2

Artificial Bee Colony – Modifications and An Application to Software Requirements Selection

Bahriye Akay

Department of Computer Engineering
Erciyes University, Melikgazi, Kayseri, Turkey

CONTENTS

2.1 Introduction

A honey bee colony exhibits collective intelligence in some of their daily activities. One of them is the foraging task which is crucial for survival of the colony. In the foraging task, bees are distributed into three roles: employed foragers,

onlooker bees and scouts. The employed foragers are responsible for exploiting the nectar of discovered food sources. When she arrives at the hive, after unloading the nectar, she gives information about her food source by dancing. The onlooker bees watch the dances of some bees in the dance area and select food source toward which to fly. After having been exploited by the bees, an exhausted food source is abandoned and its bee starts to search for a new source as a scout bee. In 2005, Karaboga was inspired by honey bee foraging and proposed the Artificial Bee Colony (ABC) algorithm [1]. In ABC, a food source location corresponds to a solution which is a vector of design parameters to be optimized and its nectar corresponds to the solution's fitness to be maximized. The ABC algorithm has three phases: the employed bee phase which exploits the information of discovered solutions, the onlooker bee phase which has a stochastic selection scheme to select high quality solutions and performs local search around them and the scout bee phase to detect exhausted solutions. The ABC algorithm combines division of labour, positive and negative feedback, multiple interaction and fluctuation properties leading swarm intelligence.

The original ABC algorithm has been proposed for solving continuous and unconstrained optimization problems. Each design parameter, $x_i \in \mathbb{R}$ and for a n-dimensional problem, the search space, $\mathbb{S}$, is defined as a n-dimensional rectangle in $\mathbb{R}^n$ ($\mathbb{S} \subseteq \mathbb{R}^n$). Therefore, a real-valued representation is adopted and local search operators are able to produce solutions in $\mathbb{R}^n$. Since there is no additional constraint and a single objective function is considered in the problem, a greedy selection is employed to favour the solutions with less objective value. The basic ABC algorithm gained success on especially high dimensional and multi-modal unconstrained problems [2].

When the algorithms proposed for unconstrained optimization are intended to solve combinatorial, constrained or multi-objective problems, they are required to be modified to adopt the search space discretion and/or to handle with feasible solutions and multiple objectives. This chapter aims to give information about problem-related modifications in the ABC algorithm related to representation, local search strategy and selection strategy to solve different kinds of problems.

The rest of the chapter is organized as follows: in Section 2.2 a brief description of the ABC algorithm is presented. In Section 2.3, the modified versions of the ABC algorithm are provided and explained in detail. In Section 2.4, an application of the ABC algorithm to a software requirements selection problem is demonstrated. Finally, Section 2.5 is devoted to conclusions of the chapter.

2.2 The Original ABC algorithm in brief

The ABC algorithm has three phases: employed bees, onlooker bees and scout bees as in the foraging task performed by a real honey bee colony. Operations performed in each phase are presented as pseudo-code in Alg. 2.

Algorithm 2 Main steps of the ABC algorithm.

Set values for the control parameters:
SN: The number of food sources,
MCN: The maximum number of cycles,
$limit$: The maximum number of exploitations for a solution, ▷ Set problem-specific parameters:
f: Objective function to be minimized,
D: Dimension of the problem,
x_j^{lb} : Lower bound of jth parameter,
x_j^{ub}: Upper bound of $j$the parameter
for $i = 1$ *to* SN **do** ▷ //Initialization
    $\vec{x}_i = x_j^{lb} + rand(0,1)(x_j^{ub} - x_j^{lb})$, $j = 1 \ldots D$
    $f_i = f(\vec{x}_i)$
    $trial_i = 0$
end for
$cyc = 1$
while $cyc < MCN$ **do** ▷ //Employed Bees' Phase
    for $i = 1$ *to* SN **do**
        $\vec{\hat{x}} = \vec{x}_i$
        $k \longleftarrow randint[1, CS], k \neq i$
        $j \longleftarrow randint[1, D]$
        $\phi_{ij} \longleftarrow rand[-1, 1]$
        $\hat{x}_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$
        if $f(\vec{\hat{x}}) < f_i$ **then**
            $\vec{x}_i = \vec{\hat{x}}$
            $f_i = f(\vec{\hat{x}})$
            $trial_i = 0$
        else
            $trial_i = trial_i + 1$
        end if
    end for
    for $i = 1$ *to* SN **do** ▷ //Calculate probabilities
        $p_i = 0.1 + 0.9 * \frac{fitness_i}{max(fitness)}$
    end for
    $i = 0$ ▷ //Onlooker Bees' Phase
    $t = 0$
    while $t < SN$ **do**
        if $rand(0,1) < p(i)$ **then**
            $t = t + 1$
            $\vec{\hat{x}} = \vec{x}_i$
            $k \longleftarrow randint[1, CS], k \neq i$
            $j \longleftarrow randint[1, D]$
            $\phi_{ij} \longleftarrow rand[-1, 1]$
            $\hat{x}_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$
            if $f(\vec{\hat{x}}) < f_i$ **then**
                $\vec{x}_i = \vec{\hat{x}}$
                $f_i = f(\vec{\hat{x}})$
                $trial_i = 0$
            else
                $trial_i = trial_i + 1$
            end if
        end if
        $i = (i + 1) \mod (SN - 1)$
    end while
    Memorize the best solution
    $si = \{i : trial_i = max(\vec{trial})\}$ ▷ //Scout bee phase
    if $trial_{si} > limit$ **then**
        $\vec{x}_{si} = x_j^{lb} + rand(0,1)(x_j^{ub} - x_j^{lb})$, $j = 1 \ldots D$
        $f_{si} = f(\vec{x}(si))$
        $trial_{si} = 0$
    end if
    $cyc++$
end while

Before starting the algorithm, values are set to the control parameters of ABC and problem-specific parameters. The control parameters of ABC are the number of food sources (SN), the maximum number of cycles (MCN) and the parameter *limit* to determine the exhausted sources. In steps 9-13, each solution vector (food source location, $\vec{x}_i$) is initialized within the range $[x_j^{lb},x_j^{ub}]$ and evaluated in the cost function (f). $\vec{trial}$ counters holding the number of exploitations are set to 0. Steps 16-29 correspond to the employed bee phase. A local search is carried out in the vicinity of each solution. In step 21, a new solution ($\vec{\hat{x}}$) is produced by using the information of $\vec{x}_i$ and $\vec{x}_k$. This local search is an analogy of food source exploitation. Steps 22-28 perform greedy selection. If the new solution, $\vec{\hat{x}}$ is better than the current solution, $\vec{x}$, the new solution is kept in the population. Otherwise, the current solution is retained and the counter ($trial$) holding the number of exploitations is incremented by 1 (Step 27). Once the employed bee phase is completed, each solution is assigned a probability value (Steps 30-32) proportional to its fitness which can be calculated by $1/f_i$. Lines 33-52 are the steps of the onlooker bee phase in which the solutions to be exploited are selected based on a stochastic scheme. If the solution's probability is higher than a random number drawn within range [0,1] (Step 36), the solution is exploited which means a local search is conducted around the selected solution (Step 42) and a greedy selection is applied between the new solution and the selected one. In the employed bee phase, each solution is involved in the local search by turn while in the onlooker bee phase, only selected solutions are involved. High quality solutions are likely to be selected (positive feedback) but low quality solutions also have a chance proportional to their fitness. Before the scout bee phase, the best solution is memorized. In the scout bee phase, the counters, $\vec{trial}$, are checked to determine whether an exhausted source exits (Lines 54-59). In each cycle of the basic algorithm, only one scout bee is allowed to occur to avoid losing the population's previous knowledge. If there is an exhausted source, this source is abandoned and a new random source is produced instead of it (Line 56).

2.3 Modifications of the ABC algorithm

In this section, some different versions of ABC are presented to give insight into how an algorithm can be modified to handle different problem characteristics. The combinatorial, constrained and multi-objective versions of the ABC algorithm are explained in the following subsections.

2.3.1 ABC with modified local search

In the local search operator of the basic ABC algorithm, information only in one dimension is perturbed to form a new solution vector and the rest

are copied from the current solution. Although this operator is good at fine tuning, it may slow down the convergence on some problems. Therefore, some new local operators are integrated to the algorithm to gain a speed up in the convergence.

One modification in the local search operator is increasing the number of dimensions to be changed by Eq. 2.1 [3].

$$v_{ij} = \begin{cases} x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) & , \quad if \quad R_{ij} < MR \\ x_{ij} & , \quad \text{otherwise} \end{cases} \tag{2.1}$$

where R_{ij} is a random number drawn from uniform distribution within the range (0,1) and MR is the modification rate controls the number of dimensions to be perturbed. For each dimension j, if $R_{ij} < MR$, then the information in the jth dimension is changed, otherwise the previous information is kept. In [3], additionally, ϕ_{ij} is drawn at random within the range $[-SF, SF]$ where SF is a variable scaling factor based on Rechenberg's 1/5 rule.

Another modification in the ABC algorithm is superimposing the best solution in a neighbourhood topology defined in a certain radius (Eq. 2.2). This operation is proposed to be used in the onlooker bee phase in [4].

$$v_{ij} = x_{bestN,j} + \phi_{ij}(x_{bestN,j} - x_{kj}) \tag{2.2}$$

where $x_{bestN,j}$ is the best solution in a neighbourhood. In [4], for each solution, mean Euclidian distance md_i is calculated and the solutions x_j are assumed to be neighbours if $d_{ij} \leq r \times md_i$. It means that two solutions are neighbours if the distance between them is less then the mean distance weighted by r which is a predefined control parameter.

2.3.2 Combinatorial version of ABC

Combinatorial optimization finds the minimum cost set, I, among all subsets of the power set of a finite set E. Hence, the algorithms should be able to search finite set E space and some efficient operators are needed to find a good approximate solution in polynomial time. When a combinatorial problem is intended to be solved by the ABC algorithm, first, a discrete representation should be adopted. In the initialization phase, random orders of the nodes are generated unlike basic ABC generating random points in continuous space. In the employed bee and onlooker bee phases, a local search operator suitable for the problem such as 2-opt, greedy sub tour mutation (GSTM), etc., can be employed to produce mutant solutions based on the order representation [5-6]. In [5], ABC is integrated with 2-opt which aims to find a different edge by re-linking the ith and jth nodes of the current solution and a randomly chosen neighbour. In [6], a GSTM operator is used to generate new solutions. The main steps of GSTM operator used in [6] are given below.

1. Select solution x_k randomly in the population, $k \neq i$.
2. Select a city x_{ij} randomly.
3. Select the value of searching way parameter $\phi \in \{-1, 1\}$ randomly.
4. if (ϕ=1) then
   - The city visited before x_{kj} is set as the previous city of x_{ij}.
5. else
   - The city visited after x_{kj} is set as the next city of x_{ij}.
6. endif
7. A new closed tour $\hat{T}$ is generated.
8. By this new connection, an open sub tour T^* is generated. R_1: first city of T^* and R_2: last city of T^*
9. if (random $< P_{RC}$) then
   - Subtract [R_1-R_2] tour from tour T ($T^{\#} \leftarrow T - [R_1 - R_2]; T^* \leftarrow [R_1 - R_2]$)
   - Hold on T^* sub-tour $T^{\#}$ so that there will be minimum extension.
10. else
    - if (random $< P_{CP}$) then
      - Copy $[R_1 - R_2]$ sub-tour in the T tour, $T^* \leftarrow [R_1 - R_2]$
      - Add each city of T^* to the T starting from the position R_1 by rolling or mixing with probability P_L.
    - else
      - Select randomly one neighbor from neighbor lists for the points R_1 and R_2 (NL_{R_1} and NL_{R_2}).
      - Invert the points NL_{R_1} or NL_{R_2} that provide maximum gain in such a way that these points will be neighbors to the points R_1 or R_2.
      - Repeat if the inversion does not take place.
    - endif
11. endif

Here, P_{CP} is correction and perturbation probability, P_{RC} is reconnection probability and P_L is linearity probability.

2.3.3 Constraint handling ABC

Constrained optimization aims to find a parameter vector $\vec{x}$ that minimizes an objective function $f(\vec{x})$ subject to inequality and/or equality constraints (2.3):

$$\begin{array}{lll} \text{minimize } f(\vec{x}), & \vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n & \\ & l_i \leq x_i \leq u_i, & i = 1, \ldots, n \\ \text{subject to :} & g_j(\vec{x}) \leq 0, & \text{for } j = 1, \ldots, q \\ & h_j(\vec{x}) = 0, & \text{for } j = q+1, \ldots, m \end{array} \tag{2.3}$$

The objective function f is defined on a search space, $\mathbb{S}$, which is defined as a n-dimensional rectangle in $\mathbb{R}^n$ ($\mathbb{S} \subseteq \mathbb{R}^n$). All variables are bounded by their lower and upper values. Feasible region $\mathbb{F} \subseteq \mathbb{S}$ is constructed by a set of m additional constraints ($m > 0$) and the global optimum is located in feasible space ($\vec{x} \in \mathbb{F} \in \mathbb{S}$). Because infeasible solutions violating the constraints may have lower cost values, the local search and/or selection operators are modified to favour the search through the feasible space. The local search operator can pass only feasible solutions into the population or infeasible solutions can be repaired to form feasible solutions. In this case, the population includes only feasible solutions. Repairing solutions or trying to generate only feasible solutions can be a time consuming task. Alternatively, the population can hold both feasible and infeasible solutions. However, when a selection is performed, a feasible solution is preferred against an infeasible one. Deb defined the precedency rules to be used in selection phase [7] and in the constraint handling ABC [8], the greedy selection mechanism is replaced with Deb's rules to make a distinction between feasible and infeasible solutions. Deb's rules are given below:

- Any feasible solution is preferred to any infeasible solution ($violation_j > 0$) (solution i is dominant),
- Among two feasible solutions, the one having better objective function value is preferred ($f_i < f_j$, solution i is dominant),
- Among two infeasible solutions ($violation_i > 0$, $violation_j > 0$), the one having smaller constraint violation is preferred ($violation_i < violation_j$, solution i is dominant).

In the constraint handling ABC, the algorithm does not need the initial population holding only feasible solutions and the population can include both feasible and infeasible solutions. Since infeasible solutions are accepted to provide diversity, ABC does not employ a reconstruction mechanism. In the onlooker bee phase, the probability assignment scheme based on only cost function should also be replaced to assign probability to infeasible solutions based on their violation values (Eq. 2.4). In this scheme, the probability values of feasible solutions are higher than those of infeasible solutions and proportional to the cost values while the probability values of infeasible solutions are

proportional to their violation values.

$$p_i = \begin{cases} 0.5 + \left(\frac{\text{fitness}_i}{\sum_{j=1}^{sn} \text{fitness}_j} \right) * 0.5 & if\ solution\ is\ feasible \\ \left(1 - \frac{violation_i}{\sum_{j=1}^{sn} \text{violation}_j} \right) * 0.5 & if\ solution\ is\ infeasible \end{cases} \tag{2.4}$$

where $violation_i$ is the summation of constraint values going beyond the feasible region. A feasible solution's probability is within the range [0.5,1] while it is in the range [0, 0.5] for an infeasible solution.

2.3.4 Multi-objective ABC

An optimization problem with more than one criterion to be optimized simultaneously is called a multi-objective optimization problem (MOP) and the solutions that do not have dominance on all objectives (especially on conflicting objectives) are called Pareto-optimal solutions (PS) or non-dominated solutions. A MOP can be defined by Eq. 2.5:

$$\begin{gathered} \min/\max\ F(x) = (f_1(x), \ldots, f_m(x))^T \\ \text{subject to } x \in \Omega \end{gathered} \tag{2.5}$$

where Ω is decision variable space, $F : \Omega \to R^m$ is the objective vector and R^m is the objective space.

In single optimization, the ABC algorithm employs a greedy selection mechanism which compares the values of two solutions evaluated in the single objective function. A higher fitness value is assigned to a solution with better objective function. A multi-objective ABC algorithm should be able to make a comparison between solutions evaluated in multi-objective cost functions and should rank the solutions by assigning a fitness value considering all objective values.

One of the multi-objective ABC algorithms is asynchronous MOABC which replaces greedy selection with a Pareto-dominance-based selection scheme (A-MOABC/PD) [9]. In Pareto-dominance, a solution vector x is partially less than another solution y ($x \prec y$), when none of the objectives of y is less than those of x, and at least one objective of y is strictly greater than that of x. If x is partially less than y, the solution x dominates y or y is inferior to x. Any solution which is not dominated by an other solution is said to be non-dominated or non-inferior [10]. A-MOABC/PD assigns cost values by Eq. 2.6 based on Pareto rank, distance and Gibbs distribution probability of a solution.

$$f_i = R(i) - TS(i) - d(i) \tag{2.6}$$

where $R(i)$ is the Pareto rank value of the individual i, $S(i) = -p_T(i)logp_T(i)$, where $T > 0$ is temperature. $p_T(i) = (1/Z)exp(-R(i)/T)$ is the Gibbs distribution, $Z = \sum_{i=1}^{N} exp(-R(i)/T)$ is called the partition function, and N is the population size [11]. $d(i)$ is the crowding distance calculated by a density estimation technique [12].

Algorithm 3 S-MOABC/NS Algorithm.

Set the control parameters of the ABC algorithm.
CS: The Number of Food Sources,
MCN: The Maximum Cycle Number,
$limit$: The Maximum number of trial for abandoning a source,
M: The Number of Objectives ▷ //Initialization
for $s = 1$ *to* CS **do**
    $X(s) = x_j^{lb} + rand(0,1)(x_j^{ub} - x_j^{lb})$, $j = 1 \ldots D$;
    $f_{si} = f_i(X(s))$,$i = 1 \ldots M$;
    $trial(s) \longleftarrow \emptyset$
end for
$cycle = 1$
while $cycle < MCN$ **do**
    ▷ //Employed Bees' Phase
    for $s = 1$ *to* CS **do**
        $x' \longleftarrow$ a new solution produced by Eq. 2.1
        $f_i(x') \longleftarrow$ evaluate objectives of new solution, $i = 1 \ldots M$
        $X(CS + s) = x'$
    end for
    $\mathcal{F} \longleftarrow$ Non-dominated sorting(X)
    Select best CS solutions based on rank and crowding distance to form new population
    Assign fitness values to solutions depending on the cost values defined by Eq. 2.6
    Calculate probabilities for onlookers by (2.4)
    ▷ //Onlooker Bees' Phase
    $s = 0$
    $t = 0$
    while $t < CS$ **do**
        $r \longleftarrow rand(0,1)$ ▷ //Stochastic sampling
        if $r < p(s)$ **then**
            $t = t + 1$
            $x' \longleftarrow$ a new solution produced by Eq. 2.1
            $f_i(x') \longleftarrow$ evaluate objectives of new solution, $i = 1 \ldots M$
            $X(CS + s) = x'$
        end if$s = (s + 1) \ mod \ (CS - 1)$
    end while
    $\mathcal{F} \longleftarrow$ Non-dominated sorting(X)
    ▷ //Scout bee phase
    $mi = \{s : trial(s) = max(trial) \wedge X(s) \notin \mathcal{F}\}$
    if $trial(mi) > limit$ **then**
        $X(mi) = x_j^{lb} + rand(0,1)(x_j^{ub} - x_j^{lb})$, $j = 1 \ldots D$
        $f_{i,mi} = f_i(X(mi))$
        $trial(mi) \longleftarrow \emptyset$
    end if
    $cycle++$
end while

Another multi-objective ABC applies non-dominated sorting (S-MOABC/NS) on the pool of solutions generated after the employed bee phase and again after the onlooker bee phase [9]. (S-MOABC/NS) employs a ranking selection method [12] and a niche method. Steps of S-MOABC/NS are given in Alg. 3. S-MOABC/NS can generate efficient, well distributed and high quality Pareto-front solutions.

2.4 Application of ABC algorithm for software requirement selection

2.4.1 Problem description

For a large scale software project, developing the software in an incremental manner may reduce the effects of a moving target problem due to changing requirements by time. In incremental development, it is essential to assign requirements to the next release, which is called a Next Release Problem (NRP) [13, 14]. The main goal in selecting the requirements is minimizing the cost and maximizing the customer satisfaction under resource constraints, budget bound and requirement dependencies. It is an $\mathcal{NP} - hard$ problem which means an exact solution cannot be produced in a polynomial time. Therefore, approximation algorithms can be used to produce sub-optimal solutions in polynomial time without violating the budget constraint.

2.4.2 How can the ABC algorithm be used for this problem?

The next release problem is a high dimensional, binary and constrained optimization problem. Therefore, the basic ABC algorithm may not be applied directly. Some modifications in decoding solutions and local search operator are required and a constraint handling method should be adopted to favour the feasible solutions whose satisfied requirements' costs comply with the budget limit.

2.4.2.1 Objective function and constraints

Let R be the set of requirements where $|R| = m$ and C be set of customers where $|C| = n$. Each customer $c_i \in C$ has an importance $\omega_i \in Z^+$ and has a set of requirements $R_i \subset R$. Each $r_i \in R$ has a cost $cost_i$ and dependencies in R can be represented by an acyclic graph $G = (R, E)$ where edge $(r, r') \in E$ means r is a prerequisite of r'. Since $(r, r') \in E \& (r', r'') \in E \Rightarrow (r, r'') \in E$, G is transitive. Parents of R_i are the set of all prerequisites of each requirement in R_i, $parents(R_i) = \{r \in R | (r, r') \in E, r' \in R_i\}$. Therefore, in order to meet the requirements of the customer, c_i, in addition to R_i, $parents(R_i)$ should also be satisfied. $\hat{R}_i = R_i + parents(R_i)$. The company needs to select a subset of customers, $S \subset C$, in order to maximize total profit without violating the budget constraint. This constrained optimization problem can be expressed as follows:

$$\begin{array}{ll} \max & \sum_{i \in S} \omega_i \\ subject\ to & cost(\bigcup_{i \in S} \hat{R}_i) \leq b \end{array} \tag{2.7}$$

When the requirements of customers are independent, it means $R_i \cap R_j = \emptyset, i \neq j$ and $\hat{R}_i = R_i$. In this case the problem is called a basic next release

problem. In order to formulate the problem, we define a boolean variable $x_i \in 0, 1, i = 1 \ldots n$ for each customer to indicate whether the customer c_i will be a selected customer ($x_i = 1$) whose requirements are satisfied or not ($x_i = 0$).

$$\begin{aligned} &\max && \sum_{i=1}^{n} \omega_i x_i \\ &\textit{subject to} && \sum_{i=1}^{n} cost(\hat{R}_i) x_i \leq b \end{aligned} \tag{2.8}$$

The main goal is to find $\vec{x}$ that maximizes the profit under the budget constraint.

2.4.2.2 Representation

Because each x_i is in the set $\{0, 1\}$, the next release problem is a constrained binary problem. In this application, in order to exploit the search space efficiently by numeric local search operators, the solutions in the population are encoded with continuous representation and they are decoded to binary values in the objective value calculation by rounding to 0 or 1.

2.4.2.3 Local search

The dimensions of next release problems are high due to the high number of customers. Hence, for an efficient and fast convergence, we used the local search defined by Eq. 2.1.

2.4.2.4 Constraint handling and selection operator

In the study, we adopted a penalty approach to convert the constrained problem into an unconstrained one. The budget values over the upper limit (b) are added to the cost function to avoid infeasible solutions violating the constraints. Since the resulting problem is unconstrained, a greedy selection scheme is applied as in the basic ABC algorithm.

2.4.3 Description of the experiments

In the study, realistic next release problem instances collected from open source bug repositories in [14, 15] are used. We abide by the instance names in the web source. There are 12 realistic next release problem instances in the set. The number of customers, the number of requirements and the total cost of each problem are given in Table 2.1. The budget upper limit is 30 and 50 percent of the total cost of the problems.

Control parameters of the ABC algorithm are the number of food sources, maximum cycle number, *limit* and modification rate (MR). We set 100 for the number of food sources, 100 for *limit*, 0.25 for MR and 5000 for the maximum cycle number.

TABLE 2.1
Properties of the Realistic NRP instances used in the experiments.

Problem	Number of customers(D)	Number of Req.s	Total Cost
nrp-e1	536	3502	13150
nrp-e2	491	4254	15928
nrp-e3	456	2844	10399
nrp-e4	399	3186	11699
nrp-g1	445	2690	13277
nrp-g2	315	2650	12626
nrp-g3	423	2512	12258
nrp-g4	294	2246	10700
nrp-m1	768	4060	15741
nrp-m2	617	4368	16997
nrp-m3	765	3566	13800
nrp-m4	568	3643	14194

2.4.4 Results obtained

The results obtained by the ABC algorithm for the software requirements selection problem are presented in Table 2.2. Each instance is analysed with the budget upper limits which are 30 and 50 percent of the total cost and each line in Table 2.2 gives the statistics of 30 runs. Satisfied costs and profit values

TABLE 2.2
Results obtained by the ABC algorithm on realistic NRP instances.

				Profit		
Percentage	Problem	Budget	Satisfied Costs (Best)	Best	Mean	Std. Dev.
0.3	nrp-e1	3945	3944	7140	7101.8	25.7026
	nrp-e2	4778.4	4775	6603	6565.6	24.6901
	nrp-e3	3119.7	3118	6166	6142.8	13.2061
	nrp-e4	3509.7	3506	5365	5347.4	11.0975
	nrp-g1	3983.1	3982	5879	5858.8	15.2009
	nrp-g2	3787.8	3787	4349	4336.6	9.0086
	nrp-g3	3677.4	3673	5645	5621.3	14.0479
	nrp-g4	3210	3210	4040	4028.7	6.8807
	nrp-m1	4722.3	4714	8904	8768.7	81.2076
	nrp-m2	5099.1	5081	7272	7214.3	23.8097
	nrp-m3	4140	4136	8311	8216.5	50.6387
	nrp-m4	4258.2	4253	6517	6466.2	36.8203
0.5	nrp-e1	6575	6569	10053	10025.8	11.9331
	nrp-e2	7964	7956	9231	9204.3	11.3827
	nrp-e3	5199.5	5199	8656	8648.3	6.7831
	nrp-e4	5849.5	5842	7495	7485.5	7.0435
	nrp-g1	6638.5	6637	8537	8508.8	12.4615
	nrp-g2	6313	6309	6212	6203.5	6.2937
	nrp-g3	6129	6121	8156	8142.5	7.4722
	nrp-g4	5350	5345	5787	5777.2	8.804
	nrp-m1	7870.5	7871	12976	12936.6	28.1512
	nrp-m2	8498.5	8489	10510	10463.6	25.1891
	nrp-m3	6900	6894	12447	12389.1	27.2782
	nrp-m4	7097	7094	9670	9638.7	18.8034

of the best of 30 runs and the mean and standard deviation of the profit values of 30 runs are also presented in Table 2.2. As seen from the results, the ABC algorithm can find high profitable and feasible solutions which do not violate budget constraints on a large-scale realistic software requirements selection problem. When the budget upper limit constraint is relaxed to 50% of total budget, more profitable solutions are obtained in terms of both the best and mean values. On all problems except for nrp-g4, the standard deviation is less and the algorithm is more stable when the budget is relaxed.

From the results, it can be concluded that, the ABC algorithm can determine software requirements resulting in the maximum profit without violating the budget constraint.

2.5 Conclusions

In this chapter, the ABC algorithm and its modifications according to the problem characteristics are presented. Local search modifications to enhance the convergence rate are described. Combinatorial local search operators integrated in the ABC algorithm to solve combinatorial problems are summarized. The nature of the constrained problems and the adaptations of the ABC algorithm to generate feasible solutions or to prefer the solutions in the feasible region are explained. ABC algorithm variants that can handle problems with more than one objective function are presented. In Section 2.4, the ABC algorithm is applied to a binary and constrained software requirement selection problem in which the aim is to minimize the cost and maximize the customer satisfaction under resource constraints, budget bound and requirement dependencies. The representation, objective function and constraint definition, local search and selection operators that can be used for this problem are mentioned in the chapter. Some large-scale realistic problems collected from the repositories are employed in the experiments. Based on the experiments performed, it can be concluded that with minor modifications in the basic ABC algorithm, it can be easily adapted to solve the problem and to select requirements that yield high profit and do not violate budget limits.

References

1. D. Karaboga. An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
2. D. Karaboga, B. Akay. A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, 214:108-132, 2008.

3. B. Akay, D. Karaboga. A modified artificial bee colony algorithm for real-parameter optimization. *Information Sciences*, 192: 120-142, 2012.
4. D. Karaboga, B. Gorkemli. A quick artificial bee colony (Qabc) algorithm and its performance on optimization problems, 2012 International Symposium on Innovations in Intelligent Systems and Applications (INISTA), pp.227-238, 2012.
5. B Akay, E. Aydogan, and L. Karacan. 2-opt based artificial bee colony algorithm for solving traveling salesman problem. Proceedings of 2nd World Conference on Information Technology (WCIT-2011), pp. 666-672, 2011.
6. D. Karaboga and B. Gorkemli. A combinatorial artificial bee colony algorithm for traveling salesman problem. In *Innovations in Intelligent Systems and Applications (INISTA), 2011 International Symposium on*, pages 50-53, June 2011.
7. K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2- 4), 311-338, 2000.
8. D. Karaboga and B. Akay. A modified artificial bee colony (abc) algorithm for constrained optimization problems. *Applied Soft Computing*, 11(3):3021-3031, 2011.
9. B. Akay. Synchronous and asynchronous pareto-based multiobjective artificial bee colony algorithms. *Journal of Global Optimization*, 57(2):415-445, October 2013.
10. N. Srinivas, K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2: 221-248, 1994.
11. X. Zou, Y. Chen, M. Liu, L. Kang. A new evolutionary algorithm for solving many-objective optimization problems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 38(5): 1402-1412, 2008.
12. K. Deb, A. Pratap, S. Agarwal, T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2): 182-197, 2002.
13. A. Bagnall, V. Rayward-Smith, and I. Whittley. The next release problem, *Information and Software Technology*, 883-890, 2001.
14. J. Xuan, H. Jiang, Z. Ren, Z. Luo. Solving the large scale next release problem with a backbone based multilevel algorithm, *IEEE Transactions on Software Engineering*, 38(5): 1195-1212, 2011.
15. http://researchers.lille.inria.fr/~xuan/page/project/nrp/