CHAPTER 7

Artificial Bee and Ant Colony Optimization

7.1 INTRODUCTION

Both the Ant Colony Optimization algorithm and the Artificial Bee Colony algorithm involve modeling of the co-operation between insects to obtain an optimum outcome for the colony as a whole. The Ant Colony optimization method has been particularly effective in solving the traveling sales person problem (TSP), a very difficult (formally an NP hard) problem. Consequently we will discuss the TSP problem and its solution using Ant Colony Optimization (ACO). Ants convey information between members of the colony using pheromones. The ABC algorithm also involves co-operation and the bee dance or waggle dance is used to convey important information to the other bees of the hive by returning bees regarding the location of nectar. This method is directly applicable to the solution of nonlinear optimization problems, rather than the TSP problem.

7.2 THE ARTIFICIAL BEE COLONY ALGORITHM (ABC)

This algorithm was introduced by Karaboga (2005) and developed in Karaboga and Basturk (2007). It involves a cooperative swarming method inspired by the behavior of honey bees and the method has had considerable success in its application to the solution of difficult global optimization problems. In the ABC algorithm the locations of the food source, nectar, represent possible solutions of the optimization problem and the amount of nectar at these positions the quality or fitness of the solution. The bees in the colony are the key to locating these positions instinctively they achieve this in the most efficient or optimal way. To do this the bee colony is divided into three groups: onlookers, employed and scouts. Half the bees are employed and the other half are onlooker bees. Scout bees are introduced as the process evolves. For every nectar food source there is an employed bee. An employed bee can search the local area for the nectar sources. It visits a source until the source is exhausted. Then that bee is designated as a scout bee and performs a random search of the whole area. Once a scout bee finds a new source of nectar, its location is memorized by the individual bee and the new location replaces the exhausted one. After each employed bee has completed its search cycle it exchanges this information with the onlooker bees. In a real bee colony this is achieved by a bee "waggle dance", remarkably this formalized dance can convey to the

Introduction to Nature-Inspired Optimization
DOI: 10.1016/B978-0-12-803636-5.00007-4

onlookers the direction, distance and quality of the nectar source. The onlooker bees analyze all the information on position and nectar amounts and decide on the most fruitful areas. They then fly to the most promising nectar sources. These concepts can be utilized in the development of an optimization algorithm. We now provide a more formal description of this process.

Initially a set of candidate solutions are generated randomly within the solution space each one associated with an employed bee.

At each iteration the employed bee tries to discover a new possible solution by searching its neighborhood this can be formulated using the equation:

$$v_{ij} = x_{ij} + r_u(x_{ij} - x_{kj}) \tag{7.1}$$

where r_u is a uniformly distributed random number in the range $[-1, 1]$ and is a different random number for every value of i and j, k and j are randomly selected from the set of indices of employed bees and the dimension indexes of the problem, respectively. The variable x_{ij} is the current solution and v_{ij} the candidate improved solution. The subscript i denotes the employed bee and j the particular dimension. The new value is checked for improvement using the fitness function if there is an improvement this employed bee moves to this new source of nectar.

Once this process is complete the information from each of the employed bees is presented to the onlooker bees who make an overall judgment of these nectar sources. They then choose one with a probability p_i based on the relative value of the particular source compared with all the other sources. This is implemented by computing the values, called fitness values from the original objective function of the problem $f(x)$, using (7.2).

$$fit_i = \begin{cases} \frac{1}{(1+f(x_i))} & \text{if } f(x_i) \geq 0 \\ 1 + \text{abs}(f(x_i)) & \text{otherwise} \end{cases} \tag{7.2}$$

and then the values of the probabilities are calculated from

$$p_i = \frac{fit_i}{\sum_{i=1}^{m} fit_i} \tag{7.3}$$

where fit_i is the fitness of solution i, the summation is for all employed bees. These values of p_i are in the range $[0, 1]$. An alternative method suggested for calculating the probability values uses the formula:

$$p_i = 0.9\frac{fit_i}{fit_{best}} + 0.1 \tag{7.4}$$

where fit_{best} is the best fitness value. Dependent on the probability values new points are generated using (7.1). An important feature of the algorithm is to abandon solutions that

are not fruitful in that they have not provided an improvement in the objective function value after a number of cycles. This is implemented by setting a limit parameter which is preset by the user. If the number of times there has been no improvement at that location exceeds this limit value this location is abandoned and replaced with new randomly generated solution within the solution space using (7.5). This can be considered as the exploratory stage and models the role of the scout bee:

$$x_{i,j} = r_{min,j} + r_u(r_{max,j} - r_{min,j}) \tag{7.5}$$

where r_u is a uniform random number in the range [0, 1]. The value of limit should be carefully considered and its setting should relate to the difficulty of the particular problem; e.g. it should increase with the size or complexity of the problem. This process is repeated at each cycle until the required number of iterations have been performed. We are now in a position to provide a pseudo code formulation of the method on which its implementation can be based. This is shown in **Algorithm 6**.

This completes the general description of the method and many studies have reported good comparative results for this algorithm. Various suggestions have been made by different researchers about the values of the given constants and other features of the ABC algorithm and we shall discuss a selection of modifications of the basic algorithm in the next section.

7.3 MODIFICATIONS OF THE ARTIFICIAL BEE COLONY (ABC) ALGORITHM

The first modification we shall consider for the ABC algorithm is that suggested by Li et al. (2014). Li et al. point out that there algorithm is a further development of work published in an earlier paper (Li et al., 2014) and they begin their discussion of improvements to the original algorithm with a description of this work which we will now describe.

The algorithm introduced in 2014 uses the vector **trial** which keeps count of the number of times an employed bee has failed to find an improved location for a nectar resource. The values of the elements of this vector **trial** are used to guide the process of exploration and exploitation that are found in the original ABC algorithm and may be described as follows:

$$x^*_{ij} = x_{ij} + r_u(x_{ij} - x_{kj})\gamma_i \tag{7.6}$$

where r_u is a uniform random number in the range -1 to 1 and is different for each value of i and j, and γ_i is defined by the equation

$$\gamma_i = \exp\left(-(trial_i - 1)\frac{\log_e 10}{(n_{var} - 1)}\right)$$

Algorithm 6 Basic ABC.

Set cycle =1; Set constant parameters: maxcycle, N_s, limit, where N_s is the number of nectar sources.
Generate initial bee population randomly within the defined solution space for the specific problem
while *cycle* < *maxcycle* **do**
  Calculate the new locations $v_{i,j}$ for the employed bees using
  for $i = 1$ **to** N_s **do**
    for $j = 1$ **to** nv **do**
      Compute $v_{i,j}$ from (7.1)
    end for
    Where k and j are randomly selected integers in the appropriate range. Note that $k \neq i$.
    Evaluate the fitness of the new sources based on the objective function
    if $fit(\mathbf{v}_i) > fit(\mathbf{x}_i)$ **then**
      $\mathbf{v}_i$ replaces $\mathbf{x}_i$.
    end if
  end for
  for $i = 1$ **to** N_s **do**
    Compute p_i from (7.3) where fit_i is given by (7.2)
  end for
  The onlooker bees select new solutions v_i based on the probabilities p_i and evaluates them using the fitness function and replace the old solutions with a new one if an improvement is achieved. If for a particular source s no improvement is achieved keep a record of that by setting $trial(s) = trial(s) + 1$. Otherwise the value of $trial(s)$ is set to zero.
  if Fitness of a nectar source has not improved for greater than *limit* iterations **then**
    abandon it and replace it by a randomly selected source location in the solution space using (7.5) this models the role of the scout in exploring new areas.
  end if
  Store the best solution so far.
  $cycle = cycle + 1$
end while

Now the value of each *trial* lies between 1 and n_{var}, where n_{var} is the number variables in the problem. This means that the value of γ_i varies between 0.1 and 1 because for each employed bee i, $trial_i$ may vary between n_{var} and 1, since if $trial_i = 1$ then $\gamma_i = 1$ and if $trial_i = n_{var}$ then $\gamma_i = \exp(-\log_e 10) = 1/\exp(\log_e 10) = 0.1$. This means the random element of exploration is reduced if failures are becoming high, thus intensifying the

search in the local region before it is abandoned. The next development described by Li et al. which constitutes their new algorithm also introduces a parameter that varies through the generations but this is based on more complex use of the trial vector. This introduces the update equation:

$$x_{ij}^{*} = x_{mj} + r_u(x_{kj} - x_{ij})\mu_i \tag{7.7}$$

where r_u is a uniformly distributed random number in the range [0, 1]. (7.7) is similar to (7.6) but uses the index m and the new variable factor μ_i. The j index is as usual the index of the specific dimension and consequently $j = 1, 2, ..., n_{var}$, k and m are specific members of the bee population and k is not equal to i. The factor μ_i is defined by (7.8):

$$\mu_i = trial_i/(trial_i + trial_k) \tag{7.8}$$

As before trial is confined in the range $[1, n_{var}]$ and provides the number of times the source location has failed to produce an improvement in the specified food source quality and is updated accordingly. Thus the effect of the random term is modified by the factor μ_i according to information about the current quality of the sources. Li et al. point out that the value of μ_i is confined by the inequality:

$$\frac{1}{n_{var}+1} \leq \mu_i \leq \frac{n_{var}}{n_{var}+1} \tag{7.9}$$

Thus the range of values for μ_i lies within the range [0, 1], since if n_{var}, the number of variables, becomes large the end points of the range tend to 0 and 1 and if small still lies within this range. For example if $n_{var} = 2$ then $1/3 \leq \mu_i \leq 2/3$. To consider the effect of the μ_i factor we note that if $trial_i$ is small it means that the source is currently good; if $trial_i$ is large then source is not efficient. This value effects μ_i and if $trial_i$ is small so is μ_i. This means less emphasis on exploration. However, if $trial_i$ is large so is μ_i and this encourages exploration. Li et al. hope this feature provides a more nuanced balance between exploitation and exploration. Li et al. also introduce the factor to the onlooker bee stage of the algorithm by using (7.10).

$$Y_{0,j}^{*} = x_{0j} + r_u(x_{kj} - x_{0j})\mu_i \tag{7.10}$$

where r_u is a uniform random number in the range [−1 1] and is different for each j. Here the local investigation of the solution space is again modified by using the factor μ_i. We note that at any time if the value of any trial factor exceeds n_{var} then it is reset to n_{var}. Towards the end of each cycle of the algorithm the average of all the trial values associated with each bee are computed. Then this average trial value is compared with $0.9n_{var}$ if it is smaller then the employed bees are recycled using (7.5) since they are

Algorithm 7 Modified ABC by Li et al.

Set initial values for all parameters.
Generate randomly the initial population within the solution space.
set all trial values to 1.
for $gen = 1$ **to** $maxgen$ **do**
  for $i = 1$ **to** N_s **do**
    for $j = 1$ **to** n_{var} **do**
      Compute $x_{i,j}*$ from (7.7)
    end for
  end for
  if improvement then accept new point and update trial value **then**
    $trial(i) = 1$
  else
    $trial(i) = trial(i) + 1$
  end if
end for
Calculate probabilities for employed bees using (7.3) or (7.4).
set $i_b = 1$
while $i_b < N_s$ **do**
  mutate for one element using (7.10).
  accept new value if improved and update trial values.
  $i_b = i_b + 1$;
end while
for $i = 1$ **to** N_s **do**
  if $trial(i) > n_{var}$ **then**
    $trial(i) = n_{var}$
  end if
end for
if average trial $> 0.9 n_{var}$ **then**
  reinitialize 0.9 of employed bees using (7.5)
  record best solution
end if

considered inefficient. The pseudo-code for this given by the authors Li et al. takes the form of **Algorithm 7**.

Li et al. have applied there modified algorithm to a vehicle path optimizing problem and the results show significant improvement particularly for high dimensions on the original ABC algorithm. It would be interesting to see how this modification worked in relation to the standard test set of non-linear optimization problems.

The final Modification to the ABC algorithm we will consider is that proposed by Gao and Liu (2011). These authors introduce two major changes to the algorithm. Firstly instead of the usual random generation of the initial population, they propose to generate the initial population based on chaotic systems and opposition based learning. Their justification for this is that this process can provide improved initial values which should result in rapider convergence. The second modification of Gao and Liu is to introduce a search procedure based on Differential Evolution which we have discussed in Chapter 2. We provide more details for these interesting suggestions. The chaotic generation of points in the solution space is achieved using the sine function and is given in (7.11),

$$c_{k+1,j} = \mu c_{k,j}(1 - c_{k,j}), \quad \text{for } k = 0, 1, 2, ..., K \tag{7.11}$$

for all j. Initially the $c_{0,j}$ are randomly selected in [0, 1], the value K is the user set number of chaotic iterations and j denotes the index of the dimension of the problem. Having generated the value of c_{kj} we can generate values that lie within the solution space using the equation:

$$x_{ij} = r_{min,j} + c_{k,j}(r_{max,j} - r_{min,j})$$

where $i = 1, 2, ..., n_s$ is the index of the members of the population, $j = 1, .., n_{var}$ the index of the variables and $r_{min,j}$ and $r_{max,j}$ are the lower and upper bounds of the problem solution space for each variable. In the second stage of this process opposition based learning is used, this procedure is described in Rahnamayan et al. (2008). A new set of points are then generated using (7.12)

$$opx_{ij} = r_{min,j} + r_{max,j} - x_{ij} \tag{7.12}$$

Finally the best n_s individuals of the $opx_{i,j}$ and $x_{i,j}$ are selected using the objective function to constitute the new population. The next modification which seeks to improve the exploitation element of the ABC algorithm is based on differential evolution and takes the form:

$$v_{i,j} = x_{best,j} + r_u(x_{r1,j} - x_{r2,j}) \tag{7.13}$$

where r_u is a uniformly selected random number in the range $[-1, 1]$. The indices r_1 and r_2 are different values chosen randomly from the indices $1, 2, ..., n_s$. An important feature of this amendment is that $x_{best,j}$ is the current best member of the bee population. This equation drives the exploitation phase of the algorithm around the current best value. Gao and Liu have given a useful outline of their modified algorithm.

Gao and Liu have carried extensive numerical tests on the performance of their algorithm on a wide range of test problems and have reported significant improvements on the performance of their algorithm when compared with the original ABC algorithm. This completes the description of our selection of modifications of the ABC algorithm.

Table 7.1 Minimization of the function RAS6 using various numbers of iterations

Iterations	Mean	Best	Worst	St Dev
100	1.5358	0.0030	2.9024	0.8192
200	0.0174	1.4646×10^{-4}	0.1573	0.0441
500	3.5527×10^{-16}	0	7.1054×10^{-15}	$1.58884646 \times 10^{-15}$

Table 7.2 Minimization of the function RAS6 using various numbers of food sources

food Sources	Mean	Best	Worst	St Dev
10	0.3161	$6.09464646 \times 10^{-4}$	1.0924	0.4333
20	0.0322	4.4813×10^{-5}	0.1751	0.0548
30	0.0047	1.5159×10^{-5}	0.0376	0.0092

Table 7.3 Minimization of the function RAS6 using different probability formulae

Formula	Mean	Best	Worst	St Dev
Using (7.3)	0.0089	9.8661×10^{-6}	0.0780	0.0177
Using (7.4)	0.2235	6.2507×10^{-4}	1.0634	0.3544

7.4 SELECTED NUMERICAL STUDIES OF THE PERFORMANCE OF THE ABC ALGORITHM

In this section we will study the behavior of the ABC algorithm when tested on standard test problems. In each case, ten or twenty runs of the algorithm are performed for the numerical studies. However because of the random nature of the process caution should be exercised before coming to conclusions about the results. Many runs of the algorithm are used to average out some of the effects of this randomness in each of the studies. The first test we undertake is to study the effect of the number of iterations used. The exercise uses the Rastrigin function with 6 variables, RAS6, 30 nectar sources a limit value of 100 before the scout bees take part. The results are provided in Table 7.1.

As expected, Table 7.1 shows that increasing number of iterations improves the performance of the algorithm minimizing Rastrigins function with six variables.

In Table 7.2 we show the effect of changing the number of food sources available to the bees. We use 200 iterations for these experiments a limit of 100 and again Rastrigin's function with six variables.

Table 7.2 shows that the performance when there are only 10 food sources is not strong since the result is distant from the true global minimum of zero. However the result markedly improves with an increase in the number of food sources to 20 and when the number of food source is 30 the best solution is found.

In Table 7.3 we consider the effect of choosing alternative formula for the probability calculation, which have been suggested, at the onlooker bee stage these are described

Table 7.4 Minimization of Rastrigin's functions with different numbers of variables

Function	Mean	Best	Worst	St Dev
RAS6	0	0	0	0
RAS8	1.6045×10^{-8}	1.1923×10^{-11}	0.0780	5.1134×10^{-4}
RAS10	9.3491×10^{-4}	9.9657×10^{-8}	0.0018	4.0658×10^{-4}
RAS15	0.9401	0.0322	2.0076	0.5565

as form1 and form2. Where the formula form1 is given by (7.3) and form2 is given by equation (7.4) in Section 7.2 of this chapter. The experiments minimize RAS6 and are carried out using 500 iterations and the number of food sources is 30.

We see that Table 7.3 shows a significant improvement in using (7.4) rather than using (7.3). However much further work on a range of problems would be required to establish this result.

In the last table, Table 7.4 we see the effect of increasing dimensionality on the difficulty of solving the problem by studying the algorithms performance on the six variable Rastrigin function (RAS6) and the eight variable Rastrigin function (RAS8) and ten variable Rastrigin function (RAS10) and finally the 15 variable Rastrigin function (RAS15). Five hundred iterations are used with 30 food sources for each dimension.

Table 7.4 shows that the results become less accurate with the increasing dimension of the problem, in particular the 15 variable problem has a relatively poor result and this does demonstrate how the increase in the dimensions of the problem is a very influential factor on the performance of the algorithm.

We now give some graphical insight into the performance of the ABC algorithm by drawing contour graphs of the function and displaying the changing position of the bees and food sources as the algorithm progresses. Figure 7.1 shows the contour plots of the Rosebrock function with two variables.

Figure 7.1 shows contour graphs at 0, 10, 50 and 1000 iterations. The trial solutions have clustered to the same point after 1000 iterations. There is only one global optimum for this problem and it lies in a very shallow valley. The algorithm appears to locating points along a contour of the function here the objective function values will be close in value.

We now consider the two variable Styblinski-Tang function. Figure 7.2 shows the contour plots for the initial locations and at 10 iterations, 50 iterations and after 200 iterations. In this case convergence is very rapid to the solution at $(-2.9035, -2.9035)$, the contour graphs show the situation at 0, 10, 50 and 200 iterations.

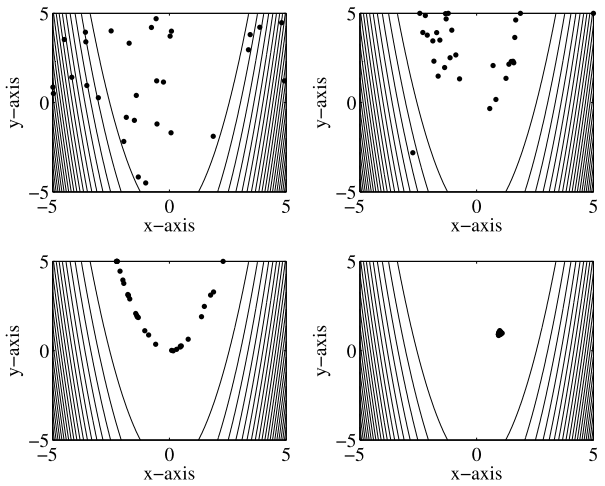

Figure 7.1 Optimization of Rosenbrock's function showing clustering along the optimum level contour and then convergence to the solution at the point [1, 1] with the function value zero. At the initial locations and after 10, 50 and 1000 iterations.

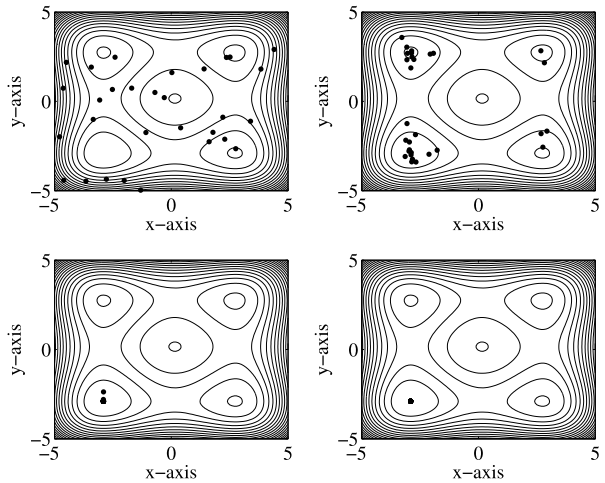

Figure 7.2 Optimizing the Styblinski-Tang showing initial nectar locations and nectar locations after 10 iterations, 50 iterations and 200 iterations.

The final example shows the contours of the egg-crate function and the state of the convergence at 0, 10, 50 and 200 iterations in Figure 7.3. This function has very many minima quite closely packed together. The swarm approaches the global minimum very rapidly and the points become practically coincident after only 200 iterations of the algorithm.

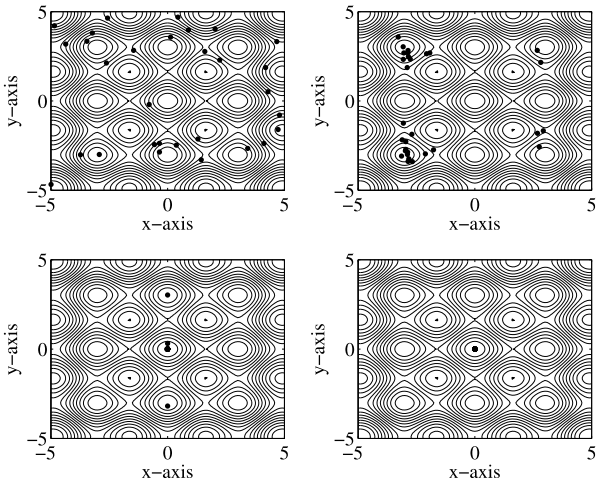

Figure 7.3 Optimization of the egg-crate function. Rapid convergence after some initial exploration of the region. Rapid convergence occurs to the true minimum at [0, 0]. The contour graphs show the initial position and after 10, 50 and 200 iterations.

7.5 SOME APPLICATIONS OF ARTIFICIAL BEE COLONY OPTIMIZATION

Hossain and El-shafie (2013) use the ABC algorithm to optimize the reservoir release per month at the Aswan high dam. The algorithm was used to solve the problem based on actual historical inflow data and it succeeded in meeting the demand over a specific period.

Bolaji et al. (2013) provides a survey of modifications to the ABC algorithm and indicate a number of applications of the algorithm. They include: stock price forecasting, artificial neural networks, image processing, electric load forecasting and flow job scheduling.

We now discuss in more detail applying the ABC algorithm specifically to the important problem of optimal financial portfolio selection. The paper of Bacanin et al. (2014) describes how the authors have applied the ABC algorithm to minimizing risk in investment portfolio selection which is major financial problem.

Many optimization problems arise in the world of finance. An important one of these is one which arises in establishing a profitable portfolio of investments. An approach to this would be to maximize the return from a selected group of investments, however Bacanin et al. (2014) discuss the alternative approach is to minimize the risk of the portfolio of investments by selecting the investments according to some criteria. This is an established problem and mathematical formulation has been provided by Markowitz (1952). Later in 1990, Markowitz, Miller and Sharpe won the Nobel prize for work in the theory of financial economics and corporate finance. This formulation of the

minimum risk approach may be expressed in the form

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j \ \text{cov}(R_i R_j) \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i R_i \geq R \\ & \sum_{i=1}^{n} w_i = 1 \\ & w_i \geq 0, \ \text{for all } i = 1, 2, ..., n \end{aligned}$$

Here w_i is the weight or proportion of the investment in the portfolio, n is the number of individual investments and R_i is the mean return on investment i. The value R represents the required minimum return for the selected portfolio. The coefficients $\text{cov}(R_i R_j)$ are the correlation coefficients between the returns i and j. In this formulation the aim is to find the values of w_i which are the proportions of the investments in the portfolio which minimize the risk or variance of the portfolio subject to constraints on the investment return requirement. The sum of the proportions w_i must be 1 and each w_i must be positive. Although the problem could be reformulated to maximize returns. Mathematically this problem is classified as a constrained quadratic programming problem.

Bacanin et al. note that the constraints on investment returns can be reflected into the objective function by introducing the parameter λ, this parameter must be positive and lies in the interval [0, 1]. The new objective function now provides a balance between risk and return and Bacanin et al. indicate that this as may be used as an alternative formulation for the original problem. The problem may be solved using a range of methods the classical approach being to use Lagrange multipliers. However the methods can be included in the objective function using penalty function methods. Note that the general solution of constrained non-linear optimization problems is discussed in Chapter 9. Bacanin et al. note that in practical applications many important additional factors can affect the formulation of this problem. For example the proportion of the investments may not be continuously variable, there may be legal restraints on the simultaneous holding of certain assets, and there may also be economic constraints on the problem. Consequently they reformulate the problem taking into account some of these issues which includes additional constraints. However Nature Inspired optimization methods could clearly be used for this type of problem and its variants and these can supply the global optimum for the problem.

Bacanin et al. then describe the ABC algorithm and apply it to historical data providing the performance of five stocks from 2007 to 2011. Bacanin et al. apply the basic model and report the results for the weights found by the ABC algorithm and state

that it has the potential to solve this type of problem. Another application similar to this one is given by Chen (2014). However this application is to uncertain portfolio selection. The returns are based on expert prediction rather than historical data. An interesting feature of this application is that the problem is formulated as a three objective non-linear programming problem subject to constraints. Where the three competing objectives are to minimize risk, to maximize return and to maximize the diversity of the portfolio. Interestingly the diversity objective uses the standard entropy function. The solution of multi-objective function problems is discussed in Chapter 9 in this text and a range of methods of solution for the problem are given. Chen applies the ABC method to specific numerical examples in portfolio selection. He indicates the results show the ABC algorithm is effective for solving the portfolio selection problem.

7.6 DESCRIPTION OF THE ANT COLONY OPTIMIZATION ALGORITHMS (ACO)

The term Ant Colony Optimization (ACO) is a collective term which is used to describe a group of algorithms which use a cooperative swarming method based on modeling the behavior of ant colonies when foraging for food. This group of algorithms was introduced by Dorigo et al. (1996) and, Dorigo and Gambardella (1997) when they used the term Ant System (AS) to describe one of the first versions of the ACO methods and this is the one we shall describe in detail. Like the ABC algorithm information is passed from some members of the colony to others. In the ACO group of algorithms this is achieved by placing pheromones rather than a waggle dance as in the ABC algorithm.

Whilst individual ants can only do very simple tasks, a colony working together can exhibit intelligent behavior. As an ant walks it deposits a chemical called a pheromone that encourages other ants to follow it. The pheromone evaporates over time and this encourages the ants to explore other routes in their search for food. The greater the number of ants that follow a particular path, the stronger the trail of pheromone. The density of pheromone deposited also increases more rapidly on shorter paths and this encourages more ants to take these shorter paths.

From the natural observed behavior of a colony of ants, Goss et al. (1989) developed a formula which gave the probability that an ant at a junction of two branches, would select a particular branch of the two. The formula takes the form:

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h}$$

Here p_1 is the probability of taking the first branch and k and h are constants fitted to the recorded behavior of the ants, Goss et al. found $k = 20$ and $h = 2$ provided good results for the set of observations they made. The values m_1 and m_2 provide the number

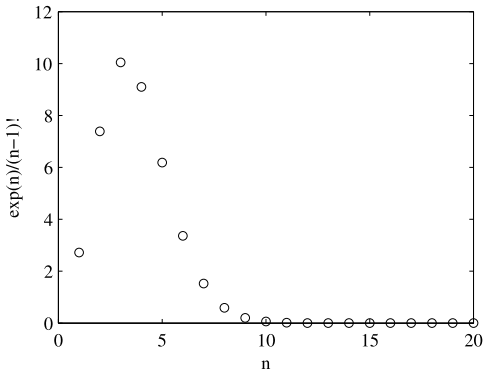

Figure 7.4 Illustrates the comparison between $(n-1)!$ and e^n.

of ants in the colony who have so far chosen the first and second branch respectively. Clearly the probability of taking the second branch p_2 is $1-p_1$ when there are only two such options. Dorigo et al. utilized and extended this concept in generating their ant colony algorithm.

These principles may be usefully applied to optimization problems in particular they can be easily modified to find the solution of particularly difficult problem known as the symmetrical traveling salesman problem or TSP which seeks to find the optimum route for a salesman who must visit a number of cities and return to the starting point of his journey by a route which will minimize the distance the salesman travels in visiting all the cities once. It is one of a number of difficult combinatorial problems which present major problems for the effective solution, e.g. the Knapsack problem and the job assignment problem.

The TSP is simple to describe but it is very difficult to solve. To solve this problem by enumeration of alternative routes for n cities involves the evaluation of $(n-1)!/2$ distinct tours (division by two since back and forth along the same route does not have a different cost). The value of $(n-1)!/2$ increases rapidly with n, hence the difficulty of the problem. For example for a five city example the method of enumeration would involve 4! or 24 possible tours but 20 cities would involve $19! = 1.2164510004 \times 10^{17}$ possible tours. It is estimated that using the direct enumeration method for 20 cities could take over 190,000 years. Despite this challenge the size of problem that can be solved is increasing rapidly for example the 1980s saw the solution of problems with 318 cities while in 2006 a TSP problem with 85900 cities was solved. This is due to the steady development of better and better algorithms. Figure 7.4 illustrates how the factorial function can grow much faster than the exponential function thus the ratio $\exp(n)/(n-1)!$ rapidly tends to zero with increasing n due to the factorial function value dominating the exponential function value.

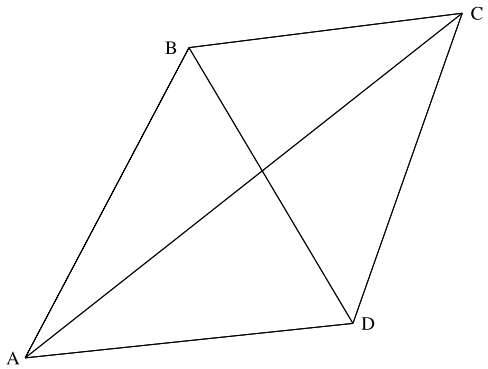

Figure 7.5 TSP problem with four towns.

The TSP problem is formally classified as NP hard which highlights its exceptional difficulty. The term NP designates the term Non-deterministic Polynomial and refers to the time taken to solve the problem. It is expected, but not proved, that no polynomial time algorithms exist to solve this type of problem. Consequently there is an ongoing search for improved algorithms that are far more sophisticated than the direct enumeration of the possible solutions which as we have seen is not a practical method for even relatively small problems. The problem may be written as a constrained minimization problem in variables having the values zero or one, for large problems this is not a practical formulation. To adapt the Ant System approach to the TSP problem we must impose specific requirements on the individual ant consistent with the behavior of a colony of ants.

1. It must visit each city exactly once;
2. A distant city has less chance of being chosen (because the more distant the city, the less the visibility);
3. The more intense the pheromone trail laid out on an edge between two cities, the greater the probability that edge will be chosen;
4. Having completed its journey, the ant has deposited more pheromones on all the edges it traversed
5. After each iteration, trails of pheromones evaporate at a specific rate.

Each TSP problem may be represented by a connected graph and an example is shown in Figure 7.5. A, B, C and D are the locations of four towns. The distances between the various points on the graph can be represented by $d_{i,j}$ where i and j are the labels of the end points on the graph. This represents the distance between the two points i and j. The concept can be related to Cartesian coordinates of the points on the graph by being defined as the Euclidean distance between them. The line connecting

i and j is called an edge. Thus members of the ant colony will traverse the edges of the graph laying pheromones as they pass. The ants will tend to choose the shortest paths and these will be enhanced by having additional pheromones deposited and thus attracting other ants. To develop an algorithm these features must be implemented in a formal way. If we define the key features of the problem as follows: the number of towns is n_t, the total ant population as m and the number of ants in each town as n_i and $\tau_{i,j}$ as the ant pheromone intensity on each trail joining the points i and j.

Dorigo et al. used (7.14) to model the manner in which ants deposited their pheromones on each trail. Thus if $\tau_{i,j}$ is the intensity of pheromone on the edge i, j this takes the form:

$$\tau_{i,j}^{(t+n_t)} = \rho\tau_{i,j}^{(t)} + \Delta\tau_{i,j} \tag{7.14}$$

where $1-\rho$ represents the amount of evaporation of the pheromone on the trail, the quantity $\Delta\tau_{i,j}$ represents the amount of pheromone added. So (7.14) represents both how the pheromone increases and how it decreases through natural evaporation. Note that this equation represents a complete cycle of the process in that all ants have completed their tours. Thus the use of the superscript $t+n_t$ in (7.14). Hence if $\Delta\tau_{i,j}^k$ represents the amount of pheromone deposited per length of trail on edge joining i and j by the kth ant then the amount deposited is the sum of all these deposits for each ant thus:

$$\Delta\tau_{i,j} = \sum_{k=1}^{m} \Delta\tau_{i,j}^k \tag{7.15}$$

where $\Delta\tau_{i,j}^k$ is defined by (7.16)

$$\Delta\tau_{i,j}^k = \begin{cases} Q/L_k & \text{if } k\text{th ant uses edge } i,j \text{ in its tour} \\ 0 & \text{otherwise} \end{cases} \tag{7.16}$$

where Q is a constant and L_k is the tour length of the kth ant. Dorigo et al. introduce further features to their algorithm by noting that since each town must be visited only once, a list must be kept of towns visited. This they call the tabu list since if a town has been visited it must not be visited again. One of these lists is associated with each ant, consequently for the kth ant there is an associated tabu list or vector, $\mathbf{tabu}_k$.

Since an individual ant makes specific choices when presented with alternative roots there must be some way of modeling this decision process. Dorigo et al. formulate this by taking into account the visibility of the next locations v_{ij} and the intensity of the pheromone trail, τ_{ij}. They propose that the specific ant k will make a choice with

probability calculated from (7.17).

$$p_{i,j}^{k} = \frac{\tau_{i,j}^{\alpha} \nu_{i,j}^{\beta}}{\sum_{k \in S_k} \tau_{i,k}^{\alpha} \nu_{i,k}^{\beta}} \tag{7.17}$$

Here S_k are the set of allowed indices for the kth ant i.e. the towns not yet visited, and $\nu_{i,j} = 1/d_{i,j}$. Thus, the visibility is reciprocal of the distance between the towns i and j thus if the distance is great the visibility tends to zero. The allowed towns are the set of towns not in the particular tabu list. The powers α and β can be adjusted to alter the effects of the visibility and pheromone trail intensity. For example, if $\beta = 0$ then the visibility has no effect. If $\alpha = 0$ then the pheromone intensity will have no effect. The larger α and β the more amplified the two characteristics become. Clearly if the town is highly visible and the intensity of pheromone on the trail is high then the probability of selecting this part of the route will be high. We note also that the probability will be between 0 and 1, since $\nu_{i,j}$ and $\tau_{i,j}$ are positive and sum of all the allowed combinations will always be greater than or equal to one of them.

A broad description of the ant system algorithm is given by Dorigo et al. They state that at time zero the ants are positioned in the towns or vertices of the network that defines the problem and trail intensities are associated with each edge. All the tabu lists for each ant have their first position set to the starting town of each ant. The ants are then ready to begin their journey and do so by selecting a route to take amongst the alternatives using the probability defined by (7.17). After n_t iterations, n_t is the number of towns, when all ants will have completed a tour and their tabu lists have been filled thus for each ant the length of their journey can be computed and values of pheromone can be updated using (7.16). The best route for all the ants is saved. Based on this descriptions Dorigo et al. provide a pseudo-code for the implementation of the ant algorithm. This takes the form given in the pseudo-code **Algorithm 8**.

This completes the description of the ant colony algorithm. Dorigo et al. make the important observation that since they have experimentally found that there is a linear relation between the number of towns and the best number of ants and in fact that the number of ants is close to the number of the cities, consequently the complexity of their algorithm is, they state, of order of the number of cycles times the cube of the number of towns.

Dorigo et al. carried out a range of numerical tests on standard TSP problems and found their algorithm produced successful results. Since its introduction in the 1990's the algorithm has become established as an important development in the field. The key parameters of the algorithm are α, β, ρ and Q. Dorigo et al. indicate that $\alpha = 1$, $\beta = 0.5$, $\rho = 0.5$ and $Q = 100$ is a good parameter set.

Algorithm 8 ANTS.

Initialize all the values and time and cycle counters. Set initial values for the pheromones at some small selected positive constant c. Set additive amounts of pheromone at zero. Place the m ants selected on the network nodes. Set number cycles, $n_c = 0$. Set number of towns, n_t
Begin main cycle
for $k = 1$ **to** m **do**
  Place the initial location of each ant in their tabu list
end for
Move all the ants on to the next town. Repeat $n_t - 1$ times.
for $k = 1$ **to** m **do**
  Use (7.17) to calculate the probability that the ant k moves to town j from its current position in town i.
  As the ant is now in town j this is placed in the $\mathbf{tabu}_k$
end for
for $k = 1$ **to** m **do**
  move the kth ant from its tabu list position n to 1.
  Compute the length of the tour of ant k as L_k.
  Find the smallest route i.e. $\min(L_k)$ for all $k = 1, 2, ..., m$.
  if better than the current minimum length route **then**
    this is updated to the new value.
  end if
  Now calculate the new pheromone values laid for each edge and for each ant.
  for $k = 1$ **to** m **do**
    Use (7.15) to calculate values of $\Delta\tau_{i,j}^k$ then

$$\Delta\tau_{i,j} = \Delta\tau_{i,j} + \Delta\tau_{i,j}^k$$

  end for
end for
For every edge i, j compute $\tau_{i,j}$ using (7.14) to update pheromone level.
Set $t = t + n_t$, $n_c = n_c + 1$
for every edge i, j set $\Delta\tau_{i,j} = 0$
if the maximum cycles have not be reached **then**
  clear all tabu lists and repeat the main cycle.
else
  output shortest route
end if

7.7 MODIFICATIONS OF THE ANT COLONY OPTIMIZATION (ACO) ALGORITHM

Dorigo et al. briefly describe alternatives to the original algorithm which they call the ant density and ant quantity algorithms. The only difference between these versions of the algorithm is in the way in which the pheromone on the trails is updated. For the ant quantity model (7.16) is replaced by

$$\Delta_{i,j}^{k} = \begin{cases} Q & \text{if } k\text{th ant uses edge } i, j \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

For the ant density model and

$$\Delta_{i,j}^{k} = \begin{cases} Q/d_{i,j} & \text{if } k\text{th ant uses edge } i, j \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

where $d_{i,j}$ is the distance between town i and town j.

Dorigo et al. carried out studies on the relative performance of these alternative updating methods and concluded that the original updating method given by (7.16) produced the best results. Dorigo (2007), in Scholarpedia, provides a useful outline of the relationship between the main ACO algorithms.

1. The first ACO algorithm called Ant System (AS) is the algorithm which we have described in Section 7.6;
2. Ant Colony System (ACS)
3. The MAX-MIN system

We shall now describe briefly the ACS algorithm and the MAX-MIN algorithm. The ACS algorithm proposed by Dorigo and Gambardella (1997) is the first major improvement on the original algorithm and is similar to the original algorithm. However one significant difference is that it uses a different rule in deciding whether to move from location i to location j. This depends on the selection of a uniform random variable U in the range [0, 1] and a parameter U_0 which is selected by the user. Then the route is selected using (7.18).

$$\text{if} \quad U \le U_0 \text{ then } j = \underset{j \in j_{allowed}}{\operatorname{argmax}}(\tau_{ij}\nu_{ij}^{\beta}) \tag{7.18}$$

else use rule (7.17) as in the original AS algorithm.

This is strongly directed by the pheromone element but is counteracted by increasing diversity by a local pheromone updating process performed by all ants but only to the edge most recently traversed. This is achieved by (7.19).

$$\tau_{ij} = (1 - U)\tau_{ij} + U\tau_0 \tag{7.19}$$

where U is a uniform randomly selected number in the range [0, 1] and τ_0 the initial value of the pheromone.

This process will strongly increase the diversity of the method. The final modification is to perform the pheromone update but only along the edges of the current best ant route hence:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho(\tau_{ij})_{best}$$

where $(\tau_{ij})_{best} = 1/L_{best}$ and L_{best} is the length of the best route.

Another important development was the introduction of the MAX-MIN Ant System (MMAS); this improvement was introduced by Stuzle and Hoos (1997). This algorithm differs in two ways from previous algorithms in that only the best ant is permitted to add pheromones and the amount of pheromone that can be added is constrained between an upper and lower bound. Specifically this means that

1. The pheromone update takes the form:
   $\tau_{i,j} = (1-\rho)\tau_{i,j} + \Delta\tau_{i,j}^{best}$ where $\Delta\tau_{i,j}^{best} = 1/L_{best}$.
   If the best ant used edge (i, j) in its tour otherwise it is taken as zero.
2. The pheromone amount $\tau_{i,j}$ is constrained by inequality (7.20).

$$\tau_{min} \leq \tau_{i,j} \leq \tau_{max} \tag{7.20}$$

The variable $\tau_{i,j}$ is added in the same manner as described in the AS algorithm. Here τ_{min} and τ_{max} are the minimum and maximum allowed for the pheromone values respectively, these values are selected by the user but τ_{max} may be set at $1/\rho L^*$ where L^* is the minimum tour length. Clearly this is only available if the optimum tour length, or a good approximation to it, is known.

Another more recent modification of the ACO algorithm was suggested by Yu et al. (2014). They introduce chaotic behavior using the logistic map to introduce a random element in the manner in which the pheromone updates are performed. Yu et al. (2014) test their algorithm on a selection of standard test problems and conclude that their algorithm provides significant improvement on other ACS algorithms.

The ACS algorithm is naturally applied to TSP problems and a wide range of combinatorial problems such as job scheduling, knapsack problems, vehicle scheduling and many others and solutions often involve integer values. Although these problems are similar to nonlinear global optimization problems in their difficult nature, the problems tackled are different in structure. Consequently we do not provide numerical studies for the algorithm, but leave it to the reader to decide if they wish to study the algorithm in more detail using, for example, algorithms implemented by Mathworks.

7.8 SOME APPLICATIONS OF ANT COLONY OPTIMIZATION

The major application of ACO is to the TSP. It is reported that the algorithm has allowed the solution of much larger TSP problems than before its introduction. However the method has been applied to standard non-linear optimization problems as well and this approach has been described in Jalali et al. (2005). They show how the ACO algorithm, normally associated with the TSP, may be applied to the optimization of standard non-linear both unconstrained and constrained test functions. They also considered the practical problem of optimizing reservoir operations to find an optimum pattern of water releases.

Automatic programming is the use of search techniques to find programs that solve a problem. The most commonly used technique is genetic programming, which uses genetic algorithms to carry out the search. Green et al. (2004) introduce Ant Colony Programming (ACP) which uses an ant colony based search in place of genetic algorithm to solve this problem. They tested and compared it with other approaches in the literature.

Abraham et al. (2013) use ACO to attempt to find numerical solutions of Diophantine equations, an established and difficult problem in pure mathematics since there are no general methods to find solutions of such equations. Their experimental results compare well with those of other machine intelligence techniques, thereby validating the effectiveness of their proposed method.

7.9 SUMMARY

We have considered two algorithms in this chapter; the Artificial Bee Colony (ABC) algorithm and Ant colony Optimization (ACO). These algorithms share a common concept of swarm intelligence, that is by allowing the transfer and use of information between all members of the colony. In the ACO case this is achieved by using pheromones and in the ABC algorithm by using onlooker bees. The Ant Colony optimization algorithm has a long established role amongst nature inspired optimization methods having been introduced in 1991 and subsequently used to successfully solved the TSP problem for large number of cities and over the years has been applied to many demanding and practical combinatorial problems with many reported successes. The algorithm success has provided a spur to further research in the area. We have described the algorithm and provided a description of some of the developments in the field.

The ABC algorithm is a relatively recent entry to the field of nature inspired optimization the algorithms having been introduced in 2005. Specifically designed for the global optimization of non-linear optimization problems, it has been extensively studied and tested on a wind range standard test problems and practical applications. It has been reported that its performance is strong in relation to many algorithms in the field. We have described the basic algorithm and some of the suggested improvements to the basic algorithm.

7.10 PROBLEMS

7.1 For Table 7.4 in this chapter, which provides the results for Rastrigins function for various dimensions using the ABC algorithm of that problem, draw a graph of the accuracy of each average result against the dimension of the problem. Compute the error of the solution of the problem by noting the optimum solution of the problem is zero.

7.2 Use (7.11) given by Gao and Liu to generate chaotically an initial set of values for Nectar/Food locations for the ABC algorithm. In (7.11) take $K = 4$ and plot each generation of points for $k = 1,\ 2,\ 3$. Take only two points so $j = 1,\ 2$ and generate initial values for each of these points. Take $\mu = 0.5$, $c_{0,1} = 0.10$, $c_{0,2} = 0.25$ and $c_{0,3} = 0.05$.

7.3 Using (7.2) and (7.3) compute the onlooker bee selection probabilities, for four bees (numbered 1, 2, ... 4) located at [2, 5], [3, 2], [1, 1], [6, 2] and assume an objective function $f(x) = x_1^2 + \cos(x_2)$. Find the sum of the probabilities.

7.4 By defining $x_{ij} = 1$ as the decision if an edge i, j is traversed or $x_{ij} = 0$ if it is not traversed and L_{ij} as the length of the edge joining the towns i and j find an expression for the minimization function. Formulate the TSP as a constrained minimization problem. You should model the constraints as imposing the requirements that each city is visited only once by the salesman and each city must be visited.

7.5 This problem uses the formula developed by Goss et al. from a study of ant colony behavior for the probability p_1 of selecting a particular branch from a choice of two:

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h}$$

Take $k = 20$ and $h = 2$ as suggested by Goss et al. For the purposes of this study assume that 2 ants have already selected the first branch and 4 ants have selected the second branch. Use this formula 4 times for 4 different ants to calculate the probability that an ant will take a particular branch. In each case if it is greater than 0.5 add 1 to the total m_1 of ants taking the first branch else add 1 to the m_2 total.

7.6 Given an ant $k = 2$, in city $i = 3$, use (7.17) to calculate the probability that this ant will select the route from city $i = 3$ to city $j = 5$, where the set of allowed cities for this ant is [2, 4, 5]. You may assume $\alpha = 1$ and $\beta = 2$ and that the required pheromone levels are given by [0.5, 1.5, 2] and the visibility values [0.3, 0.5, 0.05].