Delhi Technological University

Swarm & Evolutionary Computing
Project Report

Anurag Malyala	***2K15/CO/035***
Aparna Jain	***2K15/CO/037***
Ayush Gupta	***2K15/CO/040***
Ayush Gupta	***2K15/CO/041***

CONTENTS

Ant Colony Optimization: A Review of Its Applications

ABSTRACT

Ant colony optimization is a technique for optimization that was introduced in the early 1990's. The inspiring source of ant colony optimization is the foraging behavior of real ant colonies. This behavior is exploited in artificial ant colonies for the search of approximate solutions to discrete optimization problems, to continuous optimization problems, and to important problems in telecommunications, such as routing and load balancing.

In ACO, a set of software agents called artificial ants search for good solutions to a given optimization problem. To apply ACO, the optimization problem is transformed into the problem of finding the best path on a weighted graph. The artificial ants (hereafter ants) incrementally build solutions by moving on the graph. The solution construction process is stochastic and is biased by a pheromone model, that is, a set of parameters associated with graph components (either nodes or edges) whose values are modified at runtime by the ants. [1]

1. INTRODUCTION

Optimization problems are of high importance both in the industrial world as well as for the scientific world. Examples of practical optimization problems include train scheduling, timetabling, shape optimization, telecommunication Network design, or problems from computational biology. The research community has simplified many of These problems in order to obtain scientific test cases such as the well-known traveling salesman problem (TSP). [2] In the TSP a set of locations (e.g. cities) and the distances between them are given. The problem consists of finding a closed tour of minimal length that visits each city once and only once.

Another example is the problem of protein folding. It consists of finding the functional shape or conformation of a protein in two- or three-dimensional space, for example, under simplified lattice models such as the hydrophobic-polar model [3]. Both TSP and Folding fall under problems known as Combinatorial Optimization.

Ant colony optimization (ACO) [4] is one of the most recent techniques for approximate optimization. The inspiring source of ACO algorithms are real ant colonies. More specifically, ACO is inspired by the ants' foraging behavior. At the core of this behavior is the indirect communication between the ants by means of chemical pheromone trails, which enables them to find short paths between their nest and food sources.

ACO algorithms may belong to different classes of approximate algorithms. Seen from the artificial intelligence (AI) perspective, ACO algorithms are one of the most successful strands of swarm intelligence [5]. Seen from the operations research (OR) perspective, ACO algorithms belong to the class of metaheuristics [6].

2. DEFINITIONS

Before we can apply any algorithm to a combinatorial optimization problem, we must first define a model. We define (S,Ω,f), where.

- S is a search space defined over a finite set of discrete decision variables;
- Ω is a set of constraints among the variables; and
- $f: S \rightarrow R_0^+$ is an objective function to be minimized (as maximizing over f is the same as minimizing over −f , every COP can be described as a minimization problem).

The search space S is defined as follows. A set of discrete variables X_i , $i = 1, \dots, n$, with values $v_i^j \in D_i = \{v_i^1, \dots, v_i^{|Di|}\}$, is given. Elements of S are full assignments, that is, assignments in which each variable X_i has a value v_i^j assigned from its domain D_i . The set of feasible solutions SΩ is given by the elements of S that satisfy all the constraints in the set Ω .

A solution $s^* \in S_\Omega$ is called a global optimum if and only if

$$f(s^*) \leq f(s) \forall_s \in S_\Omega .$$

The set of all globally optimal solutions is denoted $by\ S_\Omega^* \subseteq S_\Omega$. Solving a COP requires finding at least one $s^* \in S_\Omega^*$.

3. DESCRIBING THE ACT COLONY OPTIMIZATION METAHEURISTIC

In ACO, artificial ants build a solution to a combinatorial optimization problem by traversing a fully connected construction graph, defined as follows. First, each instantiated decision variable $X_i = v_i^j$ is called a solution component and denoted by cij . The set of all possible solution components is denoted by C . Then the construction graph $G_C(V, E)$ is defined by associating the components C either with the set of vertices V or with the set of edges E.

A pheromone trail value τ_{ij} is associated with each component c_{ij} . Pheromone values are in general a function of the algorithm's iteration $t: \tau_{ij} = \tau_{ij}(t)$. Pheromone values allow the probability distribution of different components of the solution to be modelled. Pheromone values are used and updated by the ACO algorithm during the search.

The ants move from vertex to vertex along the edges of the construction graph exploiting information provided by the pheromone values and in this way incrementally building a solution. Additionally, the ants deposit a certain amount of pheromone on the components, that is, either on the vertices or on the edges that they traverse. The amount $\Delta\tau$ of pheromone deposited may depend on the quality of the solution found. Subsequent ants utilize the pheromone information as a guide towards more promising regions of the search space.

The main differences between the behavior of the real ants and the behavior of the artificial ants in our model are as follows:

- While real ants move in their environment in an asynchronous way, the artificial ants are synchronized, i.e., at each iteration of the simulated system, each of the artificial ants moves from the nest to the food source and follows the same path back.
- While real ants leave pheromone on the ground whenever they move, artificial ants only deposit artificial pheromone on their way back to the nest.
- The foraging behavior of real ants is based on an implicit evaluation of a solution (i.e., a path from the nest to the food source). By implicit solution evaluation we mean the fact that shorter paths will be completed earlier than longer ones, and therefore they will receive pheromone reinforcement more quickly. In contrast, the artificial ants evaluate a solution with respect to some quality measure which is used to determine the strength of the pheromone reinforcement that the ants perform during their return trip to the nest.

3.1. THE ACO METAHEURISTIC

```
1 Set parameters, initialize pheromone trails
2 SCHEDULE_ACTIVITIES
3   ConstructAntSolutions
4   DaemonActions    {optional}
5   UpdatePheromones
6 END_SCHEDULE_ACTIVITIES
```

Figure 1 ACO Metaheuristic

The metaheuristic consists of an initialization step and of three algorithmic components whose activation is regulated by the $SCHEDULE\_ACTIVITIES$ construct. This construct is repeated until a termination criterion is met. Typical criteria are a maximum number of iterations or a maximum CPU time.

The $SCHEDULE\_ACTIVITIES$ construct does not specify how the three algorithmic components are scheduled and synchronized. In most applications of ACO to NP-hard problems however, the three algorithmic components undergo a loop that consists

- the construction of solutions by all ants.
- the (optional) improvement of these solution via the use of a local search algorithm
- the update of the pheromones. These three components are now explained in more details.

3.1.1 Sub-procedure ConstructAntSolutions

```
1 s = < >
2 Determine N (s)
3 whileN (s) #= ∅ do
4 c ← ChooseFrom(N (s))
5 s ← extend s by appending solution component c
6 Determine N (s)
7 end while
```

A set of m artificial ants construct solutions from elements of a finite set of available solution components $C = \{c_{ij}\}, i = 1, \dots, n, j = 1, \dots, |D_i|$. A solution construction starts

with an empty partial solution $s^p = \emptyset$. Then, at each construction step, the current partial solution s^p is extended by adding a feasible solution component from the set of feasible neighbors $N(s^p) \subseteq C$. The process of constructing solutions can be regarded as a path on the construction graph $G_C(V, E)$. The allowed paths in G_C are implicitly defined by the solution construction mechanism that defines the set $N(s^p)$ with respect to a partial solution s^p.

The choice of a solution component from $N(s^p)$ is done probabilistically at each construction step. The exact rules for the probabilistic choice of solution components vary across different ACO variants. The best-known rule is the one of ant system (AS) [7] [8]

$$p(c_{ij}|s^p) = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{c_{il} \in N(s^p)} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}} , \forall c_{ij} \in N(s^p) ,$$

where τ_{ij} and η_{ij} are respectively the pheromone value and the heuristic value associated with the component c_{ij} . Furthermore, α and β are positive real parameters whose values determine the relative importance of pheromone versus heuristic information.

3.1.2 Sub-Procedure DaemonAcrtions

Once solutions have been constructed, and before updating the pheromone values, often some problem specific actions may be required. These are often called daemon actions, and can be used to implement problem specific and/or centralized actions, which cannot be performed by single ants. The most used daemon action consists in the application of local search to the constructed solutions: the locally optimized solutions are then used to decide which pheromone values to update.

3.1.3 Sub-Procedure UpdatePheromones

The aim of the pheromone update is to increase the pheromone values associated with good solutions, and to decrease those that are associated with bad ones. Usually, this is achieved by

- decreasing all the pheromone values through pheromone evaporation
- increasing the pheromone levels associated with a chosen set of good solutions S_{upd}

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \sum_{s \in S_{upd} | c_{ij} \in s} F(s) ,$$

where S_{upd} is the set of solutions that are used for the update, $\rho \in (0,1]$ is a parameter called evaporation rate, and $F: S \rightarrow R + 0$ is a function such that;

$$f(s) < f(s') \Rightarrow F(s) \geq F(s'), \forall s \neq s' \in S .$$

F(·) is commonly called the *fitness function*.

Pheromone evaporation implements a useful form of forgetting, favoring the exploration of new areas in the search space. Different ACO algorithms, for example ant colony system [9], or MAX-MIN ant system (MMAS) [3] differ in the way they update the pheromone.

Instantiations of the update rule given above are obtained by different specifications of S_{upd} , which in many cases is a subset of $S_{iter} \cup \{s_{bs}\}$, where S_{iter} is the set of solutions that were constructed in the current iteration, and sbs is the best-so-far solution, that is, the best solution found since the first algorithm iteration. A well-known example is the AS-update rule, that is, the update rule of ant system [7] [8].

$$S_{upd} \leftarrow S_{iter}$$

An example of a pheromone update rule that is more often used in practice is the IB-update rule

$$Supd \leftarrow argmax_{s \in S_{iter}} F(s)$$

The IB-update rule introduces a much stronger bias towards the good solutions found than the AS-update rule. Although this increases the speed with which good solutions are found, it also increases the probability of premature convergence. An even stronger bias is introduced by the BS-update rule, where BS refers to the use of the best-so-far solution s_{b_s} . In this case, S_{upd} is set to $\{s_{sb}\}$. In practice, ACO algorithms that use variations of the IB-update or the BS-update rules and that additionally include mechanisms to avoid premature convergence, achieve better results than those that use the AS-update rule.

4. VARIATIONS ON THE ACO ALGORITHM

4.1. ELITIST ANT SYSTEM

Ant system (AS) was the first ACO algorithm to be proposed in the literature [7] [8] [10] Its main characteristic is that the pheromone values are updated by all the ants that have completed the tour. Solution components c_{ij} are the edges of the graph, and the pheromone update for τ_{ij} , that is, for the pheromone associated to the edge joining cities i and j , is performed as follows;

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \ ,$$

where $\rho \in (0,1]$ is the evaporation rate, m is the number of ants, and $\Delta\tau_{ij}^{k}$ is the quantity of pheromone laid on edge (i,j) by the k^{th} ant. where L_k is the tour length of the k^{th} ant.

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{L_k} & \text{if } \backslash \text{ ant } k \text{ used } \backslash \text{ edge } (i,j) \text{ in } \backslash \text{ its } \backslash \text{ tour,} \\ 0 & \text{otherwise,} \end{cases}$$

When constructing the solutions, the ants in AS traverse the construction graph and make a probabilistic decision at each vertex. The transition probability $p(c_{ij}|s_k^p)$ of the k[th] ant moving from city i to city j is given by:

$$p(c_{ij}|s_k^p) = \begin{cases} \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{c_{il} \in N(s_k^p)} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}} & \text{if } j \in N(s_k^p), \\ 0 & \text{otherwise,} \end{cases}$$

where $N(s_k^p)$ is the set of components that do not belong yet to the partial solution s_k^p of ant k , and α and β are parameters that control the relative importance of the pheromone versus the heuristic information $\eta_{ij} = 1/d_{ij}$, where d_{ij} is the length of component c_{ij} (i.e., of edge (i,j)).

4.2. ANT COLONY SYSTEM

The first important difference between ACS [9] and AS is the form of the decision rule used by the ants during the construction process. Ants in ACS use the so-called pseudorandom proportional rule: the probability for an ant to move from city i to city j depends on a random variable q uniformly distributed over [0,1] , and a parameter q0 ; if q $\leqslant$q0 , then, among the feasible components, the component that maximizes the product $\tau_{il}\eta_{il}^{\beta}$ is chosen; otherwise, the same equation as in AS is used.

This rather greedy rule, which favors exploitation of the pheromone information, is counterbalanced by the introduction of a diversifying component: the local pheromone update. The local pheromone update is performed by all ants after each construction step. Each ant applies it only to the last edge traversed:

$$\tau_{ij} = (1-\varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0$$

where $\varphi \in (0,1]$ is the pheromone decay coefficient, and τ_0 is the initial value of the pheromone.

The main goal of the local update is to diversify the search performed by subsequent ants during one iteration. In fact, decreasing the pheromone concentration on the edges as they are traversed during one iteration encourages subsequent ants to choose other edges and hence to produce different solutions. This makes less likely that several ants produce identical solutions during one iteration. Additionally, because of the local pheromone update in ACS, the minimum values of the pheromone are limited.

As in AS, also in ACS at the end of the construction process a pheromone update, called offline pheromone update, is performed.

ACS offline pheromone update is performed only by the best ant, that is, only edges that were visited by the best ant are updated, according to the equation:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij}^{best}$$

where $\Delta\tau_{ij}^{best} = 1/L_{best}$ if the best ant used edge (i,j) in its tour, $\Delta\tau_{ij}^{best} = 0$ otherwise (L_{best} can be set to either the length of the best tour found in the current iteration $-iteration - best$, L_{ib} -- or the best solution found since the start of the algorithm -- best-so-far, L_{bs}).

4.3. MAX-MIN ANT SYSTEM

MAX-MIN ant system (MMAS) is another improvement [3], over the original ant system idea. MMAS differs from AS in that:

- only the best ant adds pheromone trails.
- the minimum and maximum values of the pheromone are explicitly limited (in AS and ACS these values are limited implicitly, that is, the value of the limits is a result of the algorithm working rather than a value set explicitly by the algorithm designer).

The pheromone update equation takes the following form:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \Delta\tau_{ij}^{best}$$

where $\Delta\tau_{ij}^{best} = 1/L_{best}$ if the best ant used edge (i,j) in its tour, $\Delta\tau_{ij}^{best} = 0$ otherwise, where Lbest is the length of the tour of the best ant. As in ACS, Lbest may be set (subject to the algorithm designer decision) either to L_{ib} or to L_{bs} , or to a combination of both.

The pheromone values are constrained between τ_{min} and τ_{max} by verifying, after they have been updated by the ants, that all pheromone values are within the imposed limits, τ_{ij} is set to τ_{max} if $\tau_{ij} > \tau_{max}$ and to τ_{min} if $\tau_{ij} < \tau_{min}$. It is important to note that the pheromone update equation of MMAS is applied, as it is the case for AS, to all the edges while in ACS it is applied only to the edges visited by the best ants.

The minimum value τ_{min} is most often experimentally chosen. The maximum value τ_{max} may be calculated analytically provided that the optimum ant tour length is known.

In the case of the TSP, $\tau_{max} = 1/(\rho \cdot L^*)$, where L^* is the length of the optimal tour. If L^* is not known, it can be approximated by L_{bs} . It is also important to note that the initial value of the trails is set to τ_{max} , and that the algorithm is restarted when no improvement can be observed for a given number of iterations.

5. APPLICATIONS OF ACO TO DISCRETE OPTIMIZATION PROBLEMS

ACO algorithms have been applied to many discrete optimization problems. First, classical problems other than the TSP, such as assignment problems, scheduling problems, graph coloring, the maximum clique problem, or vehicle routing problems were tackled. More recent applications include, for example, cell placement problems arising in circuit design, the design of communication networks, or bioinformatics problems. In recent years some researchers have also focused on the application of ACO algorithms to multi-objective problems and to dynamic or stochastic problems.

Recent applications of ACO to problems arising in these areas include the applications to protein folding [11],to multiple sequence alignment [12], and to the prediction of major histocompatibility complex (MHC) class II binders [13]. The protein folding problem is one of the most challenging problems in computational biology, molecular biology, biochemistry and physics. It consists of finding the functional shape or conformation of a protein in two- or three-dimensional space, for example, under simplified lattice models such as the hydrophobic-polar model.

ACO algorithms are currently among the state-of-the-art methods for solving, for example, the sequential ordering problem [14], the resource constraint project scheduling problem [15], the open shop scheduling problem, and the 2D and 3D hydrophobic polar protein folding problem [11].

6. APPLICATIONS OF ACO TO CONTINUOUS OPTIMIZATION PROBLEMS [16]

Even though ACO was designed for discrete problems, their adaptation to solve continuous optimization problems enjoys an increasing attention Early applications of ant-based algorithms to continuous optimization include algorithms such as Continuous ACO (CACO) [17], the API algorithm [18], and Continuous Interacting Ant Colony (CIAC) [19], all these approaches are conceptually quite different from ACO for discrete problems.

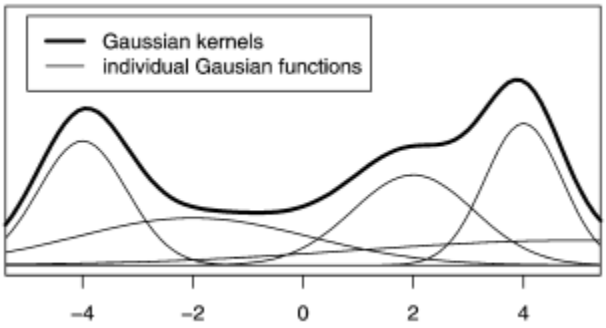
ACO algorithms for discrete optimization problems solutions are constructed by sampling at each construction step a discrete probability distribution that is derived from the pheromone information. In a way, the pheromone information represents the stored search experience of the algorithm. In contrast, ACO for continuous optimization—in the literature denoted by ACOR—utilizes a continuous probability density function (PDF). This density function is produced, for each solution construction, from a population of solutions that the algorithm keeps at all times. The management of this population works as follows. Before the start of the algorithm, the population—whose cardinality k is a parameter of the algorithm—is filled with random solutions. This corresponds to the pheromone value initialization in ACO algorithms for discrete optimization problems. Then, at each iteration the set of generated solutions is added to the population and the same number of the worst solutions are removed from it. This action corresponds to the pheromone update in discrete ACO. The aim is to bias the search process towards the best solutions found during the search.

For constructing a solution, an ant chooses at each construction step $i = 1,...,n$, a value for decision variable Xi. In other words, if the given optimization problem has n dimensions, an ant chooses in

each of n construction steps a value for exactly one of the dimensions. In the following we explain the choice of a value for dimension i. For performing this choice an ant uses a Gaussian kernel, which is a weighted superposition of several Gaussian functions, as PDF. Concerning decision variable Xi (i.e., dimension i) the Gaussian kernel Gi is given as follows:

$$G_i(x) = \sum_{j=1}^{k} \omega_j \frac{1}{\sigma_j \sqrt{2\pi}} e^{-\frac{(x-\mu_j)^2}{2\sigma_j^2}}, \quad \forall x \in \mathbb{R},$$

where the j th Gaussian function is derived from the j th member of the population, whose cardinality is at all times k. Note that ω1, μ1, and σ1 are vectors of size k. Hereby, ω1 is the vector of weights, whereas μ1 and σ1 are the vectors of means and standard deviations respectively



In ACOR this is accomplished as follows. Each ant, before starting a solution construction, that is, before choosing a value for the first dimension, chooses exactly one of the Gaussian functions j , which is then used for all n construction steps. The choice of this Gaussian function, in the following denoted by j ∗, is performed with probability

$$\mathbf{p}_j = \frac{\omega_j}{\sum_{l=1}^{k} \omega_l}, \quad \forall j = 1, \ldots, k,$$

where ωj is the weight of Gaussian function j , which is obtained as follows. All solutions in the population are ranked according to their quality (e.g., the inverse of the objective function value in the case of minimization) with the best solution having rank 1. Assuming the rank of the j th solution in the population to be r, the weight ωj of the j th Gaussian function is calculated according to the following formula:

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(r-1)^2}{2q^2k^2}},$$

which essentially defines the weight to be a value of the Gaussian function with argument r, with a mean of 1.0, and a standard deviation of qk. Note that q is a parameter of the algorithm. In case the value of q is small, the best-ranked solutions are strongly preferred, and in case it is larger, the

probability becomes more uniform. Due to using the ranks instead of the actual fitness function values, the algorithm is not sensitive to the scaling of the fitness function.

The sampling of the chosen Gaussian function j ∗ may be done using a random number generator that is able to generate random numbers according to a parameterized normal distribution, or by using a uniform random generator in conjunction with (for instance) the Box–Muller method [20]. However, before performing the sampling, the mean and the standard deviation of the j ∗th Gaussian function must be specified. First, the value of the ith decision variable in solution j ∗ is chosen as mean, denoted by μj ∗ , of the Gaussian function. Second, the average distance of the other population members from the j ∗th solution multiplied by a parameter ρ is chosen as the standard deviation, denoted by σj ∗ , of the Gaussian function:

$$\sigma_{j^*} = \frac{1}{k}\rho \sum_{l=1}^{k} \sqrt{(x_i^l - x_i^{j^*})^2}.$$

Parameter ρ, which regulates the speed of convergence, has a role similar to the pheromone evaporation rate ρ in ACO for discrete problems. The higher the value of ρ ∈ (0, 1), the lower the convergence speed of the algorithm, and hence the lower the learning rate. Since this whole process is done for each dimension (i.e., decision variable) in turn, each time the distance is calculated only with the use of one single dimension (the rest of them are discarded). This allows the handling of problems that are scaled differently in different directions.

ACOR was successfully applied both to scientific test cases as well as to real world problems such as feedforward neural network training [21] [22].

7. ACO, STUDY BY EXAMPLE

The ACO algorithm was first used for solving the Traveling Salesman Problem.

7.1 A LAYMAN'S EXAMPLE

The easiest way to understand how ant colony optimization works is by means of an example. We consider its application to the traveling salesman problem (TSP). In the TSP a set of locations (e.g. cities) and the distances between them are given. The problem consists of finding a closed tour of minimal length that visits each city once and only once.

To apply ACO to the TSP, we consider the graph defined by associating the set of cities with the set of vertices of the graph. This graph is called construction graph. Since in the TSP it is possible to move from any given city to any other city, the construction graph is fully connected and the number of vertices is equal to the number of cities. We set the lengths of the edges between the vertices to be proportional to the distances between the cities represented by these vertices and we associate pheromone values and heuristic values with the edges of the graph. Pheromone values are modified at runtime and represent the

cumulated experience of the ant colony, while heuristic values are problem dependent values that, in the case of the TSP, are set to be the inverse of the lengths of the edges.

The ants construct the solutions as follows. Each ant starts from a randomly selected city (vertex of the construction graph). Then, at each construction step it moves along the edges of the graph. Each ant keeps a memory of its path, and in subsequent steps it chooses among the edges that do not lead to vertices that it has already visited. An ant has constructed a solution once it has visited all the vertices of the graph. At each construction step, an ant probabilistically chooses the edge to follow among those that lead to yet unvisited vertices. The probabilistic rule is biased by pheromone values and heuristic information: the higher the pheromone and the heuristic value associated to an edge, the higher the probability an ant will choose that particular edge. Once all the ants have completed their tour, the pheromone on the edges is updated. Each of the pheromone values is initially decreased by a certain percentage. Each edge then receives an amount of additional pheromone proportional to the quality of the solutions to which it belongs (there is one solution per ant).

This procedure is repeatedly applied until a termination criterion is satisfied.

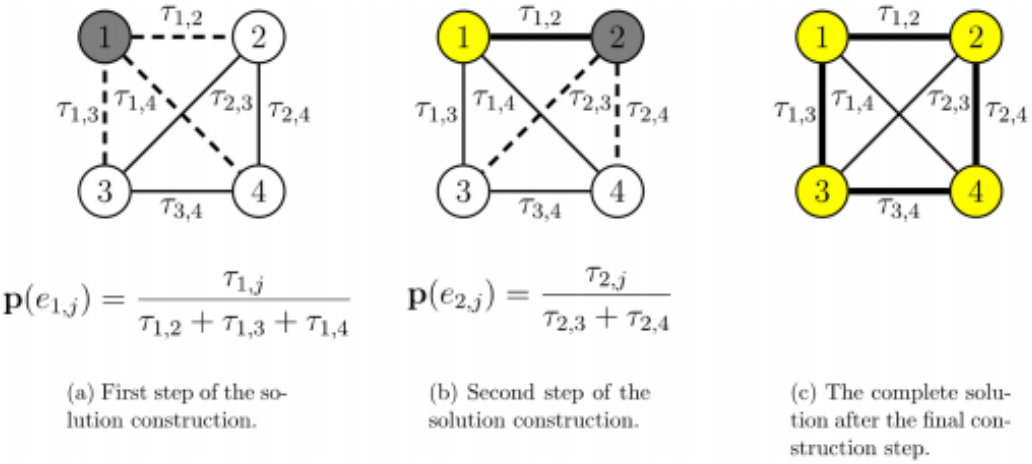
7.2 Formal Definition and Problem Solution

In the TSP is given a completely connected, undirected graph G = (V , E) with edge-weights. The nodes V of this graph represent the cities, and the edge weights represent the distances between the cities. The goal is to find a closed path in G that contains each node exactly once (henceforth called a tour) and whose length is minimal. Thus, the search space S consists of all tours in G. The objective function value f (s) of a tour $s \in S$ is defined as the sum of the edge-weights of the edges that are in s. The TSP can be modelled in many ways as a discrete optimization problem. The most common model consists of a binary decision variable Xe for each edge in G. If in a solution Xe = 1, then edge e is part of the tour that is defined by the solution.

From an Ant System Perspective, , the edges of the given TSP graph can be considered solution components, i.e., for each ei,j is introduced a pheromone value $\tau_{i,j}$. The task of each ant consists in the construction of a feasible TSP solution, i.e., a feasible tour. In other words, the notion of task of an ant changes from “choosing a path from the nest to the food source” to “constructing a feasible solution to the tackled optimization problem” [16].

Each ant constructs a solution as follows. First, one of the nodes of the TSP graph is randomly chosen as start node. Then, the ant builds a tour in the TSP graph by moving in each construction step from its current node (i.e., the city in which she is located) to another node which she has not visited yet. At each step the traversed edge is added to the solution under construction. When no unvisited nodes are left the ant closes the tour by moving from her current node to the node in which she started the solution construction. This way of constructing a solution implies that an ant has a memory T to store the already visited

nodes. Each solution construction step is performed as follows. Assuming the ant to be in node vi, the subsequent construction step is done with probability



(a) First step of the solution construction.

(b) Second step of the solution construction.

(c) The complete solution after the final construction step.

Figure 2 Example Solution Construction

$$\mathbf{p}(e_{i,j}) = \frac{\tau_{i,j}}{\sum_{\{k \in \{1,\ldots,|V|\} | v_k \notin T\}} \tau_{i,k}}, \quad \forall j \in \{1, \ldots, |V|\},\ v_j \notin T.$$

Once all ants of the colony have completed the construction of their solution, pheromone evaporation is performed

$$\tau_{i,j} \leftarrow (1 - \rho) \cdot \tau_{i,j}, \quad \forall \tau_{i,j} \in \mathcal{T},$$

where T is the set of all pheromone values. Then the ants perform their return trip. Hereby, an ant—having constructed a solution s—performs for each ei,j ∈ s the following pheromone deposit

$$\tau_{i,j} \leftarrow \tau_{i,j} + \frac{Q}{f(s)},$$

where Q is again a positive constant and f (s) is the objective function value of the solution s. As explained in the previous section, the system is iterated—applying na ants per iteration—until a stopping condition (e.g., a time limit) is satisfied.

7.3 Code and Explanation

Code 1 ACO Class Definition

```
#include "ACO.h"

#include <cstdio>
#include <iostream>
#include <cstdlib>

#include <cmath>
#include <limits>
#include <climits>

using namespace std;

ACO::ACO(int nAnts, int nCities,
        double alpha, double beta, double q, double ro, double taumax,
        int initCity) {
        NUMBEROFANTS = nAnts;
        NUMBEROFCITIES = nCities;
        ALPHA = alpha;
        BETA = beta;
        Q = q;
        RO = ro;
        TAUMAX = taumax;
        INITIALCITY = initCity;

        randoms = new Randoms(21);
}
ACO::~ACO() {
        for (int i = 0; i < NUMBEROFCITIES; i++) {
                delete[] GRAPH[i];
                delete[] CITIES[i];
                delete[] PHEROMONES[i];
                delete[] DELTAPHEROMONES[i];
                if (i < NUMBEROFCITIES - 1) {
                        delete[] PROBS[i];
                }
        }
        delete[] GRAPH;
        delete[] CITIES;
        delete[] PHEROMONES;
        delete[] DELTAPHEROMONES;
        delete[] PROBS;
}

void ACO::init() {
        GRAPH = new int*[NUMBEROFCITIES];
        CITIES = new double*[NUMBEROFCITIES];
        PHEROMONES = new double*[NUMBEROFCITIES];
```

```
48          DELTAPHEROMONES = new double*[NUMBEROFCITIES];
49          PROBS = new double*[NUMBEROFCITIES - 1];
50          for (int i = 0; i < NUMBEROFCITIES; i++) {
51                  GRAPH[i] = new int[NUMBEROFCITIES];
52                  CITIES[i] = new double[2];
53                  PHEROMONES[i] = new double[NUMBEROFCITIES];
54                  DELTAPHEROMONES[i] = new double[NUMBEROFCITIES];
55                  PROBS[i] = new double[2];
56                  for (int j = 0; j < 2; j++) {
57                          CITIES[i][j] = -1.0;
58                          PROBS[i][j] = -1.0;
59                  }
60                  for (int j = 0; j < NUMBEROFCITIES; j++) {
61                          GRAPH[i][j] = 0;
62                          PHEROMONES[i][j] = 0.0;
63                          DELTAPHEROMONES[i][j] = 0.0;
64                  }
65          }
66
67          ROUTES = new int*[NUMBEROFANTS];
68          for (int i = 0; i < NUMBEROFANTS; i++) {
69                  ROUTES[i] = new int[NUMBEROFCITIES];
70                  for (int j = 0; j < NUMBEROFCITIES; j++) {
71                          ROUTES[i][j] = -1;
72                  }
73          }
74
75          BESTLENGTH = (double)INT_MAX;
76          BESTROUTE = new int[NUMBEROFCITIES];
77          for (int i = 0; i < NUMBEROFCITIES; i++) {
78                  BESTROUTE[i] = -1;
79          }
80  }
81
82  void ACO::connectCITIES(int cityi, int cityj) {
83          GRAPH[cityi][cityj] = 1;
84          PHEROMONES[cityi][cityj] = randoms->Uniforme() * TAUMAX;
85          GRAPH[cityj][cityi] = 1;
86          PHEROMONES[cityj][cityi] = PHEROMONES[cityi][cityj];
87  }
88  void ACO::setCITYPOSITION(int city, double x, double y) {
89          CITIES[city][0] = x;
90          CITIES[city][1] = y;
91  }
92  void ACO::printPHEROMONES() {
93          cout << " PHEROMONES: " << endl;
94          cout << "   | ";
95          for (int i = 0; i < NUMBEROFCITIES; i++) {
96                  printf("%5d    ", i);
97          }
98          cout << endl << "- | ";
99          for (int i = 0; i < NUMBEROFCITIES; i++) {
100                 cout << "---------";
```

```
101         }
102         cout << endl;
103         for (int i = 0; i < NUMBEROFCITIES; i++) {
104                 cout << i << " | ";
105                 for (int j = 0; j < NUMBEROFCITIES; j++) {
106                         if (i == j) {
107                                 printf("%5s    ", "x");
108                                 continue;
109                         }
110                         if (exists(i, j)) {
111                                 printf("%7.3f ", PHEROMONES[i][j]);
112                         }
113                         else {
114                                 if (PHEROMONES[i][j] == 0.0) {
115                                         printf("%5.0f    ",
116 PHEROMONES[i][j]);
117                                 }
118                                 else {
119                                         printf("%7.3f ",
120 PHEROMONES[i][j]);
121                                 }
122                         }
123                 }
124                 cout << endl;
125         }
126         cout << endl;
127 }
128 double ACO::distance(int cityi, int cityj) {
129         return (double)
130                 sqrt(pow(CITIES[cityi][0] - CITIES[cityj][0], 2) +
131                         pow(CITIES[cityi][1] - CITIES[cityj][1], 2));
132 }
133 bool ACO::exists(int cityi, int cityc) {
134         return (GRAPH[cityi][cityc] == 1);
135 }
136 bool ACO::vizited(int antk, int c) {
137         for (int l = 0; l < NUMBEROFCITIES; l++) {
138                 if (ROUTES[antk][l] == -1) {
139                         break;
140                 }
141                 if (ROUTES[antk][l] == c) {
142                         return true;
143                 }
144         }
145         return false;
146 }
147 double ACO::PHI(int cityi, int cityj, int antk) {
148         double ETAij = (double)pow(1 / distance(cityi, cityj), BETA);
149         double TAUij = (double)pow(PHEROMONES[cityi][cityj], ALPHA);
150 
151         double sum = 0.0;
152         for (int c = 0; c < NUMBEROFCITIES; c++) {
153                 if (exists(cityi, c)) {
```

```
154                         if (!vizited(antk, c)) {
155                                 double ETA = (double)pow(1 /
156 distance(cityi, c), BETA);
157                                 double TAU =
158 (double)pow(PHEROMONES[cityi][c], ALPHA);
159                                 sum += ETA * TAU;
160                         }
161                 }
162         }
163         return (ETAij * TAUij) / sum;
164 }
165 double ACO::length(int antk) {
166         double sum = 0.0;
167         for (int j = 0; j < NUMBEROFCITIES - 1; j++) {
168                 sum += distance(ROUTES[antk][j], ROUTES[antk][j + 1]);
169         }
170         return sum;
171 }
172 int ACO::city() {
173         double xi = randoms->Uniforme();
174         int i = 0;
175         double sum = PROBS[i][0];
176         while (sum < xi) {
177                 i++;
178                 sum += PROBS[i][0];
179         }
180         return (int)PROBS[i][1];
181 }
182 void ACO::route(int antk) {
183         ROUTES[antk][0] = INITIALCITY;
184         for (int i = 0; i < NUMBEROFCITIES - 1; i++) {
185                 int cityi = ROUTES[antk][i];
186                 int count = 0;
187                 for (int c = 0; c < NUMBEROFCITIES; c++) {
188                         if (cityi == c) {
189                                 continue;
190                         }
191                         if (exists(cityi, c)) {
192                                 if (!vizited(antk, c)) {
193                                         PROBS[count][0] = PHI(cityi, c,
194 antk);
195                                         PROBS[count][1] = (double)c;
196                                         count++;
197                                 }
198
199                         }
200                 }
201                 // deadlock
202                 if (0 == count) {
203                         return;
204                 }
205                 ROUTES[antk][i + 1] = city();
206         }
```

```
}
int ACO::valid(int antk, int iteration) {
	for (int i = 0; i < NUMBEROFCITIES - 1; i++) {
		int cityi = ROUTES[antk][i];
		int cityj = ROUTES[antk][i + 1];
		if (cityi < 0 || cityj < 0) {
			return -1;
		}
		if (!exists(cityi, cityj)) {
			return -2;
		}
		for (int j = 0; j < i - 1; j++) {
			if (ROUTES[antk][i] == ROUTES[antk][j]) {
				return -3;
			}
		}
	}
	if (!exists(INITIALCITY, ROUTES[antk][NUMBEROFCITIES - 1])) {
		return -4;
	}
	return 0;
}
void ACO::printGRAPH() {
	cout << " GRAPH: " << endl;
	cout << "   | ";
	for (int i = 0; i < NUMBEROFCITIES; i++) {
		cout << i << " ";
	}
	cout << endl << "- | ";
	for (int i = 0; i < NUMBEROFCITIES; i++) {
		cout << "- ";
	}
	cout << endl;
	int count = 0;
	for (int i = 0; i < NUMBEROFCITIES; i++) {
		cout << i << " | ";
		for (int j = 0; j < NUMBEROFCITIES; j++) {
			if (i == j) {
				cout << "x ";
			}
			else {
				cout << GRAPH[i][j] << " ";
			}
			if (GRAPH[i][j] == 1) {
				count++;
			}
		}
		cout << endl;
	}
	cout << endl;
	cout << "Number of connections: " << count << endl << endl;
}
void ACO::printRESULTS() {
```

```
260         BESTLENGTH += distance(BESTROUTE[NUMBEROFCITIES - 1],
261 INITIALCITY);
262         cout << " BEST ROUTE:" << endl;
263         for (int i = 0; i < NUMBEROFCITIES; i++) {
264                 cout << BESTROUTE[i] << " ";
265         }
266         cout << endl << "length: " << BESTLENGTH << endl;
267 
268         cout << endl << " IDEAL ROUTE:" << endl;
269         cout << "0 7 6 2 4 5 1 3" << endl;
270         cout << "length: 127.509" << endl;
271 }
272 void ACO::updatePHEROMONES() {
273         for (int k = 0; k < NUMBEROFANTS; k++) {
274                 double rlength = length(k);
275                 for (int r = 0; r < NUMBEROFCITIES - 1; r++) {
276                         int cityi = ROUTES[k][r];
277                         int cityj = ROUTES[k][r + 1];
278                         DELTAPHEROMONES[cityi][cityj] += Q / rlength;
279                         DELTAPHEROMONES[cityj][cityi] += Q / rlength;
280                 }
281         }
282         for (int i = 0; i < NUMBEROFCITIES; i++) {
283                 for (int j = 0; j < NUMBEROFCITIES; j++) {
284                         PHEROMONES[i][j] = (1 - RO) * PHEROMONES[i][j] +
285 DELTAPHEROMONES[i][j];
286                         DELTAPHEROMONES[i][j] = 0.0;
287                 }
288         }
289 }
290 void ACO::optimize(int ITERATIONS) {
291         for (int iterations = 1; iterations <= ITERATIONS;
292 iterations++) {
293                 cout << flush;
294                 cout << "ITERATION " << iterations << " HAS STARTED!" <<
295 endl << endl;
296 
297                 for (int k = 0; k < NUMBEROFANTS; k++) {
298                         cout << " : ant " << k << " has been released!"
299 << endl;
300                         while (0 != valid(k, iterations)) {
301                                 cout << "  :: releasing ant " << k << "
302 again!" << endl;
303                                 for (int i = 0; i < NUMBEROFCITIES; i++)
304 {
305                                         ROUTES[k][i] = -1;
306                                 }
307                                 route(k);
308                         }
309 
310                         for (int i = 0; i < NUMBEROFCITIES; i++) {
311                                 cout << ROUTES[k][i] << " ";
312                         }
```

```
313                         cout << endl;
314
315                         cout << "  :: route done" << endl;
316                         double rlength = length(k);
317
318                         if (rlength < BESTLENGTH) {
319                                 BESTLENGTH = rlength;
320                                 for (int i = 0; i < NUMBEROFCITIES; i++)
321 {
322                                         BESTROUTE[i] = ROUTES[k][i];
323                                 }
324                         }
325                         cout << " : ant " << k << " has ended!" << endl;
326                 }
327
328                 cout << endl << "updating PHEROMONES . . .";
329                 updatePHEROMONES();
330                 cout << " done!" << endl << endl;
331                 printPHEROMONES();
332
333                 for (int i = 0; i < NUMBEROFANTS; i++) {
334                         for (int j = 0; j < NUMBEROFCITIES; j++) {
335                                 ROUTES[i][j] = -1;
336                         }
337                 }
338
339                 cout << endl << "ITERATION " << iterations << " HAS
340 ENDED!" << endl << endl;
341         }
342 }
```

Code 2 ACO Header File

```
1  #include "Randoms.cpp"
2
3  class ACO {
4  public:
5          ACO (int nAnts, int nCities,
6                  double alpha, double beta, double q, double ro, double
7  taumax,
8                  int initCity);
9          virtual ~ACO ();
10
11         void init ();
12
13         void connectCITIES (int cityi, int cityj);
14         void setCITYPOSITION (int city, double x, double y);
15         void printPHEROMONES ();
16         void printGRAPH ();
17         void printRESULTS ();
18         void optimize (int ITERATIONS);
19
```

```
private:
        double distance (int cityi, int cityj);
        bool exists (int cityi, int cityc);
        bool vizited (int antk, int c);
        double PHI (int cityi, int cityj, int antk);
        double length (int antk);
        int city ();
        void route (int antk);
        int valid (int antk, int iteration);

        void updatePHEROMONES ();

        int NUMBEROFANTS, NUMBEROFCITIES, INITIALCITY;
        double ALPHA, BETA, Q, RO, TAUMAX;

        double BESTLENGTH;

        int **GRAPH, **ROUTES;
        double **CITIES, **PHEROMONES, **DELTAPHEROMONES, **PROBS;
        Randoms *randoms;
};

```

Code 3 Main function

```
#include <iostream>
#include <cstdlib>

#include <cmath>
#include <limits>
#include <climits>

#include "ACO.h"

// Following are the declarations of various constants used throughout
the program

#define ITERATIONS              (int) 5

#define NUMBEROFANTS    (int) 4
#define NUMBEROFCITIES (int) 8

#define ALPHA                   (double) 0.5

#define BETA                    (double) 0.8

#define Q                       (double) 80

#define RO                      (double) 0.2

#define TAUMAX          (int) 2

```

```
28 #define INITIALCITY            (int) 0
29
30 int main() {
31
32         ACO *ANTS = new ACO (NUMBEROFANTS, NUMBEROFCITIES,
33                                                 ALPHA, BETA, Q, RO,
34 TAUMAX, INITIALCITY);
35
36         ANTS -> init();
37
38         ANTS -> connectCITIES (0, 1);
39         ANTS -> connectCITIES (0, 2);
40         ANTS -> connectCITIES (0, 3);
41         ANTS -> connectCITIES (0, 7);
42         ANTS -> connectCITIES (1, 3);
43         ANTS -> connectCITIES (1, 5);
44         ANTS -> connectCITIES (1, 7);
45         ANTS -> connectCITIES (2, 4);
46         ANTS -> connectCITIES (2, 5);
47         ANTS -> connectCITIES (2, 6);
48         ANTS -> connectCITIES (4, 3);
49         ANTS -> connectCITIES (4, 5);
50         ANTS -> connectCITIES (4, 7);
51         ANTS -> connectCITIES (6, 7);
52
53         ANTS -> setCITYPOSITION (0,  1,  1);
54         ANTS -> setCITYPOSITION (1, 10, 10);
55         ANTS -> setCITYPOSITION (2, 20, 10);
56         ANTS -> setCITYPOSITION (3, 10, 30);
57         ANTS -> setCITYPOSITION (4, 15,  5);
58         ANTS -> setCITYPOSITION (5, 10,  1);
59         ANTS -> setCITYPOSITION (6, 20, 20);
60         ANTS -> setCITYPOSITION (7, 20, 30);
61         ANTS -> printGRAPH ();
62         ANTS -> printPHEROMONES ();
63
64         ANTS -> optimize (ITERATIONS);
65
66         ANTS -> printRESULTS ();
67
68         return 0;
69 }
70
```

The main function defines several constants which are held throughout the program execution. These constants include value of ITERATIONS, NUMBEROFANTS, NUMBEROFCITIES, ALPHA, BETA, Q, RO, TAUMAX, and INITIALCITY.

The main function declares an object of the ACO class and initializes the graph using init() function for operations. It then adds 14 connections between cities using the connectCITIES

function provided by the ACO class. It also sets the position of all the cities in the graph using setCITYPOSITION function.

It calls the printGRAPH function for letting the user verify his inputs. Then it calls the optimize function which executes the ACO algorithm. This is followed by printing of the results which is followed by program termination.

Program Output

Initialization

The cities graph defined in the main function is depicted in the form of a table below where every city is numbered from 0 to 7. Initial pherome values assigned to the each of the path is also illustrated in table format.

```
ayush@ayush-Lenovo-ideapad-300-15ISK:~/Documents/aco/src$ g++ -Wall *.cpp -o aco
ayush@ayush-Lenovo-ideapad-300-15ISK:~/Documents/aco/src$ ./aco
 GRAPH:
  | 0 1 2 3 4 5 6 7
- | - - - - - - - -
0 | x 1 1 1 0 0 0 1
1 | 1 x 0 1 0 1 0 1
2 | 1 0 x 0 1 1 1 0
3 | 1 1 0 x 1 0 0 0
4 | 0 0 1 1 x 1 0 1
5 | 0 1 1 0 1 x 0 0
6 | 0 0 1 0 0 0 x 1
7 | 1 1 0 0 1 0 1 x

Number of connections: 28

 PHEROMONES:
  |     0       1       2       3       4       5       6       7
- | ------------------------------------------------------------------
0 |     x     0.103   1.624   0.245     0       0       0     0.571
1 |   0.103     x       0     0.233     0     1.641     0     1.452
2 |   1.624     0       x       0     0.188   1.053   1.078     0
3 |   0.245   0.233     0       x     0.737     0       0       0
4 |     0       0     0.188   0.737     x     1.472     0     1.158
5 |     0     1.641   1.053     0     1.472     x       0       0
6 |     0       0     1.078     0       0       0       x     0.844
7 |   0.571   1.452     0       0     1.158     0     0.844     x
```

The algorithm progresses in several iterations as explained in theory. We chose 5 iterations for this example. The following output snippets displays progress information of each iteration through suitable messages. This information includes each ant's release and ending cycle along with the final path taken by each ant.

We have also included the resulting pherome values after every iteration in a table format.

Iteration 1

```
ITERATION 1 HAS STARTED!

 : ant 0 has been released!
  :: releasing ant 0 again!
  :: releasing ant 0 again!
  :: releasing ant 0 again!
  :: releasing ant 0 again!
0 2 6 7 4 5 1 3
  :: route done
 : ant 0 has ended!
 : ant 1 has been released!
  :: releasing ant 1 again!
0 1 7 6 2 5 4 3
  :: route done
 : ant 1 has ended!
 : ant 2 has been released!
  :: releasing ant 2 again!
  :: releasing ant 2 again!
  :: releasing ant 2 again!
  :: releasing ant 2 again!
0 2 6 7 1 5 4 3
  :: route done
 : ant 2 has ended!
 : ant 3 has been released!
  :: releasing ant 3 again!
  :: releasing ant 3 again!
  :: releasing ant 3 again!
0 2 6 7 4 5 1 3
  :: route done
 : ant 3 has ended!

updating PHEROMONES . . . done!

 PHEROMONES:
  |     0       1       2       3       4       5       6       7
- | -------------------------------------------------------------------
0 |     x     0.879   3.636   0.196     0       0       0     0.457
1 |   0.879     x       0     1.757     0     3.650     0     2.725
2 |   3.636     0       x       0     0.151   1.639   3.996     0
3 |   0.196   1.757     0       x     2.153     0       0       0
4 |     0       0     0.151   2.153     x     4.311     0     2.496
5 |     0     3.650   1.639     0     4.311     x       0       0
6 |     0       0     3.996     0       0       0       x     3.809
```

Iteration 2

```
ITERATION 2 HAS STARTED!

 : ant 0 has been released!
  :: releasing ant 0 again!
  :: releasing ant 0 again!
  :: releasing ant 0 again!
  :: releasing ant 0 again!
  :: releasing ant 0 again!
0 2 6 7 4 5 1 3
  :: route done
 : ant 0 has ended!
 : ant 1 has been released!
  :: releasing ant 1 again!
0 2 6 7 1 5 4 3
  :: route done
 : ant 1 has ended!
 : ant 2 has been released!
  :: releasing ant 2 again!
0 2 6 7 1 5 4 3
  :: route done
 : ant 2 has ended!
 : ant 3 has been released!
  :: releasing ant 3 again!
  :: releasing ant 3 again!
  :: releasing ant 3 again!
0 1 5 2 6 7 4 3
  :: route done
 : ant 3 has ended!

updating PHEROMONES . . . done!

 PHEROMONES:
  |     0       1       2       3       4       5       6       7
- | ------------------------------------------------------------------------
0 |     x     1.457   5.228   0.157     0       0       0     0.365
1 |   1.457     x       0     2.190     0     5.992     0     3.714
2 |   5.228     0       x       0     0.120   2.064   6.270     0
3 |   0.157   2.190     0       x     4.010     0       0       0
4 |     0       0     0.120   4.010     x     5.768     0     3.535
5 |     0     5.992   2.064     0     5.768     x       0       0
6 |     0       0     6.270     0       0       0       x     6.120
7 |   0.365   3.714     0       0     3.535     0     6.120     x
```

Iteration 3

We can observe that pheromes have started to concentrate on the most travelled path.

```
ITERATION 3 HAS STARTED!

 : ant 0 has been released!
  :: releasing ant 0 again!
  :: releasing ant 0 again!
0 7 6 2 5 4 3 1
  :: route done
 : ant 0 has ended!
 : ant 1 has been released!
  :: releasing ant 1 again!
  :: releasing ant 1 again!
0 2 6 7 4 5 1 3
  :: route done
 : ant 1 has ended!
 : ant 2 has been released!
  :: releasing ant 2 again!
  :: releasing ant 2 again!
  :: releasing ant 2 again!
  :: releasing ant 2 again!
  :: releasing ant 2 again!
0 1 3 4 5 2 6 7
  :: route done
 : ant 2 has ended!
 : ant 3 has been released!
  :: releasing ant 3 again!
  :: releasing ant 3 again!
0 7 6 2 4 5 1 3
  :: route done
 : ant 3 has ended!

updating PHEROMONES . . . done!

 PHEROMONES:
  |     0       1       2       3       4       5       6       7
- | -------------------------------------------------------------------
0 |     x     1.981   4.967   0.126     0       0       0     1.782
1 |   1.981     x       0     4.843     0     6.402     0     2.972
2 |   4.967     0       x       0     0.920   3.134   8.106     0
3 |   0.126   4.843     0       x     4.691     0       0       0
4 |     0       0     0.920   4.691     x     7.705     0     3.613
5 |     0     6.402   3.134     0     7.705     x       0       0
6 |     0       0     8.106     0       0       0       x     7.987
7 |   1.782   2.972     0       0     3.613     0     7.987     x
```

Iteration 4

Some of the most travelled path visible from the table are

1. 6 to 2 (Pherome Value : 9.558),
2. 7 to 6 (Pherome Value : 9.462),
3. 4 to 5 (Pherome Value : 8.483),
4. 5 to 1 (Pherome Value : 8.195) etc.

We can expect some of these to be included in the final optimal route predicted by the ACO algorithm.

```
  :: releasing ant 0 again!
  :: releasing ant 0 again!
  :: releasing ant 0 again!
  :: releasing ant 0 again!
  :: releasing ant 0 again!
0 2 6 7 1 5 4 3
  :: route done
 : ant 0 has ended!
 : ant 1 has been released!
  :: releasing ant 1 again!
0 2 6 7 4 5 1 3
  :: route done
 : ant 1 has ended!
 : ant 2 has been released!
  :: releasing ant 2 again!
0 2 6 7 1 5 4 3
  :: route done
 : ant 2 has ended!
 : ant 3 has been released!
  :: releasing ant 3 again!
  :: releasing ant 3 again!
  :: releasing ant 3 again!
  :: releasing ant 3 again!
  :: releasing ant 3 again!
  :: releasing ant 3 again!
0 1 5 2 6 7 4 3
  :: route done
 : ant 3 has ended!

updating PHEROMONES . . . done!

 PHEROMONES:
   |     0       1       2       3       4       5       6       7
 - | ---------------------------------------------------------------
 0 |     x     2.338   6.293   0.101     0       0       0     1.426
 1 |   2.338     x       0     4.659     0     8.195     0     3.912
 2 |   6.293     0       x       0     0.736   3.261   9.558     0
 3 |   0.101   4.659     0       x     6.040     0       0       0
 4 |     0       0     0.736   6.040     x     8.483     0     4.429
 5 |     0     8.195   3.261     0     8.483     x       0       0
 6 |     0       0     9.558     0       0       0       x     9.462
 7 |   1.426   3.912     0       0     4.429     0     9.462     x
```

Iteration 5

```
ITERATION 5 HAS STARTED!

 : ant 0 has been released!
  :: releasing ant 0 again!
0 2 6 7 1 5 4 3
  :: route done
 : ant 0 has ended!
 : ant 1 has been released!
  :: releasing ant 1 again!
0 1 3 4 5 2 6 7
  :: route done
 : ant 1 has ended!
 : ant 2 has been released!
  :: releasing ant 2 again!
0 1 5 2 6 7 4 3
  :: route done
 : ant 2 has ended!
 : ant 3 has been released!
  :: releasing ant 3 again!
  :: releasing ant 3 again!
0 2 6 7 4 5 1 3
  :: route done
 : ant 3 has ended!

updating PHEROMONES . . . done!

 PHEROMONES:
  |     0       1       2       3       4       5       6       7
- | ------------------------------------------------------------------------
0 |     x     3.440   6.586   0.080     0       0       0     1.141
1 |   3.440     x       0     5.328     0     8.861     0     3.896
2 |   6.586     0       x       0     0.589   4.178   10.767    0
3 |   0.080   5.328     0       x     7.168     0       0       0
4 |     0       0     0.589   7.168     x     9.154     0     5.082
5 |     0     8.861   4.178     0     9.154     x       0       0
6 |     0       0    10.767     0       0       0       x    10.691
7 |   1.141   3.896     0       0     5.082     0    10.691     x
```

Final Outcome

The Ant Colony optimization(ACO) algorithm's best path has length of 127.509 and follows the Route along the cities 0 7 6 2 4 5 1 3. Regular TSP algorithm also gave the same result.

```
 BEST ROUTE:
0 7 6 2 4 5 1 3
length: 127.509

 IDEAL ROUTE:
0 7 6 2 4 5 1 3
length: 127.509
```

Thus, we can conclude that ACO can reduce the complexity of the TSP algorithm without degrading the quality of the solution.

8. OTHER ACO APPLICATIONS

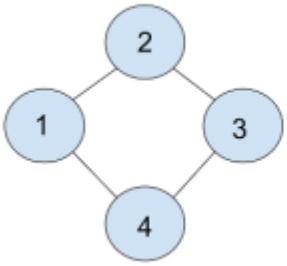
8.1. IN TELECOMMUNICATION NETWORKS

Unsuccessful call connections in telecommunication networks are frequently blamed on insufficient approaches to routing problems. ACO algorithms have been applied to counter this problem, particularly when key aspects of the network vary over time, as is the case when the call volume grows unusually high, for example. The concept was first introduced in telephone networks [23], and developed to the point of being state-of-the-art in wired networks, rivalling and outperforming former mainstays in the field.

8.8.1 Load Balancing Networks

Switch-based telecommunication networks, such as telephone networks, can be represented by an undirected graph. Each node represents the switching station, while the edges between nodes represent communication channels between callers. When congestion occurs in the network, load balancing distributes activity evenly over the nodes with the aim to minimize the number of lost calls [23]. A node tends to be linked only to its geographical neighbors. Each node has the following key attributes:

- Capacity - the amount of simultaneous calls the node can handle.
- Routing Table - each entry informs which is the next node on the route to the destination
- node.
- A probability of being the source or destination node of a call.



Destination Node	Next Node 2	Next Node 4
2	0.91	0.09
3	0.44	0.56
4	0.09	0.91

Upon implementing an ant-based control mechanism on the network model, the routing tables are replaced by “pheromone tables”. Every node has a pheromone table for every possible destination in the network, and each table has an entry for every neighbor. After initializing an ant at a given node, it moves from node to node according to the probabilities in the pheromone tables for the destination node. At the next node, the ant updates the probabilities of that node’s pheromone table entries corresponding to their source node. The table is altered to increase the probability of returning to previous nodes.

$$p_{new} = \frac{p_{old} + \Delta p}{1 + \Delta p},$$

where Δp is the probability increase. The other entries are reduced so that the sum of the entries is 1. In order to encourage the ant to find routes that are relatively short, Δp is chosen so that is reduces with the age of the ant. This bias the network in favor of ants that have taken shorter tours. Upon returning to the source node, the ant dies.

Calls from a source node will choose the route to its destinations by means of the entries in the pheromone table. Calls and ants interact with each other through these probabilities. Ants will influence the routes by the update of the pheromone tables, which in turn will influence the route taken by a call

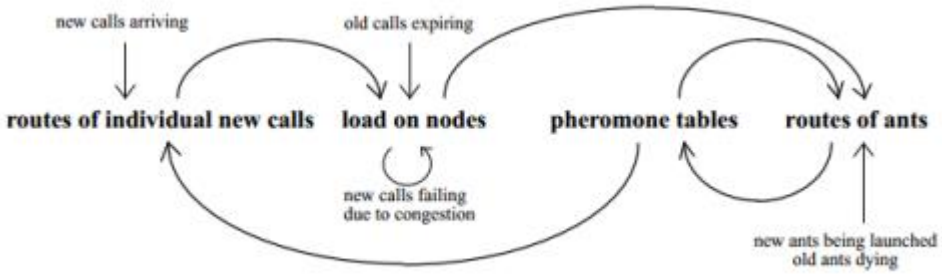


Figure 3the relationship between calls, nodes, pheromone tables and ants

8.8.2 Scheduling Problems

Scheduling problems concern the assignment of jobs to one or various machines over time. Input data for these problems are processing times but also often additional setup times, release dates and due dates of jobs, measures for the jobs' importance and precedence constraints among jobs. Scheduling problems have been an important application area of ACO algorithms, and the currently available ACO applications in scheduling deal with many different job and machine characteristics

The single-machine total weighted tardiness problem (SMTWTP) has been tackled by using variants of ACS (ACS-SMTWTP). In ACS-SMTWTP, a solution is determined by a sequence of jobs. The positions of the sequence are filled in their canonical order, that is, first a job is assigned to position one, next a job to position two, and so on, until position n. Pheromone trails are defined as the desirability of scheduling job j at position i, a pheromone trail definition that is used in many ACO applications to scheduling problems [24]. Besten et al. [24] combined ACS-SMTWTP with a powerful local search algorithm, resulting in one of the best algorithms available for this problem in terms of solution quality.

8.8.3 Machine Learning Problems

Diverse problems in the field of Machine Learning have been tackled by means of ACO algorithms. Notable examples are the work of Parpinelli et al. [25]and Martens et al. [26]on applying ACO to the problem of learning classification rules. This work was later extended by Otero et al. [27] in order to handle continuous attributes. De Campos et al. [25] adapted Ant Colony System for the problem of learning the structure of Bayesian networks, and Pinto et al. [28]have recently extended this work. Finally, the work of Socha & Blum

[21]for training neural networks by means of ACO is also an example of the application of ACO algorithms for continuous problems.

8.8.4 Other problems

Scheduling problems

- Job-shop scheduling problem (JSP)
- Open-shop scheduling problem (OSP)
- Permutation flow shop problem (PFSP)
- Single machine total tardiness problem (SMTTP)
- Single machine total weighted tardiness problem (SMTWTP)
- Resource-constrained project scheduling problem (RCPSP)
- Group-shop scheduling problem (GSP)
- Single-machine total tardiness problem with sequence dependent setup times (SMTTPDST)
- Multistage flow shop scheduling problem (MFSP) with sequence dependent setup/changeover times

Vehicle routing problem

- Capacitated vehicle routing problem (CVRP)
- Multi-depot vehicle routing problem (MDVRP)
- Period vehicle routing problem (PVRP)
- Split delivery vehicle routing problem (SDVRP)
- Stochastic vehicle routing problem (SVRP)
- Vehicle routing problem with pick-up and delivery (VRPPD)
- Vehicle routing problem with time windows (VRPTW)
- Time dependent vehicle routing problem with time windows (TDVRPTW)
- Vehicle routing problem with time windows and multiple service workers (VRPTWMS)

Assignment problem

- Quadratic assignment problem (QAP)
- Generalized assignment problem (GAP)
- Frequency assignment problem (FAP)
- Redundancy allocation problem (RAP)

Set problem

- Set cover problem (SCP)
- Partition problem (SPP)
- Weight constrained graph tree partition problem (WCGTPP)
- Arc-weighted l-cardinality tree problem (AWlCTP)
- Multiple knapsack problem (MKP)
- Maximum independent set problem (MIS)

REFERENCES

[1] M. Dorigo, "Ant Algorithm," 2007. [Online]. Available: http://dx.doi.org/10.4249/scholarpedia.1461.

[2] L. J. R. K. A. S. D. Lawler E, The travelling Salesman problem, John Wiley & Sons, 1985.

[3] H. H. Stützle T, "MAX–MIN Ant system," *Future Generat Comput Syst ,* 2000.

[4] S. T. Dorigo M, Ant Colony optimization, MIT Press, 2004.

[5] D. M. T. G. Bonabeau E, Swarm intelligence: From natural to artificial systems, Oxford University Press.

[6] R. A. Blum C, ". Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput Surveys,* 2003.

[7] V. M. a. A. C. M. Dorigo, "Positive feedback as a search strategy," Dipartimento di Elettronica, Politecnico di Milano, 1991.

[8] V. M. a. A. C. M. Dorigo, "Ant System: Optimization by a colony of cooperating agents," in *IEEE Transactions on Systems, Man, and Cybernetics*, 1996.

[9] M. D. a. L. M. Gambardella, "Ant Colony System: A cooperative learning approach to the traveling salesman problem," in *IEEE Transactions on Evolutionary Computation*, 1997.

[10] M. Dorigo., "Optimization, Learning and Natural Algorithms," Dipartimento di Elettronica, Politecnico di Milano, Milan.

[11] A.-H. R. H. H. Shmygelska A, "An ant colony optimization algorithm for the 2D HP protein folding problem," in *Proceedings of ANTS 2002—Third international workshop. Lecture Notes in Comput Sci,*.

[12] J. C. Moss JD, "An ant colony algorithm for multiple sequence alignment in bioinformatics.," *Artificial neural networks and genetic algorithms.*

[13] S. J. D. Y. Karpenko O, "Prediction of MHC class II binders using the ant colony search strategy," *Artificial Intelligence in Medicine,* 2005.

[14] D. M. Gambardella LM, "Ant Colony System hybridized with a new local search for the sequential ordering problem," *m. INFORMS J Comput.*

[15] M. M. S. H. Merkle D, "Ant colony optimization for resource-constrained project scheduling," in *IEEE Trans Evolutionary Computing*.

[16] C. Blum, "Ant colony optimization: Introduction and recent trends," *Physics of Life Reviews 2*, 2005.

[17] P. I. Bilchev B, "The ant colony metaphor for searching continuous design spaces.," in . *Proceedings of the AISB*, 1995.

[18] V. G. S. M. Monmarché N, "On how pachycondyla apicalis ants suggest a new search algorithm," *Future Generation Comput Syst*, 2000.

[19] S. P. Dréo J, "A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions," in *Ant algorithms—Proceedings of ANTS 2002—Third international workshop*, 2002.

[20] M. M. Box GEP, A note on the generation of random normal deviates. Ann Math Statist, 1958.

[21] S. K. Blum C, "Training feed-forward neural networks with ant colony optimization: An application to pattern classification," in *5th international conference on hybrid intelligent systems (HIS)*, 2005.

[22] S. K., "ACO for continuous and mixed-variable optimization.," in *ANTS 2004—Fourth international workshop on Ant colony optimization and swarm intelligence*.

[23] O. E. H. J. L. B. a. L. J. R. R. Schoonderwoerd, "Ant-based load balancing in telecommunications networks," in *Adaptive behavior*, 1997.

[24] S. T. D. M. den Besten ML, "Ant colony optimization for the total ¨ weighted tardiness problem," in *Sixth International Conference on Parallel Problem Solving from Nature.*, 2000.

[25] G. J. P. J. de Campos LM, "Learning Bayesian networks by ACO Searching in the space of orderings.," *Mathware & Soft Computing*, 2002.

[26] D. B. M. H. R. V. J. S. M. B. B. Martens D, "Classification with ACO," in *IEEE Transactions on Evolutionary Computing*, 2007.

[27] F. A. J. C. c. Otero FEB, "Ant-Miner: An ant colony classification algorithm to cope with continuous attributes," in *6th International Conference, ANTS 2008*, 2008.

[28] N. A. D. M. R. T. S. J. Pinto PC, "Using a local discovery ant algorithm for bayesian network structure learning.," in *IEEE Transactions on Evolutionary Computation*, 2009.