration of the software framework we create a software swarm for handling the symmetric Traveling Salesman Problem (TSP)[2, 3, 6]. For solving the Traveling Salesman Problem, a set of ants cooperate to find better solutions given a specific network. In the software, we use a graph to hold the traveling data, make (software) ants walking on the graph and visualize the intensity of pheromone on edges in time.

2. Theory

In this part we will discuss the theory we use to design the software framework and C++ classes. As this project is based on ant algorithms, we will give a short presentation of the properties we have found central in the context of ant algorithms and in their inspiration, natural ants. And then we will focus on the ACO metaheuristic[3] that is a template for building ant-inspired algorithms. We then continue to show an exemplary application of the basic ant algorithm (Ant System) to solve the Traveling Salesman Problem.

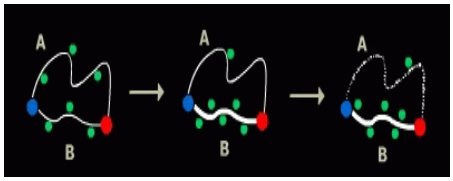
2.1 Collective behavior in social insects

In order to better understand the mechanisms in artificial social insects one must understand the collective behavior in real social insects. In the following, we will give a short description of the primary mechanisms that are determining for the collective behavior of social insects during food foraging, since this is the primary inspiration for the ACO metaheuristic. This will facilitate a later discussion regarding the creation of artificial insects, and also regarding algorithmic problems in a virtual domain.

2.1.1 Self-organization

Self-organization is a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components[2]. The way the system is made up by many small and simple entities enables it to both adapt to changes in the environment and to keep on functioning even though one (or more) of the entities stops functioning (dies). Self-organization in the context of social insects depends on the following mechanisms:

- Positive feedback: Positive feedback is simple behavioral "rules of thumb" that promote the creation of structures. Examples of positive feedback include recruitment and reinforcement[2]. For instance, depending on the quantity or desirability of the food source, stronger deposits of the pheromone are left as the ant proceeds towards its goal. Studies have shown that the ants follow the pheromone trail which exudes the strongest scent.



- Negative feedback: Negative feedback is the mechanism that counteracts the effect of positive feedback. An often-used example of negative feedback is pheromone evaporation.

- Randomness in social insects: Self-organization strongly depends on randomness (random walks and errors in trail-following etc.). Randomness is referred to as a crucial factor for the discovery of new solutions[2].

2.1.2 Stigmergy

Indirect communication occurs among insects when an individual insect modifies the environment and at a later time another insect responds to the new environment. In many ant species colonies, stigmergy refers to the deposition of pheromones by ants while they are moving. Other ants can then smell the deposited pheromones and have a natural tendency to follow the laid trail.

2.2 Artificial ants

Artificial ants are not intended to model real ants. They have abilities that cannot be attributed to real ants. The mechanisms in biological ants described earlier have artificial counterparts. In the following the term ant will refer to artificial ant.

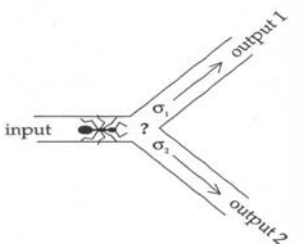
2.2.1 Self-organization

In this section we describe how the different parts of self-organization in colonies of social insects can be modeled by artificial ants in order to achieve the same useful properties of self-organization that biological systems have.

- Positive feedback: An artificial pheromone trail is associated with numerical information in ACO algorithms. In algorithms, artificial pheromone trails are deposited by ants in amounts proportionate to the quality of the solution.

- Negative feedback: In ACO algorithms, pheromone levels are modified over time; typically the amount of pheromone is reduced to a percentage of the original amount.

- Stochastic state transition rule: Every ant algorithm has a stochastic state transition rule. This rule is used to implement the probabilistic decision policy used to select the next state.



2.2.2 Stigmergy

The indirect communication mediated by modifications of environmental states which are only locally accessible by the communicating agents, is known as “artificial” stigmergy. In

ACO algorithms local pheromone trails are the only communication channels among the ants.

2.3 The Ant Colony Optimization metaheuristic

Inspired by the observations and study of ant colonies and ant colony behaviors a metaheuristic called Ant Colony Optimization (ACO)[3] is proposed for characterizing the common properties of a number of algorithms for discrete optimization problems. These algorithms are typically called ACO-algorithms or ant algorithms[3, 6]. A high-level description of the ACO metaheuristic reported in[3] in pseudo-code is as follows:

```
procedure ACO_Meta_heuristic()
   while ( termination_criterion_not_satisfied )
       schedule_activities
        ants_generation_and_activity();
        pheromone_trail_update();
        daemon_actions();{optional}
       end schedule_activities
   end while
end procedure

procedure ants_generation_and_activity()
   while ( available_resources )
       schedule_the_creation_of_a_new_ant();
       new_active_ant();
   end while
end procedure

procedure new_active_ant() {ant lifecycle}
   initialize_ant();
    M= update_ant_memory();
while (current state ≠ target state )
   A = read_local_ant-routing_table();
   P = compute_ transition_probabilities(A;M; problem constraints);
   next state = apply_ant_decision_policy(P; problem constraints);
   move_to_ next state(next_state);
   if (online_step-by-step_pheromone_update)
       deposit_pheromone_on_the_visited_arc();
      update_ant-routing_table();
   end if
   M= update_internal_state();
end while
if (online_delayed_pheromone_update)
   evaluate_solution();
   deposit_pheromone_on_ all_visited_arcs();
   update_ ant-routing_ table();
end if
   die();
end procedure
```

The above pseudo-code states that ACO algorithms construct solutions by constructing paths in a graph. Ant algorithms can be constructed by simply implementing the procedures defined in the pseudo-code to solve a specific problem.

2.4 Ant Algorithms

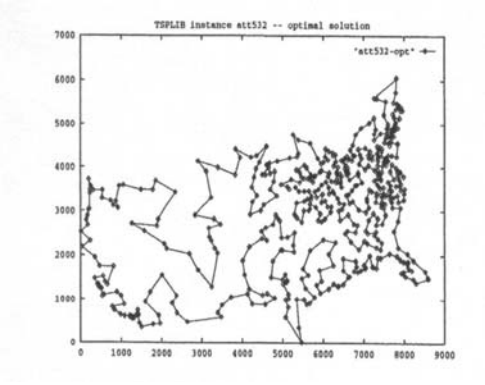
Since we implement an ACO algorithm called Ant System (AS)[3, 6] in our software, we will now discuss the AS. AS is the first ACO algorithm proposed and has been applied to the TSP problem. AS initially showed promising results, but did not scale well enough to be applicable on large graphs and was therefore not competitive with state-of-the-art algorithms for TSP. In spite of the limitations of the AS itself, most of ACO algorithms are inspired by it. In addition to the TSP problem, ant algorithms are applied to some of other application domains, such as quadratic assignment, job scheduling, vehicle routing and graph coloring problems[3], etc.

2.4.1 Swarm Intelligence and Ant algorithms

It is worth mentioning that ant algorithms belong to a larger field of algorithms called Swarm Intelligence algorithms. Swarm Intelligence (SI)[2] is the property of a system whereby the collective behaviors of (unsophisticated) agents interacting locally with their environment cause coherent functional global patterns to emerge. SI provides a basis with which it is possible to explore collective (or distributed) problem solving without centralized control or the provision of a global model.

2.4.2 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a NP hard optimization problem and it is a standard test-bed for new algorithms. It is, therefore, natural that the first application of an ant colony algorithm was to a path optimization problem: the Traveling Salesman Problem[2].



The TSP problem can be stated as finding the minimal length of the walked path, a closed tour on a graph with each node representing a city. There are two kinds of TSP problems:

- Symmetric traveling salesman problem (TSP): Given a set of n nodes and distances for each pair of nodes, find a roundtrip of minimal total length visiting each node exactly once. The distance from node i to node j is the same as from node j to node i.

- Asymmetric traveling salesman problem (ATSP): Similar to the TSP problem. But in this case, the distance from node i to node j and the distance from node j to node i may be different.

In ACO algorithms ants are simple agents which construct tours by moving from city to city on the problem graph. The ants' solution construction is guided by artificial pheromone trails and a priori available heuristic information.

2.4.3 Ant System (AS)

There were three different versions of the AS:

- Ant-density and Ant-quantity: The pheromone update is done directly after a move from one city to another adjacent city.

- Ant-cycle: The pheromone update is only done after all the ants have constructed the tours and the amount of pheromone deposited by each ant is set to be a function of the tour quality.

In the following, we will just discuss the Ant-cycle algorithm. Ant-cycle performs much better then the other two variants, that's also the algorithm we implement in our project as the default algorithm for solving the TSP problem.

2.4.3.1 Stochastic state transition rule

Probability that ant k goes from city i to city j:

$$p_{ij}^{k}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha}.\left[n_{ij}\right]^{\beta}}{\sum_{l \in allowed_k}\left[\tau_{il}(t)\right]^{\alpha}.\left[n_{il}\right]^{\beta}} \; if \;\; j \in allowed_k, \; else \; 0$$

i, j = edge between nodes i, j
n_{ij} = 1/distance (i,j)
α= weight for marking: determine the relative influence of the pheromone trail
β= weight for node "neighbourhoodness": influence of heuristic information
$allowed_k$= unvisited cities for ant k

If $\alpha = 0$, the closest cities are more likely to be selected (corresponds to classical stochastic greedy algorithm).

If $\beta = 0$, only pheromone amplification is at work. This leads to rapid emergence of a stagnation situation with suboptimal generated tours. The ant chooses the edge to continue his path if the probability is higher than a random generated number.

2.4.3.2 Trail deposition and pheromone trail update

After all ants have constructed their tours, the pheromone trails are updated. This is done by first lowering the pheromone strength on all arcs by a constant factor and then allowing each ant to add pheromone on the arcs it has visited:

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)$$

$0 < \rho < 1$ is evaporation rate. It enables the algorithm to forget previously done bad decisions.

$\Delta\tau_i^k(t) = 1/L^k(t)$ If arc (i,j) is used by ant κ and 0 otherwise (L = length of ant's tour)

3. Software Design

The task of our project is to present a design of a generic software framework capable of visualizing different kinds of swarm algorithms in different application contexts. To meet this requirement the framework should be designed in a way so that it can be extended easily for simulating various swarm algorithms.

We use the object-oriented programming language C++ to implement the framework. All elements of a swarm system are modeled using C++ classes, which is defined based on different levels of abstraction so that the code can be reused as much as possible when future features will be added. For testing purpose, we simulate the basic ant algorithm (Ant System (AS)) in the traveling salesman setting.

3.1 Three-layered Framework

For enhancing the extendability of the software framework, we propose a three-layered design: the Model, the Application and the View layers. Inspired by the ACO metaheuristic the Model layer implements the base logic of ant algorithms. On the contrary, the Application layer implements application-oriented operations. That is, the generic part of the ant algorithms is modeled in the Model layer while concrete solutions to the real word applications (e.g. TSP problems) are implemented in the Application layer. Because algorithm visualization is also an important part of the functionality we need to implement for this project, we design the View layer for the visualization purpose.

- The Model layer: Common properties of ant algorithms as indicated by the pseudo-coded ACO metaheuristic are implemented in this layer. All classes belonging to this layer model the base logic of ant algorithms, such as UpdatePheromoneTrail, DepositPheromone, MoveToNextState, ScheduleActivities, etc. Most classes in this layer are supposed to be abstract

classes as they represent ant algorithms at a high level of abstraction. These classes, such as CAgent, CAnt, CColony, CAntSystem, CEnvironment and CPheromone are designed as top-level classes in the hierarchical class framework, which can be inherited by various kinds of application-oriented classes.

- The Application layer: The application layer contains classes which hold application specific information. These classes, such as CTSPAnt, CTSPAntSystem, are separated from the Model layer for their application-oriented characteristics. Inheriting from the corresponding classes in the Model layer, classes in this layer can be defined for different agent-based applications, e.g. implementing a class CRoutingAnt for simulating network routing approach, or a class CGraphColorAnt for handling Graph Coloring problem.

- The View layer: Classes defined in the View layer implement the graphical user interface (GUI) and the visualization. In the current stage we have the class CGUIGraph for creating 2-D displays on the screen indicating the results of swarm actions. Encapsulating visualization functionalities into classes makes it easy to create multiple graphical views. We use CSwarmView class, which is inherited from the Microsoft Foundation Class (MFC) CView to handle user messages associated with the User-Interface Objects. We implement the text output function in the MFC class CMainframe.

3.2 Class Design

It is now time to present the class design. In the current stage we only implement those classes which are necessary for creating a graph-based simulation program in the context of TSP problem. But under the three-layered framework further extensions of the design can be easily conducted by adding new top-level classes for the Model layer, creating new inheriting classes for the Application layer or designing new classes offering more visualization functionalities. The following diagram created in UML with Rational Rose, shows all the important classes and their operations. We notice that according to their roles in a software swarm system these classes can be grouped into three categories:

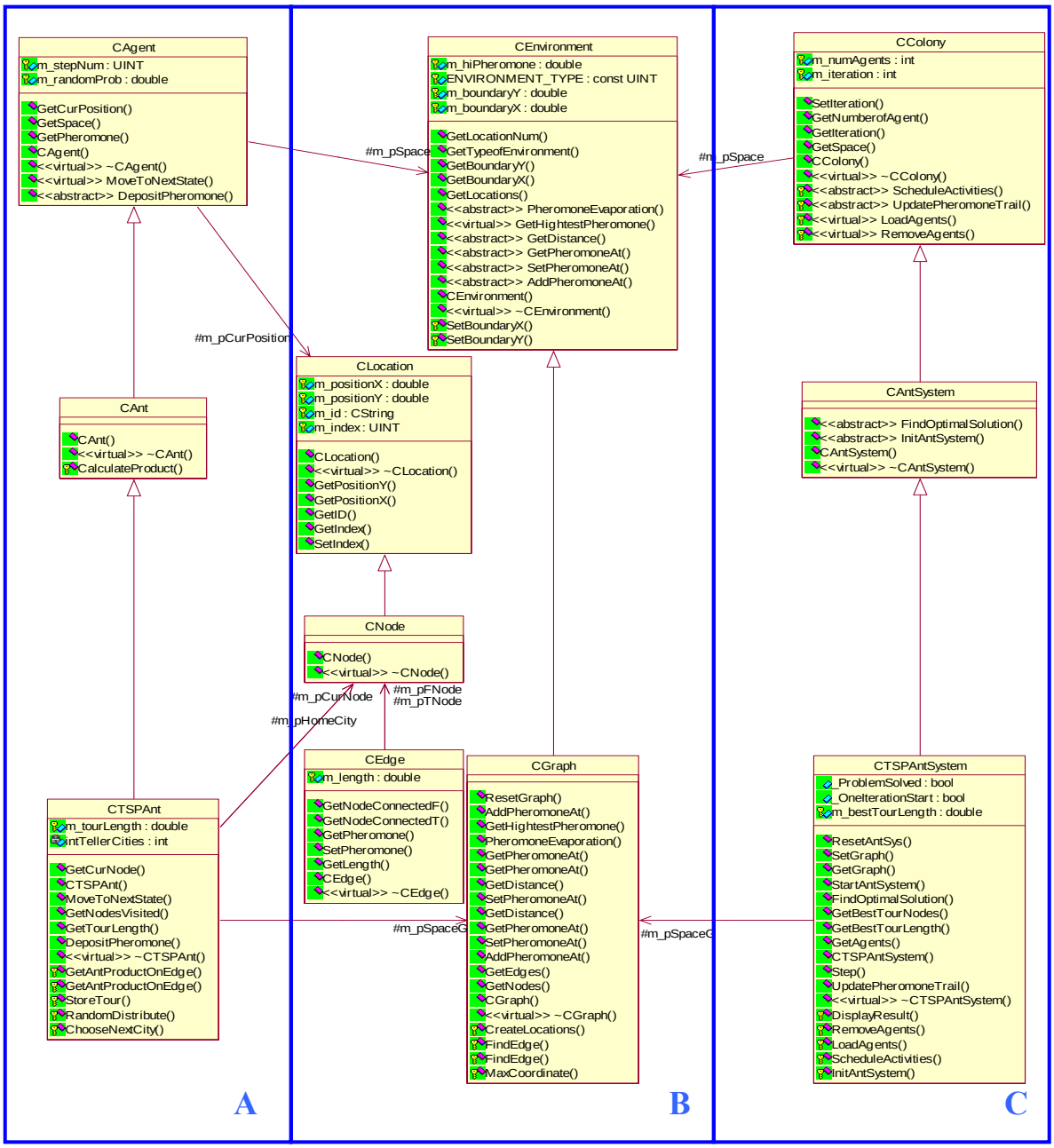
- Individuals(A): Classes belonging to this category model the individual agent which has limited memory and is capable of performing simple actions.

- Environment(B): Classes in this category model the environment in which software agents work. A graph, a matrix or a set of cells are the possible working environment.

- Swarm(C): Classes put in this category model the swarm, a collection of agents executing a schedule of actions.

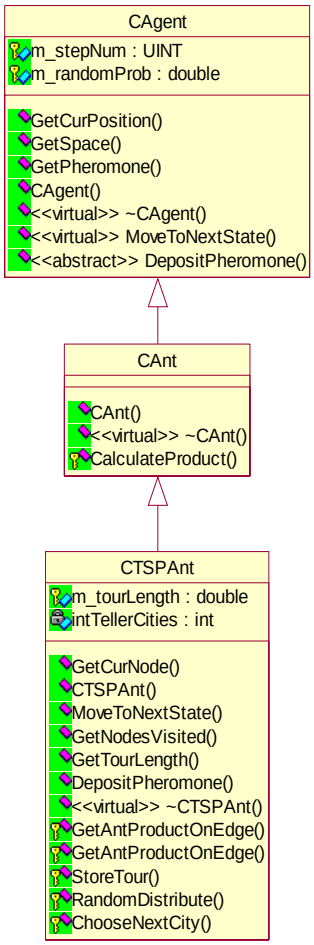
In addition to the above three categories indicated by the following class diagram, we will discuss two extra categories:

- View(D): Classes in this category implement graphical user interface and visualization.
- Auxiliaries(E): Classes in this category facilitate problem-specific operations.



3.2.1 Individuals

There are three classes: CAgent, CAnt and CTSPAnt created for modeling an individual agent. CAgent and CAnt are both top-level classes designed for the Model layer while CTSPAnt for the Application layer.



3.2.1.1 CAgent

CAgent is an abstract class, which models the behaviour of an individual agent in a high level of abstraction. Operations are designed referring to the pseudo-coded Ant Colony Optimization (ACO) - metaheuristic presented in[3]. The basic logic of ant algorithms implemented in this class includes:

- Hold dynamic information of the environment and agent's current position.
- Make a stochastic move to the next state. The concrete definition of the virtual operation MoveToNextState() should be given by subclasses which may implement different algorithms.
- Drop pheromones in the environment. The abstract operation DepositPheromone() must be implemented by subclasses.

3.2.1.2 CAnt

CAnt is a subclass of CAgent. The mathematical implementation of the stochastic state transition rule (referring to section 2.4.1.2) that is often used by many ant algorithms is implemented in the CalculateProduct() operation.

3.2.1.3 CTSPAnt

CTSPAnt implements problem specific functionalities for solving the TSP problem using the Ant System (AS):

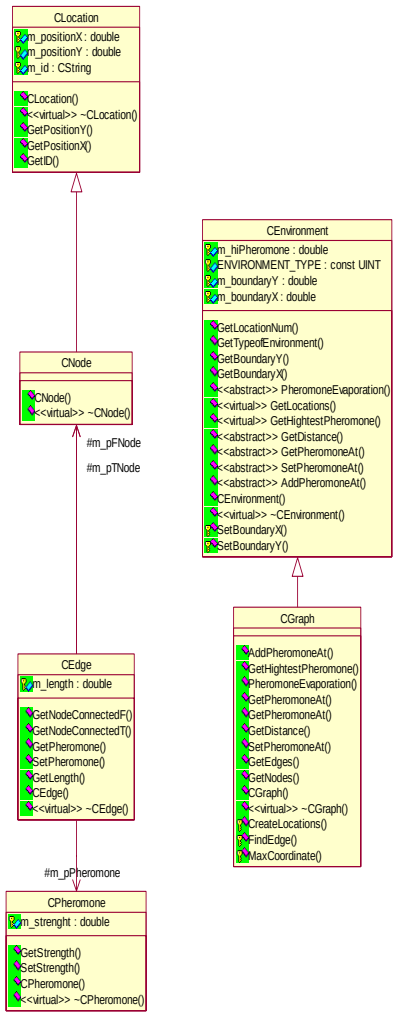
- Initially, an ant is put on a randomly chosen city in a graph-based environment. The RandomDistribute() function gives a random initial home position (starting node), to which the ant returns afterwards.

- Implement the ant-cycle algorithm in the TSP context: choose the next city by taking the pheromone intensity, heuristic value and randomness into account (ChooseNextCity()); calculate the amount of the pheromone according to the quality of the route and increase the pheromone intensity on the edges the ant walked through (DepositPheromone()).

- Remember visited cities (StoreTour()); calculate the length of the tour (GetTourLength()) and get information of all the nodes visited (GetNodesVisited()) for backtracking its path to drop the pheromone.

3.2.2 Environment

In the current design, there are six classes created for modeling the working environment of software agents: CEnvironment, CGraph, CLocation, CNode, CEdge and CPheromone. All the classes in this category belong to the Model layer because they don't carry any application-specific information.



3.2.2.1 CEnvironment

CEnvironment is an abstract class, which represents the general concept of the environment, working space of a collection of agents with pheromone deposited in it. Basic properties associated with a swarm environment are defined in this class:

- Work with a number of locations, which can be graphical nodes or cells in a grid, etc.

- Hold information of space boundaries.

- Calculate distances between locations, and manipulate pheromone intensity in the environment. The abstract operation GetDistance() and several abstract operations for pheromone handling such as SetPheromoneAt() and AddPheromoneAt() must be implemented by subclasses.

3.2.2.2 CLocation

CLocation is created for the common concept of a location, which is employed by the CEnvironment class. It is the super-class of classes which represent specific location structures e.g. a node or a cell.

3.2.2.3 CGraph

CGraph is a simple implementation of a graph. A graph is a collection of nodes and edges. For the TSP problem, the graph is a good representation of the environment. Nodes can be described as cities and routes between cities can be represented by edges. Main functionalities implemented in this class include:

- Manipulate the pheromone intensity over edges (PheromoneEvaporation() and SetPheromoneAt(), etc.).
- Hold coordinate information (MaxCooridinate()).
- Calculate Euclidian distance between two nodes (GetDistance()).

3.2.2.4 CEdge

CEdge represents an undirected connection between two nodes. This class contains information about the length of the edge and the pheromone intensity over it.

3.2.2.5 CNode

CNode represents a vertex in a graph, which is a subclass of CLocation.

3.2.2.6 CPheromone

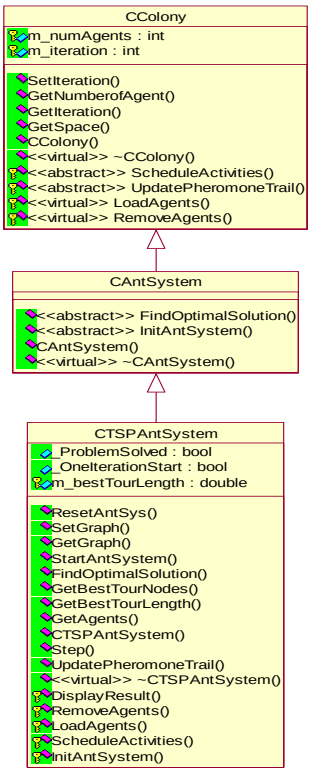
CPheromone is designed to keep the intensity information of a pheromone. As the pheromone is a substance in an environment we put it to the Environment category.

3.2.3 Swarm

Classes in this category are designed to model a swarm, a collection of agents executing a schedule of actions in a certain environment. Three classes are put in this category: CColony, CAntSystem and CTSPAntSystem. CColony and CAntSystem belong to the Model layer because they model the common properties of a swarm, while CTSPAntSystem belongs to the Application layer because it is TSP-specific.

3.2.3.1 CColony

CColony is an abstract class, which defines operations based on the pseudo-coded Ant Colony Optimization (ACO) – metaheuristic. Functionalities implemented in this class, are applicable to all kinds of swarm systems:



- Generate agents at the beginning of every cycle and remove agents at the end. Virtual operations LoadAgents() and RemoveAgents() should be implemented in subclasses because the implementation depends on what kind of agents we work with.

- Schedule the main activities which agents will execute. The abstract operation ScheduleActivities() must be implemented by subclasses since the schedule of activities is algorithm dependent .

- Update pheromone intensity through pheromone deposition and evaporation. The abstract operation UpdatePheromoneTrail() must be implemented by subclasses because the pheromone update mechanism may be different for different ant algorithms.

3.2.3.2 CAntSystem

CAntSystem is an abstract class, which inherits from CColony class. Because discrete optimization is a major application field of ant algorithms, we define an abstract operation FindOptimalSolution() in this class, which must be implemented by inheriting classes using the application-specific optimality criterion.

3.2.3.3 CTSPAntSystem

CTSPAntSystem implements problem specific functionalities for solving the TSP problem using the ant-cycle algorithm (AS):

- Create a swarm of ants (instances of CTSPAnt class) designed for handling the TSP problem in a graph-based environment.

- Implement ant-cycle algorithm in the traveling context: keep track of the best tour, update the pheromone trails according to the result of ant activities and stop the operation when the number of iterations has been reached.

3.2.4 View

All the classes in this category belong to the View layer in the three-layered framework and are designed to implement the Graphical User Interface (GUI) and the visualization.

3.2.4.1 CGUIGraph

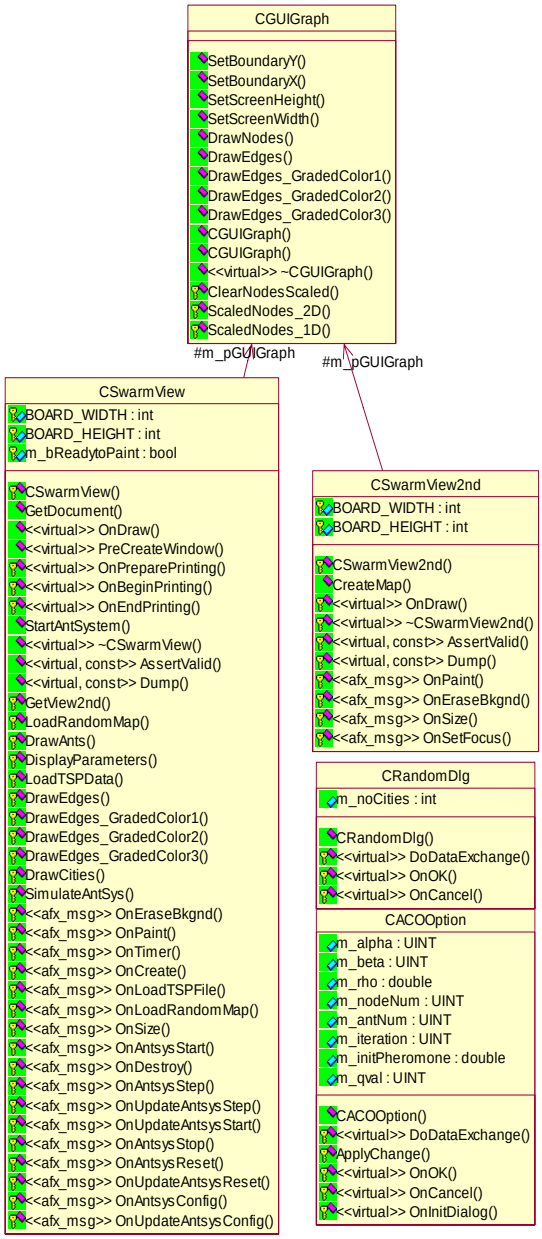
CGUIGraph is designed to visualize the graph-based environment. Operations such as drawing nodes, edges on the screen and coloring the pheromone level over the edges are implemented. Encapsulating these graph-related visualization functionalities into one class makes it easy to create multiple displays.

3.2.4.2 CSwarmView

CSwarmView inherits from the Microsoft Foundation Class (MFC) CView. This class handles user messages associated with User-Interface Objects. In addition, it is associated with the display window in which the pheromone level of the environment is visualized using different gray levels.

3.2.4.3 CSwarmView2nd

CSwarmView2nd inherits from the MFC class CView. It is associated with the display window in which the best route in each cycle of the algorithm is highlighted.



3.2.4.4 CACOOption

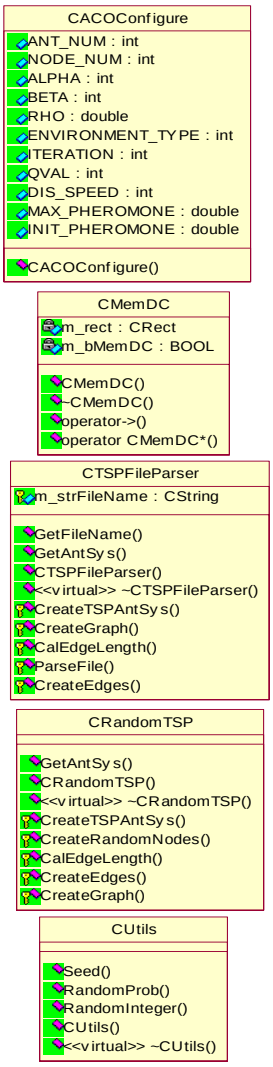
CACOOption is associated with the option dialog, containing the configuration settings for the Ant-cycle (AS) algorithm. Users can easily change the default value of parameters before starting the ant system. The result depends greatly on the settings, and some fine tuning is needed to have a good result for a particular problem.

3.2.4.5 CRandomDlg

CRandomDlg is associated with the random map dialog, which is used to ask for the number of nodes for creating a random TSP map.

3.2.5 Auxiliaries

In order to create a program in the object oriented programming style, we group relevant operations into one class. With their specific functionalities, classes in this category are used as auxiliary classes for creating the TSP simulation program.



3.2.5.1 CACOConfigure

CACOConfigure is the class used to hold all the algorithm related data. This is, parameters associated with the chosen ant algorithm. Parameters will be updated when users adjust the setting by filling in new values in the parameter setting dialog.

3.2.5.2 CMemDC

We use a user-defined C++ class CMemDC developed by Keith Rule (http://www.codeproject.com/gdi/flickerfree.asp) for double buffering. This is a technique used in computer animation to avoid screen flicker, which was a major problem due to the fact that the screen is updated (repainted) several times each second. The technique used is to create an off-screen buffer to which the image is drawn. The final image is then copied to the screen.

3.2.5.3 CTSPFileParser

CTSPFileParser as indicated by its name is used to import a TSP instance from the TSPLIB benchmark library[7]. TSPLIB is a

library of sample instances for the TSP (and related problems) from various sources and of various types. Currently, we only make this class capable to parse TSP instances which have their EDGE_WEIGHT_TYPE as EUC_2D. That means position of cities is indicated by 2-D coordinates and the distance between two cities is calculated by using Euclidean norm.

3.2.5.4 CRandomTSP

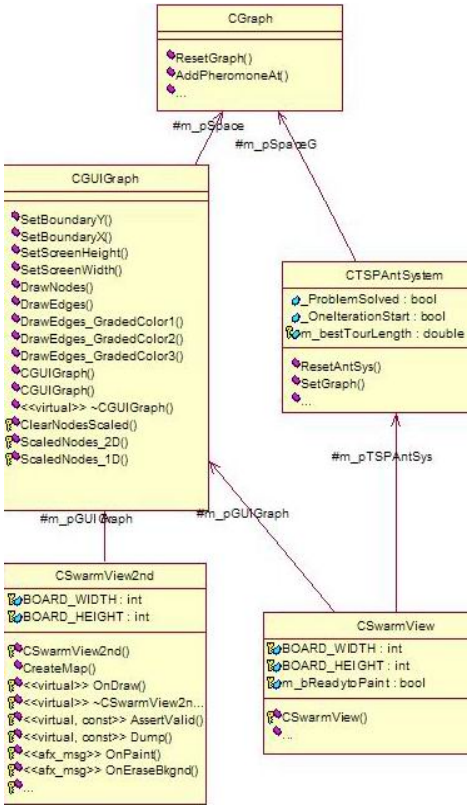
CRandomTSP is used to create random TSP data. Working with the random map dialog, this class is used to asks users for the number of cities and create a TSP map with those cities randomly distributed on it.

3.2.5.5 CUtils

The nondeterministic nature of ants leads to their unpredictable tendency to move from the current state to the next state. Intuitively, we can bring randomness to the simulation program by means of randomly generated numbers. So the CUtils class is created as a random data generator.

3.3 Visualization Approach

As the major task of our project, visualizing the swarm algorithm should be implemented in the simulation program and important aspects to visualize are the structure of the environment, the pheromone distribution in the space and the solutions emerged in the software swarm. In this section we will discuss the class design from the visualization point of view.



As described in section 3.1, Classes implementing visualization functionalities belong to the View layer. More specifically, those classes are CGUIGraph, CSwarmView and CSwarmView2nd. As indicated by the class diagram on the right, the classes mentioned above have to work with the class CTSPAntSystem and the class CGraph in order to get information about the dynamics of the swarm system and the environment in which the swarm system exists.

CGUIGraph class is designed to produce 2-D displays on the screen. Only the information about the graph-based environment where the swarm exists is needed to create the display. For visualizing the graph-based environment

within a predefined display area, operations, such as DrawNodes(), DrawEdges, SetBoundaryX() and SetScreenWidth() are defined for this class. In addition, in order to give a graphical representation of the volatile property of pheromones, we define operations (e.g. DrawEdges_GradedColor1 ()) which visualize the environment in graded color indicating the relative intensity of pheromones over edges. If the pheromone intensity over an edge is high compared to other edges, the color used to draw this edge will have lower brightness (e.g. dark blue).

CSwarmView is associated with the main display window. Swarms of ants are created in this class and the dynamic information about the evolvement of those swarms can be tracked by this class. There can be several swarms associated with it at the same time. Containing all the real-time data about a swarm or several swarms, CSwarmView assigns the visualization task with the environment related information to the instance of CGUIGraph class when visualization task is triggered.

CSwarmView2nd is associated with the auxiliary display window. Having this class, we can create a simulation program with two display windows having different views of the swarm dynamics. The visualization task is also assigned to the instance of CGUIGraph class.

3.4 Implementation

In order to examine the three-layered design, we choose to simulation the most basic ant algorithm, Ant System (Ant-cycle) in the context of traveling salesman problem. C++ is chosen as the programming language to implement the framework. The program was developed and compiled using Visual C++ 6.0. Other object oriented programming languages such as Java are also possible programming languages. We choose to use the Microsoft Foundation Classes (MFC) application framework. The benefits of using MFC application framework are:

- Application framework is small and fast.
- The visual C++ tools reduce coding drudgery.
- The framework is rich in features.

Because the TSP problem is NP-hard, the simulation process can become very time-consuming as the number of cities increases. If the simulation program had only one thread, messages associated with the user interface objects (e.g. buttons and menu items) would be blocked when the main thread is occupied by the running algorithm. In consideration of this problem we assign the algorithm execution task to an auxiliary thread, and the rest of the functionalities: visualization and message response are handled in the main thread. Based on this design, user controls of the simulation program can be resumed

very fast (the response delay is hardly noticed.) just after the main thread hands over the computation task to the auxiliary thread.

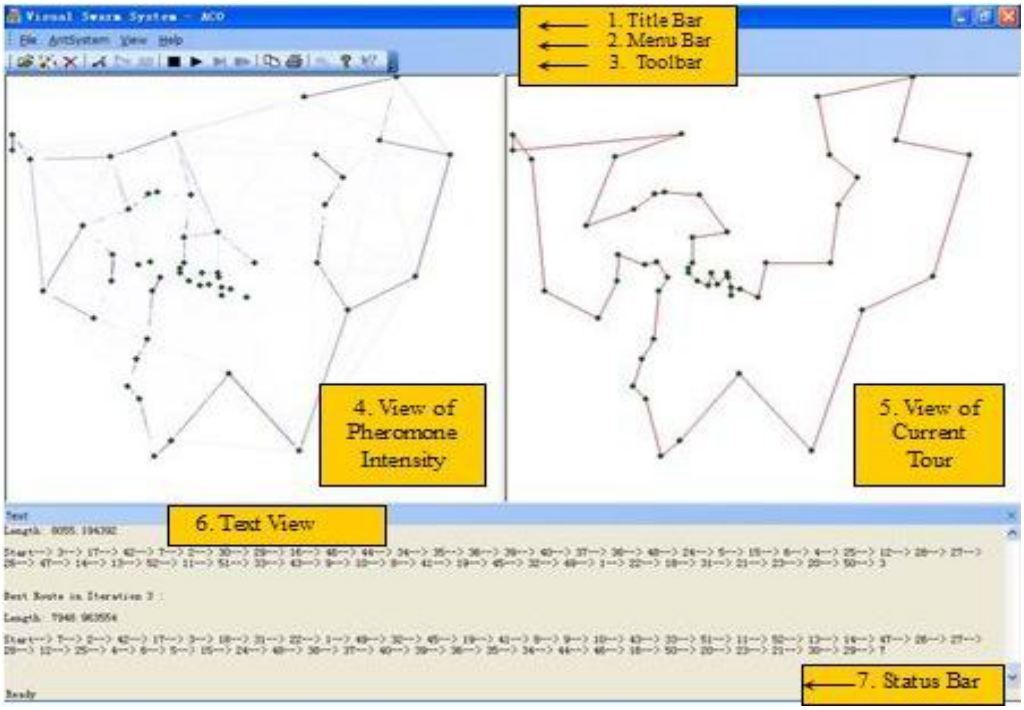
4. Application

As the product of the ideas described in this paper, a simulation program is developed. The program allows the user to apply an ant algorithm (AS) to TSP problems which can be imported from the benchmark TSP library (http://www.math.princeton.edu/tsp/) or randomly created by users themselves. Furthermore, as an important functionality of this program the dynamics of the artificial swarm is visualized and the text information about the result of ant actions is provided. In order to create an application with a professional and user-friendly interface, we employ the Professional User Interface Suite Library[4], which enables us to easily create user-interface objects with Office 2003 look & feel.

4.1 Overview

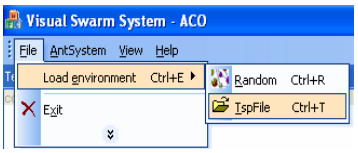
To find your way between all the forms and buttons, we first present an overview of our program. It is a windows application, containing following components:

- Dockable menu bar (2) and toolbar (3) for providing all the controls of the program
- Title bar(1) and status bar (7) for providing the user with software related information
- View of Pheromone Intensity (4) for displaying the environment in graded color indicating the pheromone intensity over edges
- View of Current Tour (5) for highlighting the best route in the current iteration of the chosen algorithm
- Dockable Text View (6) for providing text information on the solutions given by the chosen ant system.

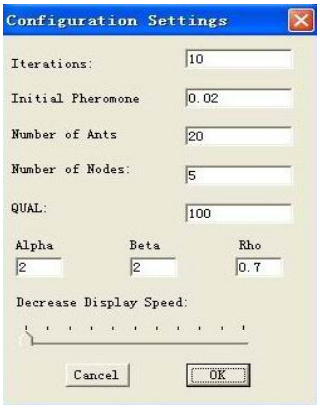


4.2 Graphical User Interface (GUI)

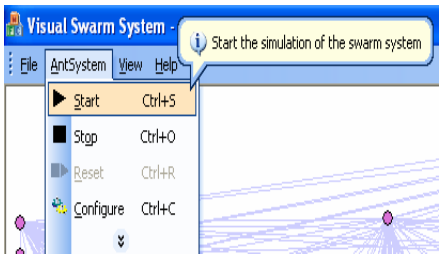
- Two choices are given to the user to build up a TSP working space. One is to import the TSP instances from the TSPLIB benchmark library. The other is to create random TSP data. Both kinds of data are scaled according to the size of the display window and then represented as nodes on the screen.



- With the configuration setting dialog, the users can change parameters of the ant algorithm, which enables them to observe the behavior of the algorithm with different config. The settings for the ant algorithm are the number of ants, Alpha, Beta, the evaporation rate (Rho) and the number of iterations (the number of cycles the algorithm works). Sometimes, when the TSP problem is relatively simple, the simulation is conducted so fast that the time between two consecutive iterations is not long enough for the user to track the result. In order to handle this problem a slider is provided for controlling the execution speed of the algorithm.

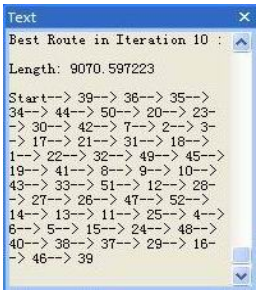


- The available menu items are self explanatory. The Start menu is used to start the simulation. The Stop menu is used to stop the simulation. The Step menu is used to step through the simulation. The Reset menu resets the simulation: pheromones left in the environment are set to be the initial value; controls ever performed to change the execution speed of the algorithm are canceled.

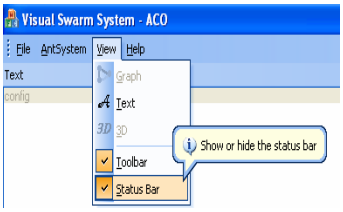


- A dockable toolbar is created as the shortcut to the menu items which may be frequently used.

- A dockable text output window is created for providing algorithm related information: the parameter settings, the best route in every cycle of the algorithm, etc.



- Selection of the application appearance is provided. For instance, users can choose to show or hide the text output window.

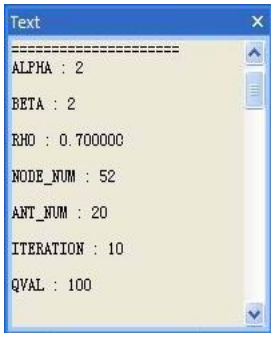


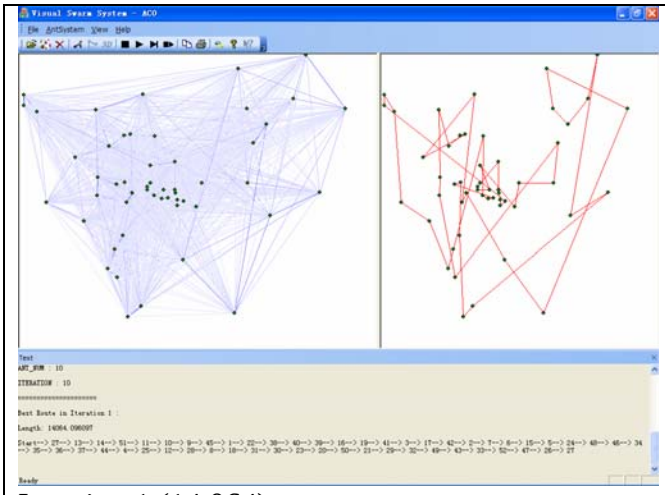
4.3 TSP Simulation

In this section we will see how the Ant System handles the TSP problem through the simulation program. Since our task was to create a framework to visualise the result, we are not responsible for the result itself. It could be noticed that the result is not guaranteed to be optimal by applying the Ant System.

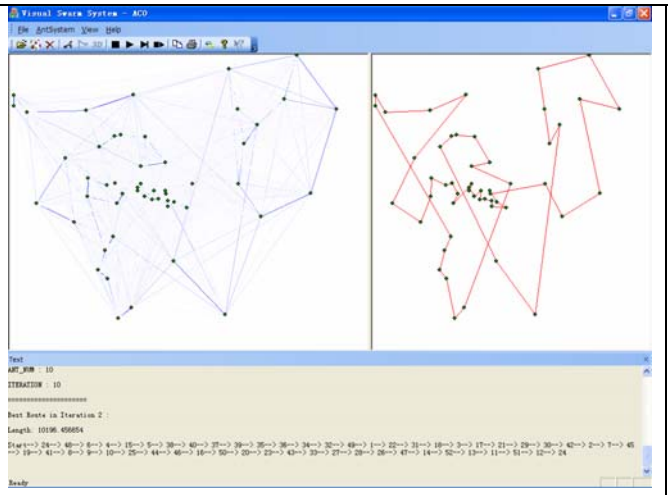
4.3.1 Berlin52: 20 ants

We start with the TSP instance from the TSPLIB benchmark library (Berlin52). This TSP instance contains position information of 52 Berlin cities. 20 ants are allowed to find the best path in 10 iterations. Alpha and Beta are set to be 2, the evaporation rate Rho is set to be 0.7.

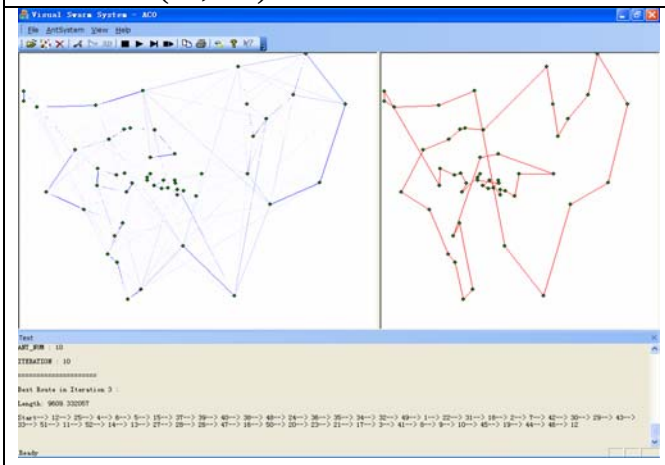




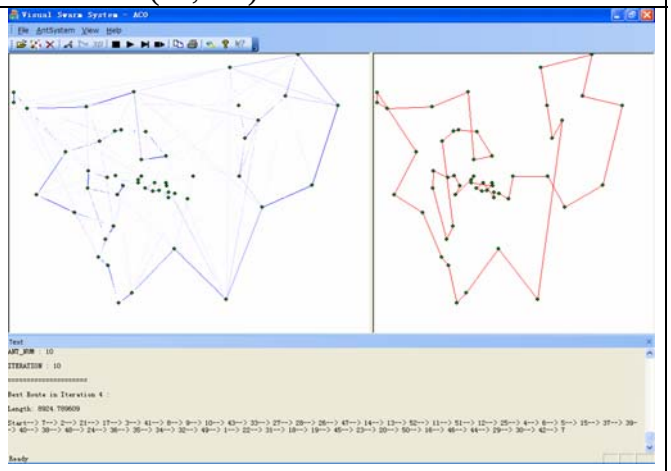
Iteration 1 (14,064)



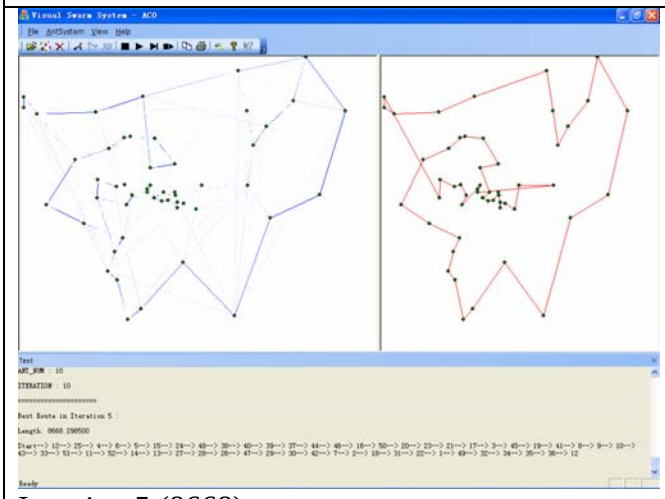
Iteration 2 (10,196)



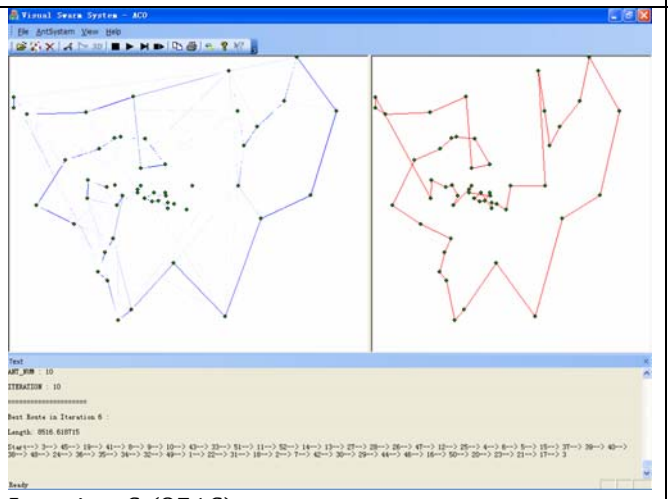
Iteration 3 (9609)



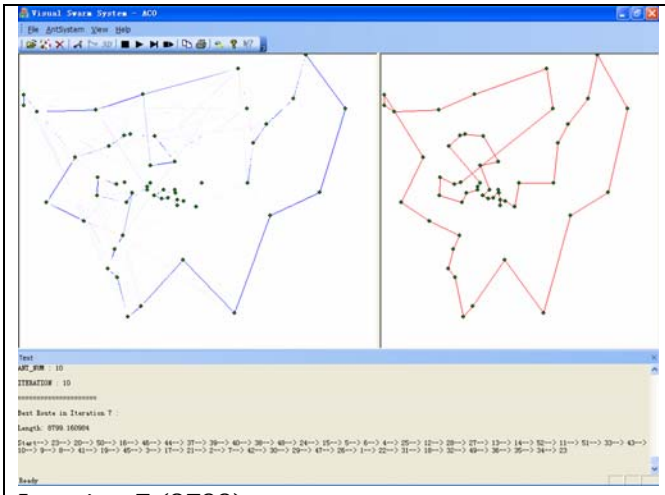
Iteration 4 (8924)



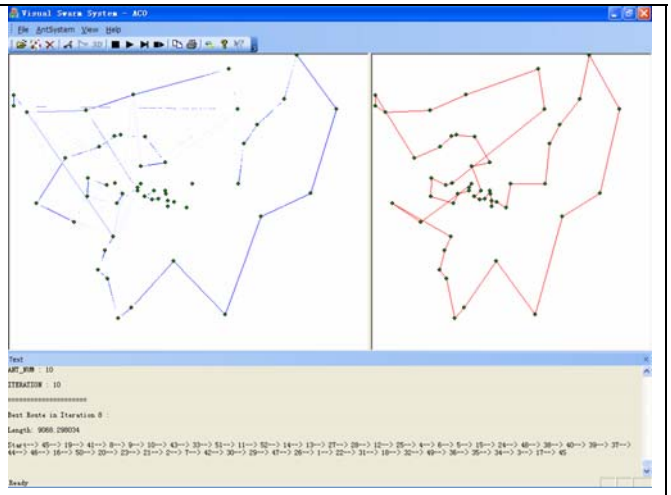
Iteration 5 (8668)



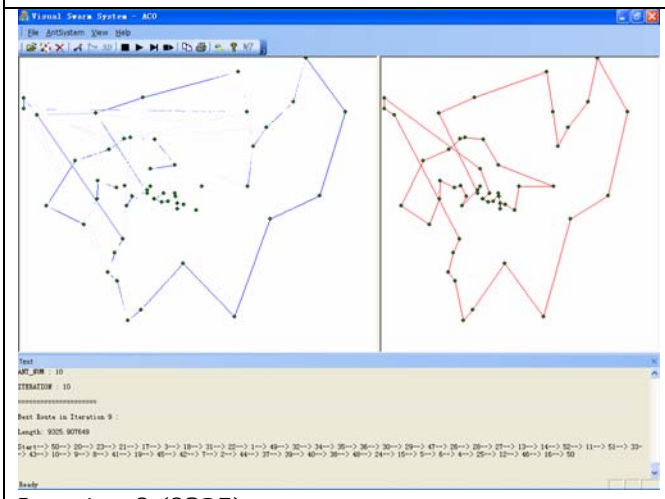
Iteration 6 (8516)



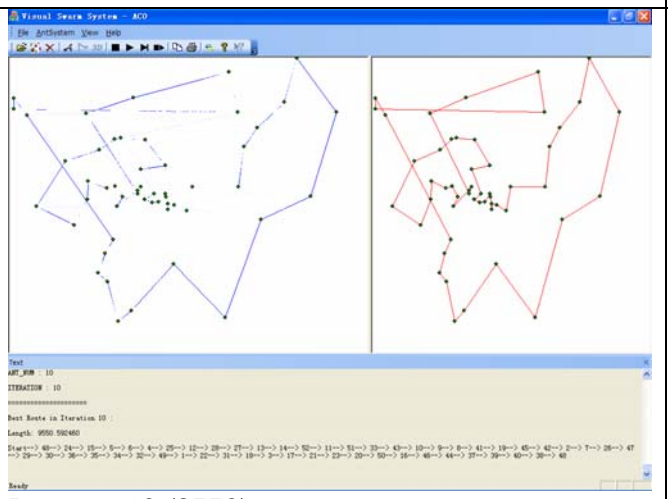
Iteration 7 (8799)



Iteration 8 (9068)



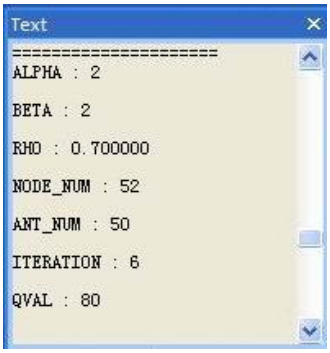
Iteration 9 (9325)



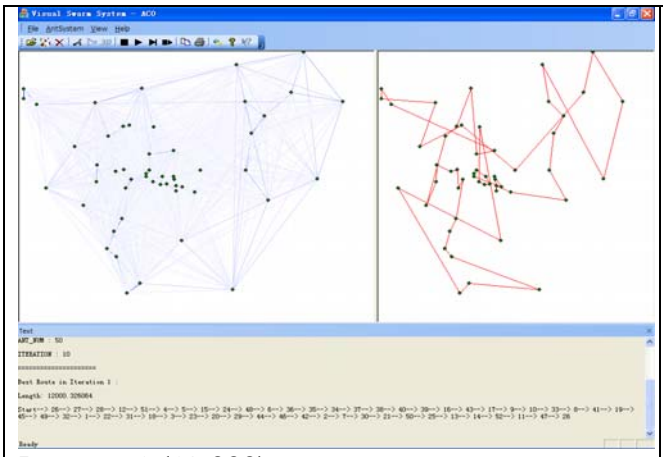
Iteration 10 (9550)

We notice that the result is not so good. The best route in the last iteration is about 9550 long. But the best route an ant walked through is at a certain moment 8516 long but this path is gone after a while.

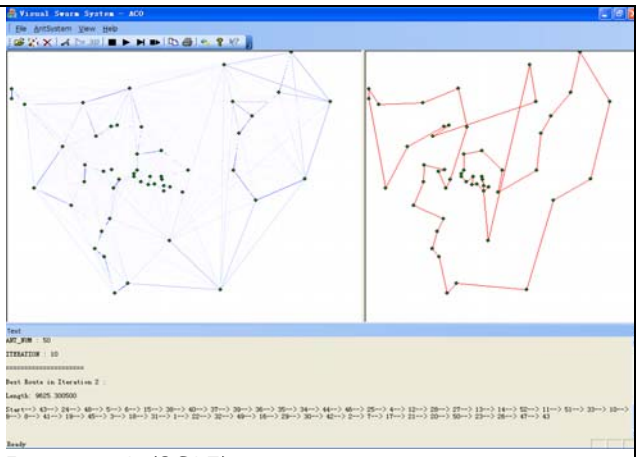
4.3.2 Berlin52: 50 ants



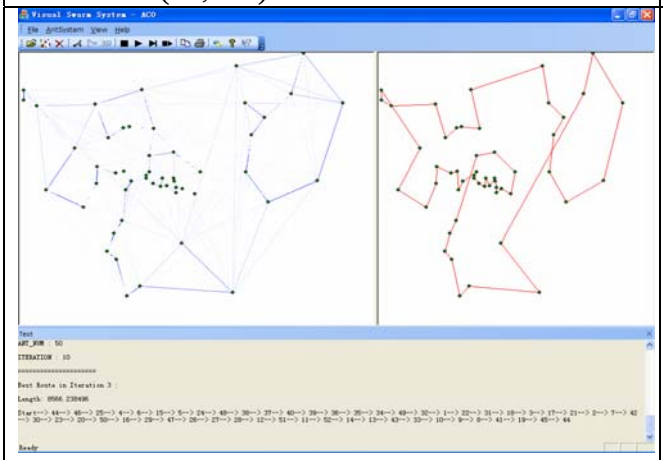
This time we drop 50 ants on the same map of Berlin. We also change the constant QVALUE used in the algorithm for pheromone strength calculation from 100 to 80. Having more ants work on the same problem, we observe that within 6 iterations, the best path ever found is 7805 long compared to 8516 in the previous simulation.



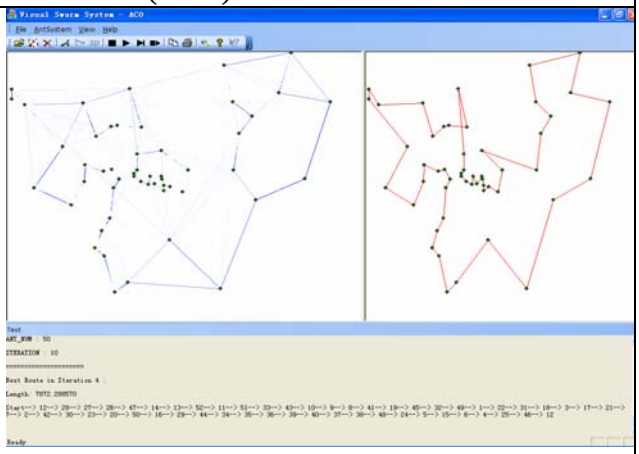
Iteration 1 (12,000)



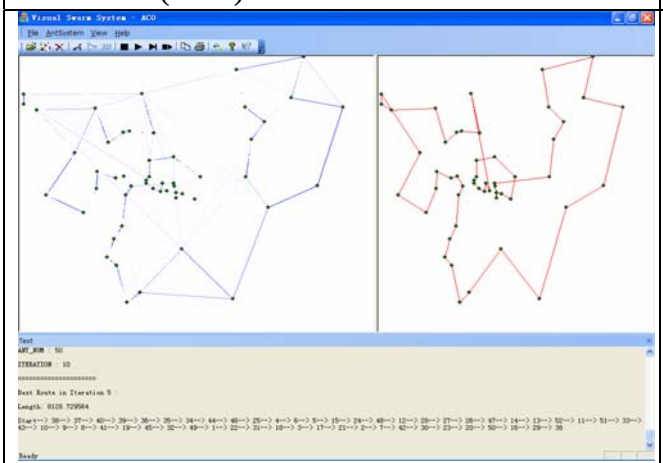
Iteration 2 (9625)



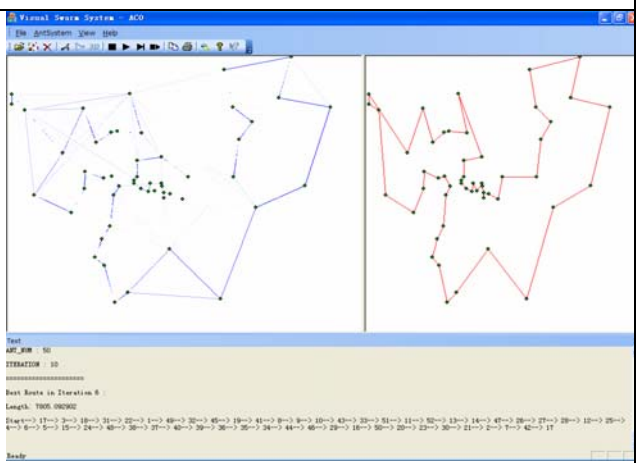
Iteration 3 (8566)



Iteration 4 (7872)



Iteration 5 (8128)



Iteration 6 (7805)

5. Conclusion

The main purpose of our project is to design a software framework which is suitable for visualizing swarm algorithms. For visualizing swarm algorithms, we should design a framework which can first of all model those algorithms. After a thorough study of different swarm algorithms, we propose a framework which is made resilient to different simulation purposes. The most important characteristic of the framework is its three-layered structure which can represent swarm algorithms in different levels of abstraction.

Modeling is a central part of all the activities that lead up to the deployment of a good software. Based on our framework design, we implement it in the object oriented programming language C++. Elements in a swarm system, such as the environment, the pheromone, and the agent are modeled using C++ classes. For illustration purpose, we implement an ant algorithm – Ant System (AS) in the context of the TSP problem. Because of the extendability of the framework other swarm algorithms can also be easily implemented under the layered framework.

In order to ensure an easy implementation of the visualization, we propose a class design which encapsulates visualization functionalities associated with different environment representations (e.g. graph, matrix or 3-D space) in different classes. In the current stage, we implement a class (CGUIGraph) for visualizing a 2-D graph-based environment. In addition, we employ a Microsoft Foundation Classes (MFC) extension library Prof-UIS that enables us to deliver current application with a professional and user-friendly interface.

Based on the software design we develop a simulation program, by which the behavior of software ants in the TSP working space is simulated. Easy controls are provided through user interface objects. The structure of the environment, the changing distribution of pheromones and the solutions given by the swarm are visualized over time to provide indicative information on the performance of the ant algorithm.

References

[1] Dante R. Chialvo and Mark M. Millonas(1995).
How Swarms Build Cognitive Maps.
In *Working Papers from Santa Fe Institute*.

[2] Eric Bonabeau, Marco Dorigo, Guy Theraulaz(1999).
Swarm Intelligence: From Natural to Artificial Systems.
Oxford University Press.

[3] M. Dorigo, G. Di Caro, L. M. Gambardella(1999).
Ant Algorithms for Discrete Optimization.
Proceedings of the Congress on Evolutionary Computation(Vol 2, pp. 1470-1477).
IEEE Press.

[4] Sergiy Lavrynenko
Professional User Interface Suite Library
http://www.codeguru.com/Cpp/W-D/docking/article.php/c4587/
http://www.prof-uis.com/

[5] Swarm Development Group (http://www.swarm.org)
A tutorial introduction to Swarm
http://www.swarm.org/csss-tutorial/frames.html

[6] Thomas Stützle and Marco Dorigo(1999).
ACO Algorithms for the Traveling Salesman Problem.
Evolutionary Algorithms in Engineering and Computer Science (pp. 163-183).

[7] G. Reinelt (1991).
TSPLIB | A Traveling Salesman Problem Library.
ORSA Journal on Computing(Vol 3, pp. 376-384).

[8] Tom Holvoet
Slides for the Multi Agent Systems Course.