I.J. Intelligent Systems and Applications, **2014, 04, 40-48**
Published Online March 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2014.04.04

Redundancy Level Optimization in Modular Software System Models using ABC

Tarun Kumar Sharma
Amity Institute of Information Technology, Amity University Rajasthan, Jaipur, India
E-mail: taruniitr1@gmail.com

Millie Pant
Department of Applied Science and Engineering, IIT Roorkee, Roorkee, India
E-mail: millidma@gmail.com

Abstract— The performance of optimization algorithms is problem dependent and as per no free lunch theorem, there exists no such algorithm which can be efficiently applied to every type of problem(s). However, we can modify the algorithm/ technique in a manner such that it is able to deal with a maximum type of problems. In this study we have modified the structure of basic Artificial Bee Colony (ABC), a recently proposed metaheuristic algorithm based on the concept of swarm intelligence to optimize the models of software reliability. The modified variant of ABC is termed as balanced ABC (B-ABC). The simulated results show the efficiency and capability of the variant to solve such type of the problems.

Index Terms— Artificial Bee Colony, Software reliability, Optimization, Metaheuristics, Swarm Intelligence

I. Introduction

Metaheuristic search techniques are gaining significant attention and interest of researchers, academicians and scientists to find optimal solutions to optimization problems at a reasonable computational cost in various domains of Science and Engineering. The focus of the present study is on the optimization problems arising in the field of Software Engineering, an emerging area of research where metaheuristics are applied. Software is a strong foundation of Information Technology and Society and developing the strategic competence among nationalities [1 - 2]. Drastic working and life style change can be experienced thoroughly after the emergence of Software. However developing the quality software is a very careful practice. The development posses many issues and goals like reliability, overrun of costs, user requirements etc. The improvement of software process has important practical significance to defuse software crisis, as it is influencing the development and management of software [3].

A lot of research has been done and is still continuing to overcome the various issues in software development process. A brief discussion is given in the literature review section of the present study.

The present study focuses on the optimization reliability issues of software development process, which is also one of the important user requirements. In terms of software system, reliability can be defined as the probability that software operates without failure in a specified environment, during a specified exposure period [4]. A discrepancy between expected and actual output is called failure. Failure is a consequence of fault, also called a defect in the program that, when executed results a failure. The reliability of the software can be improved by carefully implementing the application of redundancy, but it requires additional resources. A number of reliability models have been proposed and developed for the prediction and the assessment of the reliability of fault-tolerant software systems[5]. However, the problem of reliability optimization for fault-tolerant software has not been addressed by many researchers else in [5] which contribute this lack of interest partly.

To optimize the redundancy level of the modules [6] we have modified the searching behavior and have used a recently proposed metaheuristic, Artificial Bee Colony (ABC) algorithm.

ABC is a swarm based intelligent algorithm, proposed by Karaboga in 2005 [6]. It uses the intelligent and social movement of three kind of bees (scout, employed and onlooker) to simulate the optimization problems. The algorithm is detailed with pseudocode in section2. The flexibility, efficiency and ease of implementation of ABC attract many researches to utilize its functionality to solve many real life applications/problems.

Here we would like to mention that a preliminary version of this paper has been presented in Seventh International Confe rence, Bio-Inspired Computations, Theory and Applications (BIC-TA 2012), where the proposed variant is tested on a set of five benchmark functions [7]. This paper is an extended version of the presented paper in terms of both the algorithm as well as its implementation on real life problems. In this paper the scout bee phase of the variant is further modified to increase the diversity of the food sources.

The paper is structured as follows: Section 3 describes applications of Artificial Bee Colony algorithm in the software engineering discipline; the proposed enhanced variant, B-ABC, of basic ABC is discussed in Section 4. In Section 5 software system structure with models are discussed. Parameter settings and simulated results are presented Section 6 and finally the conclusions are drawn in Section 7.

II. Delineation of Artificial Bee Colony

Artificial bee colony (ABC) is a recently proposed population based metaheuristic algorithm which takes its inspiration from the intelligent foraging behavior of swarm of real honey bees. It was initially proposed by Karaboga in 2005 [6] for solving unconstrained optimization problems, where it showed a better performance in comparison to genetic algorithm (GA), particle swarm optimization (PSO) and differential evolution (DE) when applied to various function optimization problems [9][10]. Later in 2007 [11], ABC, modified with small changes was used to solve unconstrained optimization problems. Constrained and unconstrained ABC versions are discussed below in detail.

2.1 Unconstrained ABC

In ABC the colony of honey bees is comprised of three types of bees namely scouts, employed and onlooker bees. The bees in the colony perform tasks like searching for the nector and sharing the information about the food source intelligently by dividing the labor themselves. The main difference between ABC and other intelligent swarm based algorithms is that in ABC food sources (the population generated) represents the solutions of the problem, not the bees.

The scout bee initiates the food sources randomly which is later exploited by employed bees. The employed bees pass the information about the food source based on their nectar quality to the onlooker bees waiting in the hive. This sharing of information is done by performing a special dance called waggle dance. In ABC the number of employed bees is equal to the number of food sources and each employed bee is assigned to one of the food sources. Employed bees upon reaching to the food source, calculate a new location or fly to the nearby position from the old and preserve the best position. This is a greedy selection process. The number of onlooker bees is also the same as that of employed bees and are allocated to the food sources based on their profitability. Similarly as the employed bees, onlooker bees also calculate the new position from the old one. If the food source does not improve after predetermined number of iterations, then employed bees abandons that food source and becomes scouts and searches the new food source randomly. Mathematical explanation of the complete process is described below.

Mathematical Outline of Artificial Bee Colony:

Define *SN* as the total number of bees, N_e as the colony size of the employed bees and N_o as the size of onlooker bees, which satisfy the equation $SN = N_e + N_o$. The number of food sources is equal to the number of employed bees because each food source is exploited by only one employed bee around the hive. The standard ABC algorithm can be expressed as follows:

1. Randomly initialize a set of feasible food sources (x_1; . . . ; x_{SN}), and the specific solution x_i can be generated by:

$$x_{ij} = x_{Lj} + rand(0,1)(x_{Uj} - x_{Lj}) \tag{1}$$

where $j \in \{1,2,\ldots,D\}$ is the jth dimension of the solution vector. Calculate the fitness value of each solution vector respectively.

2. For an employed bee in the nth iteration $x_i(n)$, search new solutions in the neighborhood of the current position vector according to the following equation:

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \tag{2}$$

where $x_{ij} \in SN$, $j \in \{1,2,\ldots,D\}$, $k \in \{1,2,\ldots,N_e\}$, $k \neq i$. is a random number between -1 and 1.

3. Apply the greedy selection operator to choose the better solution between searched new vector v_{ij} and the original vector x_{ij} into the next generation. The greedy selection operator ensures that the population is able to retain the elite individual, and accordingly the evolution will not retreat.

4. Each onlooker bee selects an employed bee from the colony according to their fitness values. The probability distribution (p_i) of the selection operator can be described as follows:

$$p_i = \frac{fit_j}{\sum_{i=1}^{N_e} fit} \tag{3}$$

where fit_i is the fitness value of the solution i which is proportional to the nectar amount of the food source in the position *i*.

5. The onlooker bee searches in the neighborhood of the selected employed bee's position to find new solutions using Eq. (2). The updated best fitness value can be denoted with f_{best}, and the best solution parameters.

6. If the searching times surrounding an employed bee exceeds a certain threshold limit, but still could not find better solutions, then the location vector can be re-initialized randomly according to the Eq. (1).
7. If the iteration value is larger than the maximum number of the iteration then stop, else, go to 2.

2.2 Constrained ABC

In the present study, we have followed 'three feasibility rules' method given in [12] to decide which solution vector (food source) will be beneficial for handling constraints. An advantage of this method is that unlike penalty method we need not have a penalty constant, which itself is a tedious work to decide. Moreover, here we consider feasible as well as infeasible solutions and prioritize these solutions as per the following rules:

- If we have two feasible food sources, the one giving the best objective function value is selected.
- If one food source is feasible and the other one is infeasible, the feasible one is selected;
- If both food sources turn out to be infeasible, then we selected the food source giving the minimum constraint violation.

It can be observed that these rules bias feasible food sources over infeasible food sources and a pairwise comparison (tournament selection) is done to select the best option.

In this method a control parameter called modification rate (MR), pre defined by the user is introduced. With the help of MR, it is decided stochastically whether a food source x_i should be retained or not. It given by the equation:

$$v_{ij} = \begin{cases} x_{ij} + \phi_{ij}(x_{ij} - x_{kj}), & if \ R_j \le MR \\ x_{ij} \ , & otherwise \end{cases} \tag{4}$$

where k $\in \{1, 2, \ldots, SN\}$ – randomly chosen such that $k \neq i$; j {1, 2,...,N}.R_j is generated randomly between 0 and 1in each iteration.

III. Applications of Artificial Bee Colony in Software Engineering: An Overview

Since its development, Artificial Bee Colony algorithm is widely applied to solve/optimize various real life applications in different disciplines of science and engineering. The detail can be consulted in [13]. In the field of software engineering Artificial Bee Colony has been widely applied to software testing, cost estimation, and software reliability. Brief descriptions of the applications of ABC in software engineering are listed below:

- In [14] and [15] Mala et al. applied ABCA to very promising research area in software engineering i.e. optimization of software test suite.
- Bacanin et al. in [16] proposed modified variant of basic ABCA to describe an object-oriented software system for continuous optimization.
- Dahiya et al. [17] introduced automatic structural software tests generation using a novel search technique based on Artificial Bee Colony.
- Kilic et al. [18] presented a solution for solving hard combinatorial automated software refactoring problem which is the lies in the domain of search-based software engineering.
- AdiSrikanth et al. [19] introduced optimal software test case generation to attain better path coverage using ABCA algorithm.
- Liang and Ming in [20] discussed and studied the use of ABCA optimization technique with two-tier bitwise interest oriented QRP to reduce message flooding and improve recall rate for a small world peer-to-peer system.
- Suri and Kalkal in [21] presented a review of software testing applications of ABCA and its variants.
- Li and Ma in [22] a solution method for logic reasoning using ABCA was proposed.
- Bacanin et al. in [23] improved ABCA optimization was studied on the performance of object-oriented software system.
- Sharma et.al. in [24] applied modified version of ABCA to parameter estimation of software reliability growth models.
- Koc et al. in [25] proposed a solution for automated maintenance of object-oriented software system designs via refactoring using ABCA.
- Sharma and Pant in [26] estimated the software cost parameters using halton based ABCA optimization.
- Tajinder Singh and Mandeep Kaur Sandhu in [27] presented a survey of ABCA on software testing environment and its advantages over the Genetic Algorithm
- Suri, B., and Mangal, I.in [28] introduced a hybrid algorithm using ABCA to reduce the test suite.

IV. Proposed Enhanced Variant: B-ABC

Exploration and exploitation are the two antagonist upon which the success of any evolutionary algorithm relies. Artificial Bee Colony, like other evolutionary algorithms has some downside which impedes its performance. Artificial Bee Colony algorithm performs well at exploration while poor at exploitation. This issue motivates to make an attempt to balance exploration and exploitation. In this study it is done by enhancing the search mechanism of the onlooker bees in the solution search space. In the proposal the probability of each parent to generate a better food source is increased by allowing each location of food source (solution) to generate more than one food source. This is done by using different mutation operator that incorporate information of the best food source in the current population and as well as the information of the current parent to define the new search directions. This mutation operator allows each parent to generate more than one food sources in the same generation. The pseudocode of the proposed variant to generate multiple food sources is given below.

Pseudocode of B-ABC

```
For k =1 to q
        Select randomly r1≠r2≠r3≠i
        For j = 1 to D
                If (U(0, 1))>0.5 Then
        $v_{ij} = x_{r3,j} + \alpha(x_{best,j} - x_{r2,j}) + \beta(x_{ij} - x_{r1j})$
                Else
                $v_{ij} = x_{ij}$
                End If
        End For
        If K > 1 Then
                If ($f(v_{ij}) < f(x_{ij})$) Then
                        $x_{ij} = v_{ij}$
                        End If
                Else
                $x_{ij} = v_{ij}$
                End If
End For
```

The pseudocode describes the generation of multiple food sources, where q is user defined parameter to keep the best food source generated. r_1, r_2 & r_3 are randomly chosen random numbers such that $r_1 \neq r_2 \neq r_3 \neq i$. $x^j_{i,G}$ and $x^j_{best,G}$ is the current parent and best individual in the current population G of the food sources respectively.

The influence of the best and parent food sources (solutions) is indicated by α and β factors, respectively, in the search direction of the food sources.

Further, to increase the diversity of the food sources (population), the scout bee generates a new food source x_{ij} by using the food source subject to be replaced x_{ij} as a base to generate a new search direction biased by the best food source so far $x_{best,\,j}$ and a randomly chosen food source x_{jk}. The modified scout bee phase is given in equation (5):

$$x_{ij} = \begin{cases} x_{ij} + rand()(x_{i,\max} - x_{i,\min}, & if\ U(0,1) \\ x_{ij} + \phi_{ij}(x_{kj} - x_{ij}) + (1-\phi)(x_{ij} - x_{best,j}), & otherwise \end{cases} \tag{5}$$

V. Software Redundancy Models

Notations:

K:	is the number of functions that the software system is required to perform.
n:	Number of modules within the system.
F_k:	Frequency of the use of function k, k = 1, 2,. . . , K.
m_i:	is the number of available versions for module i,i = 1,. . . , n.
R_{ij}:	Reliability estimation of version j of module i.
X_{ij}:	Binary variable i.e. 1 if version j is selected for module i, else 0.
R_i:	Estimated reliability of module i.
R:	Estimated reliability of the software system.
C_{ij}:	Development cost for version j for module i.
B:	Available budget.

In this study four different software models are taken from literature [6] to optimize the redundancy level of the modules for each software structure so as to maximize reliability at a limited given cost. The models are developed using modular techniques and are required to perform one or more, user specified functions discussed. A program when executed performs some function(s) and each program is further divided into several modules. Each module may be called or referred by more than one program. Four models for different software structures are discussed below:

5.1 Model 1

In this model an optimal set of modules are selected to perform one function without redundancy. In software system that comprises of a single program, when executed performs one major function. And in turn a program is comprised of modules that executes sequentially. Budget constraint and the critical nature of software limits keeping more than one versions of each module, though there are more than one versions of each module are available. The model that describes the situation of optimal selection of modules for a single program in order to optimize (maximize) reliability with respect to the development budgetary constraints is given below:

$$Maximize\ R = \prod_{i=1}^{n} R_i \qquad (6)$$

w.r.t. $\sum_{j=1}^{m_i} X_{ij} = 1,$

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} X_{ij} C_{ij} \le B$$

$$X_{ij} = 0,1;\quad i = 1,\ldots,n;\quad j = 1,\ldots,m_i$$

where $R_i = \sum_{j=1}^{m_i} X_{ij} R_{ij}$ (6a)

5.2 Model 2

Here in this model an optimal set of modules are selected to perform one function with redundancy. A software system that performs critical function and whose failure results to be very stern. In such condition software are designed to be fault-tolerant. For this purpose multiple redundant versions for each module are kept. In such condition it can be reasonably understood that the allocation of budget is large enough to support and allow redundancy of modules. The key objective that arises in this situation is to determine the optimal set of modules, allowing redundancy, and to maximize the reliability of the software system with the budgetary constraint. The model is presented below:

$$Maximize\ R = \prod_{i=1}^{n} R_i \qquad (7)$$

w.r.t. $X_{ij} = 0,1;\quad i = 1,\ldots,n;\quad j = 1,\ldots,m_i$

where $\sum_{j=1}^{m_i} X_{ij} \ge 1,$

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} X_{ij} C_{ij} \le B$$

$$R_i = 1 - \sum_{j=1}^{m_i} (1 - R_{ij})^{X_{ij}} \qquad (7a)$$

The probability that at least one of the mi, versions is performing correctly defines the reliability of the module i (given as one minus the probability that none of the m_i, versions is performing correctly).Constraint set mentioned for the model assures that for each module i at least one version is selected.

5.3 Model 3

In the model 3 the set of modules without having redundancy, are selected for a system with K functionality. This model comprises of several programs and each program performs a specific function. In turn, each program comprises of a series of modules. Programs can be called by their corresponding functions and program may call any modules. The objective of this model is again just like discussed above in two models i.e. to determine the optimal set of modules for the programs, without allowing redundancy, and in such a way that the reliability of the software system is maximized within the budgetary constraints. Model 3 is presented below:

Let S_k symbolize the set of modules corresponding to program k. For each module $i \in S_k$ there are mi, versions available. Here different programs can call the same module. All the modules to be called by all programs are numbered from 1 to n. The problem can be formulated as follows:

$$Maximize\ R = \sum_{k=1}^{K} F_k \prod_{i \in S_k} R_i \qquad (8)$$

w.r.t. $\sum_{j=1}^{m_i} X_{ij} = 1, \qquad i = 1,\ldots.,n$

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} X_{ij} C_{ij} \le B$$

$$X_{ij} = 0,1;\quad i = 1,\ldots,n;\quad j = 1,\ldots,m_i$$

where R_i is referred from Model 1.

5.4 Model 4

This model is similar to the model 3 above. In this case models are selected with redundancy i.e., the choice of more than one version for each one of the modules are allowed. The problem is presented as follows:

$$Maximize\ R = \sum_{k=1}^{K} F_k \prod_{i \in S_k} R_i \qquad (9)$$

w.r.t. $\sum_{j=1}^{m_i} X_{ij} \ge 1, \qquad i = 1,\ldots.,n$

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} X_{ij} C_{ij} \le B$$

$$X_{ij} = 0,1;\quad i = 1,\ldots,n;\quad j = 1,\ldots,m_i$$

where R_i is referred from Model 2.

VI. Parameter Settings, Results and Discussions

6.1 Experimental Settings

For the best performance of B-ABC following parameters are taken. The colony size (SN) or the number of solutions in the colony is 40, the value of modification rate (MR) is 0.4, and the maximum cycle number (MCN) is 1000. The total objective function evaluation number is SN × MCN i.e. 40,000 and the value of limit is equal to SN × N where N is the dimension of the problem. In this paper, 30 independent runs are performed for each test function in C++. The integer and binary variables are handled by rounding of the decision variables to nearest integer [29]. In this study we have generalize the maximization problems as minimization ones. The process is represented graphically in Fig. 2, where the optimal value of the minimization problem (x^*) is both is both, the minimum of the function $f(x)$ and the maximum of the function $-f(x)$.

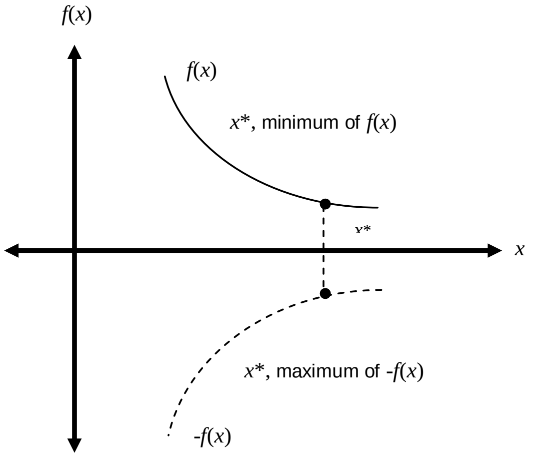

Fig. 1: Minimum of $f(x)$

6.2 Result Analysis

To solve the above discussed software system structure models, numerical examples have been taken.

1) Numerical example for the Model 1 is as: Let there are 3 modules in the software i.e. n = 3 (m_1=3, m_2=3, m_3=3). The cost and the reliability of the modules are taken as R_{11}=0.90, R_{12}=0.80, R_{13}=0.85; C_{11}=\$3, C_{12}=\$1, C_{13}=\$2; R_{21}=0.95, R_{22}=0.80, R_{23}=0.70; C_{21}=\$3, C_{22}=\$2, C_{23}=\$1; R_{31}=0.98, R_{32}=0.94; C_{31}=\$3, C_{32}=\$2 and given budget i.e. B = 6, the Model 1 can be formulated as follows (equation (10)):

Maximize $(0.9X_{11} + 0.8X_{12} + 0.85X_{13})*(0.95X_{21} + 0.8X_{22} + 0.7X_{23})*(0.98X_{31} + 0.94X_{32})$ (10)

w.r.t. $X_{11} + X_{12} + X_{13} = 1$

$X_{21} + X_{22} + X_{23} = 1$

$X_{31} + X_{32} = 1$

$3X_{11} + X_{12} + 2X_{13} + 3X_{21} + 2X_{22} + X_{23} + 3X_{31} + 2X_{32} \leq 6$

where

$X_{11}, X_{12}, X_{13}, X_{21}, X_{22}, X_{23}, X_{31}, X_{32} = 0,\ 1.$

The optimal solution found by ABC and the proposed variant called B-ABC is presented in the Table 1. The objective function value obtained using basic B-ABC is better than that of basic ABC. Also B-ABC took only 27913 function evaluations to solve model 1 which is about 30% faster than ABC.

Table 1: Optimal Solution of Model 1

Algorithm	Decision Variables	Obj. Func. Value	Cost (\$)	Func. Eval. Number
ABC	X_{31}, X_{22}, X_{12}	0.6272	6	36176
B-ABC	X_{12}, X_{21}, X_{32}	0.714	6	27913

2) The same problem discussed above is considered for solving Model 2, with a difference of budget only. Here the budget is taken as \$10. The optimal solution of the model using ABC and modified variant is given in Table 2. In this case B-ABC performs 34% faster than basic ABC in achieving objective function value.

Table 2: Optimal Solution of Model 2

Algorithm	Decision Variables	Obj. Func. Value	Cost (\$)	Func. Eval. Number
ABC	$X_{11}, X_{12}, X_{22}, X_{23}, X_{31}$	2.499	10	36219
B-ABC	$X_{31}, X_{21}, X_{23}, X_{12}, X_{13}$	2.6680	10	26984

3) To solve Model 3, the numerical example considered is as follows: K = 2, F_1 = 0.70, F_2 = 0.30, n = 3, s_1 = {1, 2}, s_2 = {2, 3}; m_1 = 2, m_2 = 2, m_3 = 2; R_{11} = 0.80, R_{12} = 0.85; R_{21} = 0.70, R_{22} = 0.90; R_{31} = 0.95 R_{32} = 0.90; C_{11} = \$2, C_{12} = \$3; C_{21} = \$1, C_{22} = \$3; C_{31} = \$4, C_{32} = \$3 and the available budget is taken as \$8. Mathematically, Model can be formulated as:

Maximize $(0.392X_{11}X_{12} + 0.504X_{11}X_{22} + 0.4165X_{12}X_{21} + 0.5355X_{12}X_{22} + 0.1995X_{21}X_{31} + 0.189X_{21}X_{32} + 0.2565X_{22}X_{31} + 0.243X_{22}X_{32})$ (11)

w.r.t. $X_{11} + X_{12} = 1$

$X_{21} + X_{22} = 1$

$X_{31} + X_{32} = 1$

$2X_{11} + 3X_{12} + X_{21} + 3X_{22} + 4X_{31} + 3X_{32} \leq 8$

where

$X_{ij} = 0,1;\ i = 1, 2, 3 \quad j = 1, ..., m_i.$

In this model both the algorithms achieved the same optimal solution and is presented in Table 3.But B-ABC as above again performed better in terms of function evaluation i.e. 20% faster than basic ABC.

Table 3: Optimal Solution of Model 3

Algorithm	Decision Variables	Obj. Func. Value	Cost ($)	Func. Eval. Number
ABC	X_{11}, X_{22}, X_{32}	0.747	8	39134
B-ABC	X_{11}, X_{22}, X_{32}	0.747	8	32546

4) For the Model 4 the same problem discussed above is again considered with the budget of $9 and the optimal result for this model is presented in Table 4. Here again the proposed variant performed better in achieving better objective function value as well as better convergence of 19% when compared with basic ABC.

Table 4: Optimal Solution of Model

Algorithm	Decision Variables	Obj. Func. Value	Cost ($)	Func. Eval. Number
ABC	X_{11}, X_{21}, X_{22} , X_{32}	0.8052	9	39034
B-ABC	X_{11}, X_{12}, X_{21}, X_{32}	0.9975	9	32612

The number of function evaluation taken to execute the considered Models 1 - 4 using ABC and B-ABC are shown graphically in Fig. 2.

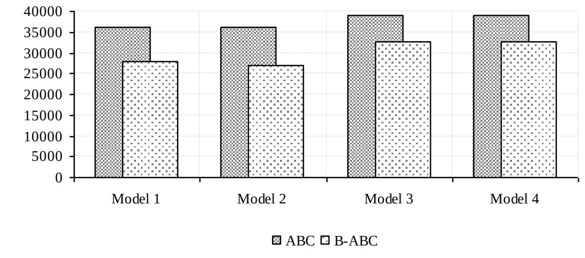

Fig. 1: Function Evaluation Number taken to solve Model 1 - Model 4 using ABC & B-ABC

VII. Conclusion

In this study we have modified the structure of basic ABC by improving the foraging mechanisms of onlooker bee and later the scout bee phase for improving diversity. The modified variant is termed B-ABC. B-ABC is applied to optimize (maximize) the reliability of four modular software system models while ensuring budget remain within limit (available resources). The four different software system models considered to optimize using ABC and B-ABC are (a) Single program without redundancy (b) Single program with redundancy (c) Multiple programs without redundancy and (d) Multiple programs with redundancy. The simulated results explain the efficiency of the proposed variant in terms of accuracy and convergence to solve such type of problems.

Acknowledgments

The authors would like to thank the anonymous reviewers for their careful reading of this paper and for their helpful comments.

References

[1] Popp R L, Pattipati K R, Bar-Shalom Y. m-Best S-D assignment algorithm with application to multitarget tracking[J]. IEEE Trans. on AC, 2001, 37 (1):22 - 38.

[2] Carbone, P., Buglione, L., and Mari, L., “A comparison between foundations of metrology and software measurement”, IEEE T. Instrum. Meas., Vol. 57, No. 2, (2008), pp. 235-241.

[3] Wang, YX. and Patel, S., “Exploring the cognitive foundations of software engineering”, Int. J. Soft. Sci. Comp. Intel., Vol. 1, No. 2, (2009), pp. 1-19.

[4] OHagan, P., Hanna, E. and Territt, R., “Addressing the corrections crisis with software technology”, Comp., Vol. 43, No. 2, (2010), pp. 90-93.

[5] Musa, JD., “A theory of software reliability and its application”, IEEE Transactions on Software Engineering, Vol. 1, No. 3, (2010), pp.312-327.

[6] Belli, F. and Jedrzejowicz, P., “An approach to reliability optimization of software with redundancy”, IEEE Transactions on Software Engineering, Vol. 17, No. 3, (1991), pp. 310-312.

[7] Berman, O., and Ashrafi, N., “Optimization Models for Reliability of Modular Software Systems”, IEEE Transactions on Software Engineering, Vol. 19, No. 11, (1993), pp. 1119-1123.

[8] Karaboga, D., “An idea based on honey bee swarm for numerical optimization”, Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, Turkey (2005).

[9] Sharma, TK. and Pant, M., “Modified Foraging Process of Onlooker Bees in Artificial Bee Colony”, Bio Inspired Computation Theory and Applications (BIC-TA 2012), Gwalior, India (Dec. 14-16, 2012), 479-487, 2012.

[10] Karaboga, D. and Basturk, B., “A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm”, J. Global Optimiz., Vol. 39, No. 3, (2007), pp. 459–471.

[11] Karaboga, D. and Basturk, B., “On the performance of artificial bee colony (ABC) algorithm”, Appl. Soft Comput., Vol. 8, No. 1, (2008), pp. 687–697.

[12] Karaboga, D. and Basturk, B., “Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems”, International Fuzzy Systems Association World Congress (IFSA), Cancun, Mexico, (June 18-21, 2007), 789–798, 2007.

[13] Deb, K., “An efficient constraint handling method for genetic algorithms”, Computer Methods in Applied Mechanics and Engineering, Vol. 186, No. 2/4, (2000) , pp. 311–338.

[14] Karaboga, D., Gorkemli, B., Ozturk, C. and Karaboga, N., “A comprehensive survey: artificial bee colony (ABC) algorithm and applications”, Artif Intell Rev, DOI 10.1007/s10462-012-9328-0.

[15] Mala, DJ., Mohan, V. and Kamalapriya, M., “Automated software test optimisation framework—an artificial bee colony optimisation-based approach”, IET Softw, Vol. 4, No. 5, (2010), pp. 334–348.

[16] Mala, DJ., Kamalapriya, M., Shobana, R. and Mohan, V. “A non-pheromone based intelligent swarm optimization technique in software test suite optimization”, IAMA: 2009 international conference on intelligent agent and multi-agent systems, Madras, India, (July 22-24, 2009), 188–192, 2009.

[17] Bacanin, N., Tuba, M. and Brajevic, I., “An object-oriented software implementation of a modified artificial bee colony (abc) algorithm”, 11th WSEAS neural networks, fuzzy systems and evolutionary computing, Iasi, Romania, (June 13-15, 2010), 179–184, 2010.

[18] Dahiya, SS., Chhabra, JK. and Kumar, S., “Application of artificial bee colony algorithm to software testing”, 21st Australian software engineering conference (ASWEC), Auckland, New Zealand, (April 6-9, 2010),149–154, 2010.

[19] Kilic, H., Koc, E. and Cereci, I., “Search-based parallel refactoring using population-based direct approaches”, Third International Symposium, SSBSE 2011, Szeged, Hungary, (September 10-12, 2011), 271–272, 2011.

[20] Adi Srikanth, Kulkarni, NJ., Naveen, KV., Singh, P. and Srivastava, PR., “Test case optimization using artificial bee colony algorithm”, First International Conference, ACC 2011, Kochi, India, (July 22-24, 2011), 570–579, 2011.

[21] Liang, CY. and Ming, LT., “Using two-tier bitwise interest oriented qrp with artificial bee colony optimization to reduce message flooding and improve recall rate for a small world peer-to-peer system”, 7th international conference on information technology in Asia (CITA 11), Kuching, Sarawak, (July 12-13, 2011) 1–7, 2011.

[22] Suri, B. and Kalkal, S., “Review of artificial bee colony algorithm to software testing”, Int J Res Rev Comput Sci., Vol. 2, No. 3, (2011), pp. 706–711.

[23] Li, LF. and Ma, M., “Artificial bee colony algorithm based solution method for logic reasoning”, Comput Technol Dev, doi:CNKI:SUN:WJFZ.0.2011-06-035.

[24] Bacanin, N., Tuba, M. and Brajevic, I., “Performance of object-oriented software system for improved artificial bee colony optimization”, Int J Math Comput Simul., Vol. 5, No. 2, (2011), pp. 154–162.

[25] Sharma, TK., Pant, M. and Abraham, A., “Dichotomous search in ABC and its application in parameter estimation of software reliability growth models”, IEEE NaBIC 2011, Salamica, Spain, (Oct. 20-22, 2011), 207-212, 2011.

[26] Koc, E., Ersoy, N., Andac, A. and Camlidere, ZS., Cereci, I. and Kilic, H., “An empirical study about search-based refactoring using alternative multiple and population-based search techniques”, 26th International Symposium on Computer and Information Sciences, London, UK, (September 26-28, 2011), 59–66, 2011.

[27] Sharma, TK. and Pant, M., “Halton Based Initial Distribution in Artificial Bee Colony Algorithm and its Application in Software Effort Estimation”, International Journal of Natural Computing Research (IJNCR), Vol. 3, No. 2, (2012), pp. 86 - 106, 2012.

[28] Singh, T. and Sandhu, MK., “An Approach in the Software Testing Environment using Artificial Bee Colony (ABC)”, Optimization. International Journal of Computer Applications, Vol. 58, No. 21, (2012), pp. 5-7.

[29] Suri, B. and Mangal, I., “Analyzing Test Case Selection using Proposed Hybrid Technique based on BCO and Genetic Algorithm and a Comparison with ACO”, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, No. 4, (2012), pp. 206-211.

[30] Srinivas, M. and Rangaiah, GP., “Differential Evolution with Tabu List for Solving Nonlinear and Mixed-Integer Nonlinear Programming Problems”, Ind

Authors’ Profiles

Tarun Kumar Sharma: Assistant Professor in Amity Institute of Information Technology, Amity University Rajasthan, India. His research areas are evolutionary and Swarm intelligence algorithms and their applications in Software Engineering. He is in Editorial Board and reviewer of many refereed Journals. He has published about 40 research papers in Journal of repute and in refereed international Journals.

Millie Pant: Associate Professor, Indian Institute of Technology, Roorkee, India. She has published above 200 research publications in referred journals and international conferences. She has been keynote speakers to various seminars, conferences and development programs. Her key research areas are Evolutionary Computing, Swarm Intelligence and their application in various areas of Engineering.

How to cite this paper: Tarun Kumar Sharma, Millie Pant,"Redundancy Level Optimization in Modular Software System Models using ABC", International Journal of Intelligent Systems and Applications(IJISA), vol.6, no.4, pp.40-48, 2014. DOI: 10.5815/ijisa.2014.04.04