Adaptive Binary Artificial Bee Colony Algorithm

Rafet Durgut[a], Mehmet Emin Aydin[b]

[a]*Karabuk University, Dept. of Computer Engineering, Karabuk, Turkey*
rafetdurgut@karabuk.edu.tr
[b]*UWE Bristol, Dept. of Computer Science and Creative Technologies, Bristol, UK*
mehmet.aydin@uwe.ac.uk

Abstract

Metaheuristics and swarm intelligence algorithms are bio-inspired algorithms, which have long standing track record of success in problem solving. Due to the nature and the complexity of the problems, problem solving approaches may not achieve the same success level in every type of problems. Artificial bee colony (ABC) algorithm is a swarm intelligence algorithm and has originally been developed to solve numerical optimisation problems. It has a sound track record in numerical problems, but has not yet been tested sufficiently for combinatorial and binary problems. This paper proposes an adaptive hybrid approach to devise ABC algorithms with multiple and complementary binary operators for higher efficiency in solving binary problems. Three prominent operator selection schemes have been comparatively investigated for the best configuration in this regard. The proposed approach has been applied to uncapacitated facility location problems, a renown NP-Hard combinatorial problem type modelled with 0-1 programming, and successfully solved the well-known benchmarks outperforming state-of-art algorithms.

Keywords: Artificial Bee Colony, 0-1 Programming, Adaptive Operator Selection, Uncapacitated facility location problems

1. Introduction

Optimisation remains one of the prominent engineering problems, which influences many other fields in engineering research including real-world problem solving. It is well known that optimisation requires never-ending attention from researchers due to its crucial role in handling many other

*Corresponding author
Email address: rafetdurgut@karabuk.edu.tr (Rafet Durgut)

engineering problems. This includes efforts for both functional and combinatorial optimisation. A large literature on engineering optimisation is produced on research results, which drives attentions on a trade-off between computational complexity and the quality of solutions that put decision-makers in a dilemma in this regard. "Metaheuristics" is known as a mature sub-field of research in optimisation, which offers a number of search and problem solving frameworks to implement for each problem type. The literature is full of many successful metaheuristic implementations [1] across the whole horizon of problem solving field including transport [2], finance and economy [3], health-care systems [4], and many engineering problems [5].

Metaheuristic approaches, which are developed inspiring of natural processes and intelligence, can be viewed in two categories with respect to the number of solutions considered while operating; (i) population-based, e.g. evolutionary computation and swarm intelligence, and (ii) individual-based approaches, e.g. simulated annealing, tabu search etc. Particularly, swarm intelligence algorithms recently become very popular due to their flexibility in implementation and ability to model collective intelligence and behaviours, which can become very efficient in search and problem solving. The most renown swarm intelligence algorithms are ant colony optimisation [6], particle swarm optimisation [7], and artificial bee colony (ABC) [8] algorithms. Metaheuristics and swarm intelligence approaches are very adaptive in implementation for various discrete and combinatorial problems with minor revisions across domains minimising the reality of no-free lunch theorem [9]. More useful details can be captured from these articles; [10, 11, 12].

ABC algorithms are relatively recently developed swarm intelligence algorithms inspired by natural collective behaviours of bee colonies in searching for food sources. They have been implemented for solving a number of real-world problems alongside theoretical performance benchmarking problems. Success has been proven for a number of functional optimisation in continuous space by various works [8]. Due to performance and efficiency issues, hybrid approaches have also been studied for combinatorial and functional domains [13]. A number of studies attempted for solving combinatorial optimisation problems across manufacturing to economy and finance with ABC variants [14] [15] [16]. It is well-known that many combinatorial problems have been mathematically modelled in binary form for higher efficiency purposes. Binary approach, a.k.a. 0-1 programming in operations research, has been used for solving a number of combinatorial optimisation problems such as quadratic assignment [14], knapsack [17] and set-covering [18] problems.

Similar to other metaheuristic and swarm intelligence approaches, ABC algorithms use neigh-

bourhood functions, which are also called operators, to move from one state of the problem to another, e.g. the next state. However, the performance of the ABC variant remains very much dependent on the boundaries and limitations of neighbourhood functions. Hybridisation has mainly been introduced to avoid the negative impacts of these boundaries (limitations). A typical hybridisation is to use multiple neighbourhood functions, algorithms[19, 20], or operators subject to an interchanging scheme to minimise the impact of the such limitations, which impose a selection rule to opt one operator from the set of existing ones. A random selection scheme is introduced by [13] and [21] for these purposes. Few operator selection schemes based on multi-armed bandit approaches have been studied for the purpose of opting genetic operators in genetic programming domain [22]. These prioritisation schemes can be named as probability matching (PM), adaptive pursuit (AP) or upper confidence bound (UCB) methods [23]. These schemes work based on credit assignment procedures, which are awarding processes for successful operators to make them more prominent/preferable over the alternatives in the prospective cases. The success of an operator can be identified with either improvement in the fitness values or a binary process represents the state of success; success/fail (1/0) [24]. To the best of the authors' knowledge, there is not any study attempted a hybrid ABC algorithm to solve binary problems use adaptive approaches.

The main idea of this paper is to create an adaptive scheme for organising multiple neighbourhood functions to work adaptively within the framework of ABC algorithm. This scheme aims to help create an adaptive and hybrid ABC algorithm to solve binary problems. It is important to note that the keyword of "neighbourhood function" is used interchangeable with "operator" in the rest of this paper. The adaptive scheme, called adaptive operator selection, devises an approach to arrange when and how each alternative operator is to take up the role of producing new solution within the stages/phases of ABC algorithm [25]. Interchangeably use of different operators is expected to fertilise the search process as each operator uses different breeding rule, which can rescue search from local optima whenever it sticks in. It is not trivial to arrange rescue operators to offer a way out of local optima driven by other operators, where there should be a measure based on capabilities for each operator to indicate if it offers better action for given search circumstances. The measure for capabilities can be designed based on success and failure history of each operator. This requires studying credit assignment and update process using the success and failure data [26, 27] in order to make operator selection process smart and the main algorithm efficient.

In this paper, an adaptive operator selection procedure is investigated, and finally suggested, to

ensemble in ABC algorithm for solving one of combinatorial optimisation problems, uncapacitated facility location problem, which is modelled with binary programming. The performance is studied with well-known benchmark problems from OR Library [28].

2. Related Works

Binary ABC has been implemented for solving binary, 0-1 programming, problems. The studies carried out for solving binary problems with ABC variants propose use of different schemes to improve the performance and efficiency on various levels. The success of all variants of ABC is bounded with the limitations of ensembled neighbourhood structures and operators similar to all other metaheuristics. One way to break through the corresponding limitations to enhance the performance further is either to design new and more capable functions, which will also impose its own limitations, or to exploit multiple neighbourhood functions to work in a bespoke order [29], which is aimed in this study. This may also be called as a hybrid adaptive approach since it brings different distinctive procedures together to work in collaboration.

There are not many studies have been undertaken to solve binary problems using ABC. Table 1 tabulates the most relevant ABC studies for solving binary problems indicating strength and weaknesses. Few approaches such as [30], [31] and [15] are developed to adapt original ABC, which works in continuous domain, for solving binary problems using converter functions, while more works propose binary versions of ABC for solving binary problems more efficiently [32] [33] [34] [35]. The majority of the studies included in Table 1 use either single binary operators or original ABC scheme with a converting functions. Few of them such as [36], [37] and [38] exploit multiple operators with either non-adaptive selection schemes or sole adaptive schemes. These leaves a substantial deficit in research that more adaptive approaches are to be thoroughly investigated with respect to efficiency in prioritisation and credit assignment to contributing operators in prioritisation process.

It can clearly be observed from the right-most column of Table 1, titled "*Area to improve*", that binary ABC algorithms have not been comprehensively studied to reveal if adaptive selection schemes would help improve efficiency and performance of binary ABC algorithms in problem solving adaptively selecting operators from the set of existing alternative operators. In this paper, a comprehensive study has been conducted to clarify if the adaptive selection schemes would help improve the algorithmic performance and which of the schemes would assist to perform better. The winning scheme has been identified via an extensive experimentation and suggested ultimately.

Table 1: Summary of related works

Article	Year	Domain	Contribution	Area to improve
[30]	2005	Continues	The first ABC algorithm; works in continues domain only.	Requires additional operations for binary or discrete domains
[32]	2012	Binary	Uses another optimisation model to convert the problem to binary space; works for rather smaller size problems.	slower approximation for larger problems, may stuck in local minima.
[33]	2013	Binary	Uses logic gates to convert the problem to binary space.	Does not use neighbourhood information and approximates slowly
[36]	2014	Continuous	Uses multi-operator ABC using 3 different operators selecting operators either randomly or based on success count.	Credit assignment process has not been considered.
[39]	2015	Binary	Uses additional operations to convert continues problem to binary space	the performance in continues domain and the characteristics of conversion operators bind its success in binary space. approximation is slower.
[31]	2015	Continuous	An adaptive approach is proposed, based on success counter and roulette wheel processes, to work with 5 operators in the continuous domain.	Alternative selection operators and credit assignment procedures are not considered.
[15, 40]	2015	Binary	Genetic operators are used to covert to binary space	Local minima is very likely
[37]	2015	Continuous	Uses 3 operators to generate 3 alternative solutions per update cycle.	Credit assignment or success-based approach is not considered. Approximates slowly
[41]	2018	Binary	Uses comparative approaches in conversion to binary space.	Performance is bound by it success in continuous domain. Approximates slowly.
[38]	2019	Continues	Uses Probability Matching to select operators adaptively among 5 operators used with ABC and developed to solve continuous problems	Neither alternative detailed analysis nor other adaptive mechanisms considered.
[35]	2020	Binary	Extends the idea suggested in [33] with considering multiple revisions in the solutions	Single operator is used. The performance is bound by the characteristics of the operator

3. Artificial Bee Colony

The artificial bee colony (ABC) algorithm is one of recently developed swarm intelligence approaches inspired by food search behaviors of the honey bee swarms. The original algorithm has been developed by Karaboğa [30], which imitates the collective behaviour of honey bees within their hives. The algorithm implies use of two types of bees within the hive; employed and onlooker bees. These social insects fulfil collective behaviour in three different phases as modeled into this approach, where first phase imposes each employed bee to improve its own food source, while the second phase involves each onlooker bee to look for improving the quality of its own food source. In the final phase, an exploration is initiated for new food sources by onlooker bees, subsequently transformed into scout bees, if non-adequate improvement is achieved. Further investigations and enhancements for functional optimisation problems are reported in [13].

The conceptualisation of the ABC algorithm translates the natural processes and activities

into algorithmic components and functionalities, where "food source" is translated into a "feasible solution" denoted with $\mathbf{x}_i$, while "nectar amount" is recognised as the fitness of a solution denoted by $F(\mathbf{x}_i)$ as given in Eq. 1.

$$F(\mathbf{x}_i) = \begin{cases} \frac{1}{1+f(\mathbf{x}_i)} & f(\mathbf{x}_i) \geq 0 \\ 1 + |f(\mathbf{x}_i)| & \text{otherwise} \end{cases} \tag{1}$$

The probability of a particular food source to be selected through the process of ABC algorithm is calculated with Eq. 2, while a neighbouring solution such as $\mathbf{x}_n = \mathbf{x}_i + \mathbf{v}_i$ generated using Eq. 3

$$p(\mathbf{x}_i) = \frac{F(\mathbf{x}_i)}{\sum_{j=1}^{N} F(\mathbf{x}_j)} \tag{2}$$

$$\mathbf{v}_i = \mathbf{x}_i + \phi_i(\mathbf{x}_i - \mathbf{x}_n) \tag{3}$$

where $\mathbf{x}_i, \mathbf{x}_n, \mathbf{v}_i$ in the equations refer to the current solution, neighbor solution and candidate solution, respectively. ϕ_i is a randomly generated number in the scale of $[-1, 1]$. $i = 1, 2 \ldots, N$ indicates the index of the food source, where N indicates the number of food sources. On the other hand, the scout bees can be generated using Eq. 4 when no improvement is realised by onlooker bees.

$$x_{i,j} = LB_j + rand(0, 1) \times (UB_j - LB_j) \tag{4}$$

where, $x_{i,j}$ is the j^{th} decision variable as the member of $\mathbf{x}_i$ solution vector; $j = 1, 2, .., D$ is the index, D is the total number of decision variables, LB and UB are the upper and lower boundary values defined for the decision variable.

3.1. Binary Artificial Bee Colony

Swarm intelligence algorithms have been originally developed for either discrete or continuous problems, and then implemented for the problems from other type; ant colony optimisation (ACO) was originally developed for combinatorial problems, e.g. traveller salesman problem, while particle swarm optimisation (PSO) and artificial bee colony (ABC) algorithms were developed for continuous problems. In order to implement a search algorithm for a problem type that was not originally

developed for, additional conversion procedures are embedded into the algorithms, which applies to ABC, too. Two approaches are mainly devised for implementing ABC to solve combinatorial problems. The commonly used approach is to keep the continuous variables in use with an operation to map the variable in use to a binary vector and vice versa. Here, $\psi : [-a, a]^D \rightarrow [0, 1]^D$ is employed for mapping [41], where ψ, a and D are the mapping function, the boundary of the range of the variable, and the dimension of the the variable vector, respectively. This conversion strategy looks very inline with the idea of phenotype-genotype concepts in evolutionary computation. The other approach imposes use of binarification functions such as $z_i = round(|y_i \pmod 2|) \bmod 2$, where z_i and y_i are the i^{th} element of binary vector and real vector, respectively [31].

Eq. 5 is commonly used to initialise the population of solutions by many binary problem solving algorithms.

$$x_{i,d} = \begin{cases} 0, & \text{if } rand < 0.5 \\ 1, & \text{otherwise} \end{cases} \tag{5}$$

Ozturk et al. [40] use genetic operators including crossover and swap operators as neighbourhood functions. Santana et al. [34] proposes cloning $n \geq 1$ dimensions from the selected solution, $x_{i,j}$, and its neighbour solution, $x_{k,j}$, and applying other well-engineered operations to compose a new solution while Kashan et al. [32] suggests use of dissimilarity index in between a neighbour solution and the selected solution as a measure to produce a new solution. On the other hand, logical operators have been used in generating new binary vectors to be new solution by the authors of [42] and [33]. The former work suggests use of $v_{i,j} = x_{i,j} \otimes (\phi_{i,j} \odot (x_{i,j} \oplus x_{k,j}))$ and the latter study uses $v_{i,j} = x_{i,j} \otimes (\varphi_{i,j}(x_{i,j} \otimes x_{k,j}))$ to update each dimension of the new solution, $v_{i,j}$, where $\otimes$, $\odot$ and $\oplus$ respectively represent XOR, AND and OR logical operators, $\phi_{i,j} \in [0, 1]$ valued randomly, $\varphi_{i.j}$ is logical NOT operator that inverts the bit values with 50% probability. Durgut [35] proposes an iterative procedure to run the rule proposed by [33] dynamically for more dimensions than a single one. The estimation of $\varphi_{i,j}$ is pragmatically conducted considering the neighbouring solution.

As explained above, ABC has been proposed for continuous problems, hence, modifications are required to implement it for binary optimisation problems. More details for the following three methods presented so as to use them later.

3.1.1. binABC Algorithm

binABC algorithm has been proposed by Kiran et al. [33] using *binABC* operator with which the new solutions can be generated with Eq. 6 that replaces Eq. 3 in original ABC. *binABC* imposes use of XOR logical operator to generate new binary solutions from a selected solution, $x_{i,j}$, and one of its neighbours, $x_{k,j}$. It is important to note that the variables provided in Eq. 3 are in vector form while are in scalar form in Eq. 6. The parameter of φ is used as logical NOT operator with which the output of the parenthesis is inverted with 50% probability.

$$v_{i,j} = x_{i,j} \otimes \varphi(x_{i,j} \otimes x_{k,j}) \tag{6}$$

XOR operator is applied to current,$x_{i,j}$, and neighbor, $x_{k,j}$, solutions, then the output value is negated if $\varphi_{i,j} < 0.5$, kept as is, otherwise. Afterwards, XOR is re-applied to the current solution, $x_{i,j}$ and the output value filtered out with $\varphi_{i,j}$ for the final output value, $v_{i,j}$.

3.1.2. disABC Algorithm

disABC is proposed by Kashan et al. [32] which calculates a similarity measure by Eq. 7 in which the similarity of the bits in two compared solutions plays the key role. A dissimilarity measure, which names the algorithm, is subsequently determined for the operator to generate the new solution with for neighbour solution generation. The approach imposes use of Eq. 5 to initialise the solutions replacing Eq. 4 as used in original ABC, then to calculate Jaccard's similarity constant with Eq. 8 between a chosen solution and one of its neighbour solutions for evolutionary iterations.

$$sim(\mathbf{x}_i, \mathbf{x}_k) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}} \tag{7}$$

$$dissim(\mathbf{x}_i, \mathbf{x}_k) = 1 - sim(\mathbf{x}_i, \mathbf{x}_k) \tag{8}$$

where M_{11} is the number of 1 bits in both $\mathbf{x}_i$ and $\mathbf{x}_k$ at the same positions, while M_{01} and M_{10} are determined, accordingly. Eq. 9 declares the dissimilarity of the selected solution with the solution to be generated, v_i is an approximate dissimilarity between the selected solution, $\mathbf{x}_i$ and one of its neighbour solutions, $\mathbf{x}_k$, normalised with ϕ, which is optimally calculated using the minimisation model presented in Eq. 10. Once the model solved to optimum the new solution to-be, $\mathbf{v}_i$, is

generated with a new update equation, $\mathbf{v}_i = \mathbf{x}_i + \phi \times dissim(\mathbf{x}_i, \mathbf{x}_k)$, replacing Eq. 3.

$$dissim(\mathbf{v}_i, \mathbf{x}_i) \approx \phi \times dissim(\mathbf{x}_i, \mathbf{x}_k) \tag{9}$$

$$\min \ |dissim(\mathbf{v}_i, \mathbf{x}_i) - \phi \times dissim(\mathbf{x}_i, \mathbf{x}_k)| \tag{10}$$

Subject to:

$$M_{11} + M_{01} = n_1$$
$$M_{10} \leq n_0$$
$$\{M_{10}, M_{11}, M_{01}\} \geq 0 \text{ and } \in \mathbb{Z}$$

where ϕ is a random positive value, n_1 and n_0 represent the number of bits with a value of 1 and 0 in the current solution, $\mathbf{x}_i$. The aim in here is to solve the model to identify the best ϕ value to normalise $dissim(\mathbf{x}_i, \mathbf{x}_k)$ measure. Detailed information and examples can be found in [32].

3.1.3. Improved binABC (ibinABC)

This algorithm is an improved version of *binABC*, which originally updates a single dimension, one among D, of decision variable per operation, while various other swarm intelligence algorithms propose updating multiple variables within the complete vector of decision variables. Obviously, there is a trade-off between exploration and exploitation balance to handle while attempting the updates.

ibinABC attempts to balance exploration and exploitation with an exponentially calculated rate, d_t as in Eq.11 inspired by the idea of updating multiple dimensions per operation in [34]. d_t number of dimensions will be updated at iteration t.

$$d_t = rand(0, \alpha) + e^{-(\frac{t}{t_{max}}) \times 0.1 \times D} + 1 \tag{11}$$

where, the α is randomly determined perturbation number, D is the problem dimension, number of decision variables, and t and t_{max} indicate the current and maximum number of iterations, respectively. We note that change in multiple dimensions increases the exploration but reduces exploitation. In order to balance that d_t is set to decrease with growing t so that the exploration becomes higher in earlier iterations while exploitation gets stronger in later iterations.

On the other hand, *ibinABC* imposes a dynamic normalisation factor, φ, in contrary to the one proposed by Kiran et al. [33] as in Eq.6. The operator in Eq.6 is revised in the way that φ dynamically takes values for each iteration instead of the original setup, $\varphi = 0.5$. In fact, this pre-fixed threshold potentially weakens the exploitation, especially in later stages, as it involves more random process. Eq.12 proposes a new way to determine φ. This rule allocates 0 to φ if the new solution is worse, a calculated value in the range of [0,1] otherwise, as updated depending on the iteration, t.

$$\varphi = \begin{cases} \varphi_{max} - (\frac{\varphi_{max} - \varphi_{min}}{t_{max}}) \times t & F(x_k) < F(x_i) \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

where φ_{max} and φ_{min} represent the upper and lower limits of the defined range, respectively [35].

3.2. Adaptive Operator Selection

ABC algorithm has been implemented in various variants for wide rage of applications. The majority of the implementations have been developed with either single neighbourhood function or hybrid with some kind of local search. Dugenci and Aydin [21, 13] have introduced a hybrid form to randomly select operators among alternatives. However, it does not offer an adaptive and merit-based scheme. Adaptive operator selection is a merit-based scheme that imposes to exploit different operators interchangeably whenever applies whilst the search ongoing. This has been successfully applied to individual-based search approaches, particularly systematically implemented in variable neighbourhood search algorithms [43]. This has recently been implemented for population-based metaheuristics, too [44]. Wu et al. [29] overview the up-to-date literature for population based metaheuristics and ensemble algorithms with this respect. Fialho et al. [45, 46, 27] have experimentally studied the impact of average, extreme and immediate/instant credit/awards upon the performance of genetic algorithms, while Chent et al. [38] have implemented an approach used with ABC for solving continuous problems combining three search strategies borrowed from differential evaluation (DE) studies using probability matching (PM) method.

Wang et al. [36] proposed a framework to orchestrate three ABC variants in which a selected rule is kept re-used as long as it produces success, another rule is randomly pulled up from the preset rule base. Gao et al.[37] have implemented an ABC for continuous problems using three search strategies in which the operators are selected with roulette-wheel that determines the probabilities based on success rates, while Kıran et al. [39] have increased the number of strategies to five

using the best producing one adaptively with their ABC variant. Finally, Xue et al [22] suggest a self-adaptive structure to use among three search strategies, which are benefited of global best solution. To the best knowledge of the authors, any adaptive approach to select operators has not been applied to binary ABC algorithms so far.

4. Proposed Methodology

This section details the proposed methods and corresponding material sources including the set of benchmarking uncapacitated facility location problems. The main focus goes on how the neighbourhood functions, the operators, are integrated into binary ABC in an adaptive way, which requires a selection scheme. The adaptive operator selection (AOS) procedure and credit assignment rule for the selected operators are introduced accordingly. Furthermore, introduction in uncapacitated facility location problem follows, accordingly. Apparently, AOS involves with two interacting components; (i) operator selection - how operators are selected from the pool of operators based on the credit level, (ii) credit assignment - how to assign/update credit to each operator employed.

4.1. Operator Selection Schemes

Operator selection is the process of choosing an operator from the pool of operators using the credit level of each, where the pool of operators is devised and integrated in the main search algorithm. The selection is conducted with a particular rule, which can be bespoke from a very random rule to a complex heuristic procedure. It is paramount to indicate that the main concern in search algorithms is the balance set up between exploration and exploitation, where exploration is to look for a solution rather randomly (blindly), while exploitation is the way to search with the guidance gained/provided within the algorithm. This concern might be called as the dilemma of *exploration versus exploitation* (EvE). The success of an operator selection rule relies significantly on how it addresses the dilemma of EvE. Among heuristic rules, *probability matching* (PM), *adaptive pursuit* (AP) and *multi-armed bandit* based methods [25] are chosen to work with in this study. PM and AP heuristic rules use roulette-wheel mechanism to determine the probabilities. PM determines the probabilities as in Eq. 13.

$$p_{i,t} = p_{min} + (1 - Kp_{min})\frac{q_{i,t}}{\sum_{j=1}^{K} q_{j,t}}, \quad i = 1, 2..K \tag{13}$$

where K is the number of operators in the pool, $p_{min} \in [0, 1]$ represents the minimum probability of being selected, and $q_{i,t}$ is the credit level/value of operation i at time t. Both PM and AP use p_{min} to set a base probability for each operator, which would help address the *dilemma of EvE* with allocating a minimum chance to every operators to be selected. PM imposes to calculate the probabilities of being selected per operation, while AP uses the strategy of "*winner takes all*" approach that credits more to promising options. AP calculates the probabilities with Eq. 14.

$$p_{i,t+1} = \begin{cases} p_{i,t} + \beta(p_{max} - p_{i,t}), & \text{if } i = i_t* \\ p_{i,t} + \beta(p_{min} - p_{i,t}), & \text{otherwise} \end{cases} \tag{14}$$

where $i_t* = \arg\max_{i=1,..,K}\{q_i\}$ and $\beta \in [0, 1]$ denotes learning coefficient, while $p_{i,0} = 1/K$ as the initial probability value. PM is experienced to take longer time to approximate a balanced EvE policy while AP offers the *winner-takes-all* normalised with learning coefficient, β. More discussions can be found in [27].

The operator selection rules based on multi-armed-bandit (MAB) approach are considered as friendly to *EvE dilemma* in which exploitation phase is supported/empowered with opting highly prioritised operators while weaker operators are given opportunity with supporting exploration phase. UCB method is one of most commonly used MAB approach, which determines the operator to select with Eq. 15.

$$op_t = \arg\max_{i=1,..,K}\{q_{i,t} + C \times \sqrt{\frac{2\log\sum_{j=1}^{K} n_{j,t}}{n_{i,t}}}\} \tag{15}$$

where op_t represents the selected operator, C works as a scaling factor, n is number of times the operator selected while $q_{i,t}$ and $n_{i,t}$ on the right-hand-side of equation help control *EvE dilemma*, respectively.

4.2. Credit Assignment Mechanisms

Credit assignment is the process of evaluating the success level of a chosen operation that is exploited once selected. Each selected operator per iteration is evaluated following the delivery of operation with respect to the level of success; if it achieves a full success, partial success or a failure. A reward value is decided and assigned to the selected operator in order to update its credibility for the next runs. The reward, in another word credit, is determined based on either immediate results

(instant), or as the average of a pre-set window of the time, where the average is recalculated after delivery of each operation considering last W number of iterations, i.e. run of an operation. The level of success per run of operations can be estimated based on either (i) the value of objective function, denoted as O.V., or (ii) the success value, denoted as S.V. so forth. In the case of instant reward/credit using objective function, the reward will be as in Eq. 16.

$$r_{i,t} = f(v_i) - f(x_i) \tag{16}$$

where $r_{i,t}$ is the reward estimation for operation i at time t, $f(v_i)$ is the calculated objective value after applying the chosen operator, and $f(x_i)$ is the known objective value of the current solution, x_i. Reward calculation with immediate results. i.e. instant case, can cause degeneration or disruption in the later stages [45]. For instance, it is not so difficult to improve a very rough solution, particularly at very early stages of the search while can be very difficult in the later sages, therefore, the quantity calculated for reward should not be the same. One idea to overcome this issue is to normalise the difference between $f(v_i)$ and $f(x_i)$ with the factor of the rate between attained objective value and the global best value as in Eq. 17.

$$r_{i,t} = \frac{f(v_i)}{gbest_t}(f(v_i) - f(x_i)) \tag{17}$$

The credit assignment is conducted following the estimation of reward, $r_{i,t}$. The type of reward to be used in the credit assignment using the following update function, Eq. 18, can be (i) *instant* reward, (ii) *average* reward, or (iii) *extreme* reward, where both of (ii) and (iii) are estimated within a sliding window of the time, with the size of W, in which the average is considered as the mean and the best reward is considered as extreme. The credit assignment is fulfilled using the update rule given in Eq. 18.

$$q_{i,t+1} = (1 - \alpha)q_{i,t} + \alpha r_{i,t}, \quad i = 1, 2..K \tag{18}$$

where $q_{i,t+1}$ and $q_{i,t}$ are the updated and current credit levels for operation i in time $t + 1$ and t, while K is the number of operators in the pool and $\alpha \in [0, 1]$ is the adaptation factor. Meanwhile, MAB uses the *average* reward as the credit update, which is calculated up-to-date or the average of sliding time window.

4.3. Adaptive Binary Artificial Bee Colony (ABABC)

The binary ABC algorithms furnished with adaptive selection operators mean to be called adaptive binary artificial bee colony (ABABC) algorithms. The main idea of ABABC is sketched in Figure 1, which centers around the candidate generation process supplied by the population of the solutions and the pool of operators, denoted with $\mathcal{O}$. The pool can be defined as a set of binary operators; $\mathcal{O} = \{Op_i | i = 1..|\mathcal{O}|\} = \{binABC, disABC, ibinABC\}$, which allows the selection of an operator, Op_i, from the pool, $\mathcal{O}$, using one of the operator selection schemes explained above, i.e. one of *PM*, *AP* or *UCB*. A new candidate solution, v_j, is generated with an operator selected from the pool, $Op_i \in \mathcal{O}$, based on the information cloned from a selected solution, x_j, and one of its neighbours, x_k. Once the generated solution, v_j, is found better performing than the selected solution, x_j, i.e. $f(v_j)$ is better than $f(x_j)$, a reward, $r_{i,t}$, is estimated and fed into credit assignment rule to update the chosen operator's credit level, $q_{i,t}$.

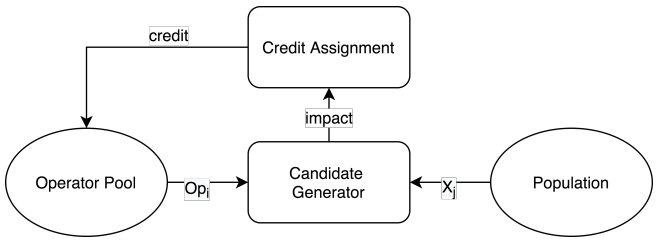


Figure 1: General overview of adaptive operator selection process with support of population and pool of operators

The proposed approach is algorithmised in pseudo code as provided in Algorithm 1. Each solution, i.e. bee, takes part the population, i.e. the complete bee colony, is generated using Eq. 5 and evaluated with Eq. 1. Then, an operator for each bee is selected from the pool, where all operators are equally credited, initially, here at initialisation phase, but, are updated based on their success in due course.

The next phase of the algorithm runs with *employee bees*, following the initialisation, in which each employee bee updates the colony with a particular food source running its own allocated operator to update on the solution with corresponding fitness value calculated accordingly. The generated candidate solution replaces the original one if the new solution is better, it is discarded otherwise. The corresponding counts are updated, accordingly. Then, the *onlooker bees* come to effect, where the probabilities of food sources are estimated, operators are assigned to the bees and then the same procedure as employee bees is executed. This phase creates an alternative

improvement opportunity in parallel to employee bee phase. The update phase follows these two to revise the credit levels and apply *memorisation* for updating the best solution so far, accordingly. The following phase, last of stage of an iteration, is to check the age of non-improving bees and to replace with randomly generated new bees, i.e. *scout bees*, if corresponding non-improving bees are out-aged, to age each bee otherwise. The algorithm keeps iterating until the pre-set stopping criterion is met.

4.4. Uncapacitated Facility Location Problem

UFLP has been studied with metaheuristics including simulated annealing [47, 48, 49], tabu search [50] and genetic algorithms [51, 48]. The benchmarking problem instances released by OR Library [28] are very popular to test the algorithmic performances of metaheuristic approaches. The problem mathematically is defied to be a minimisation problem in which the best facility locations are chosen among many alternatives so that the overall cost including capital expenditure, i.e. setup cost, and operational expenditure, i.e. shipment cost, is minimised subject to the set of constraints.

Let m and n be the number of alternative facility locations and the number of customers, respectively. $\mathbf{c} = \{c_{i,j} | i = 1..m, j = 1..n\}$ denotes shipment cost between facility location, e.g. warehouse, i and delivery point, e.g. customer, j and $\mathbf{f} = \{f_i | i = 1..m\}$ denotes setup cost for facility location i. The problem requires to work out with two binary decision variables; x_i is identify if location i is to take part of the set of facilities to open, and $y_{i,j}$ is to decide if location i it to serve customer j, where x_i will value of 1 if the location is to open, 0 otherwise. Similarly, $y_{i,j}$ is to be 1 if location i serves customer j, 0 otherwise. The details of mathematical model is provided in Eq: 19 and 20. The UFLP model has turned to be a pure 0-1 programming model, while there exists another version of UFLP, which implements a mixed integer linear programming model considering shipment decision variable, $y_{i,j}$ here, as a non-binary integer variable, as in references of [47],[48], [49] and [51].

$$\min f = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} y_{i,j} + \sum_{i=1}^{n} f_i x_i \tag{19}$$

Algorithm 1 The pseudo code of the adaptive binary artificial bee colony

Initialization phase:
Set algorithm parameters
Create initial population
Evaluate population
while Termination criteria is not met **do**
    Employed bee phase:
    Select operators and assign to bees
    Increment operator counter
    for i=1 to N **do**
        Select neighbor, apply operator and obtain candidate solution (v_i)
        if $f(v_i)$ is better than $f(x_i)$ **then**
            Replace v_i with x_i
            Get reward and add to $r_{op,t}$
            Increment reward counter
            Reset trial counter
        else
            Increment trial counter
        end if
    end for
    Onlooker bee phase:
    Calculate probabilities for food sources
    Select operators and assign to bees
    Increment operator counter, t=0
    for i=1 to N **do**
        Determine current solution according to probability
        Select neighbor food source
        Apply operator and obtain candidate solution (v_i)
        if $f(v_c)$ is better than $f(x_c)$ **then**
            Replace v_c with x_c
            Get reward and add to $r_{op,t}$
            Increment reward counter
            Reset trial counter
        else
            Increment trial counter
        end if
    end for
    Update Phase:
    Credit assignment
    Memorization
    Scout bee phase:
    if Limit is exceed for any bee **then**
        Create random solution for the first exceeding bee and evaluate it
    end if
end while

Subject to:

$$\begin{aligned} &\sum_{i=1}^{n} y_{i,j} = 1, \quad j = 1, 2, ..., m \\ &y_{i,j} \geq x_i, \quad i = 1, 2, ..., n \quad j = 1, 2, ..., m \\ &y_{i,j}, x_i \in \{0, 1\} \end{aligned} \tag{20}$$

ABABC has been implemented to solve UFLP in a very straightforward way since the problem is modelled as a 0-1 programming problem and ABABC is a binary problem solving algorithm. Each solution, i.e. bee, is created in a pair of binary elements; $\mathbf{x}_{1\times n}$ is a vector of n binary values to represent the facility locations if each is open or not and $\mathbf{y}_{m\times n}$ is a matrix of binary values to identify the relationship between customers and open facilities. Here, $\mathbf{x}_{1\times n}$ has to include $\bar{n} \leq n$ number of 1 bits to keep facilities open while each row of $\mathbf{y}_{m\times n}$ has to include only one 1 bit per open facility for satisfying the constraints in the model. The objective function given by Eq.19 is used to calculate the fitness values, $F(\mathbf{x}, \mathbf{y})$, using Eq.1.

The performance of the variants of the algorithms considered in this study has been evaluated using the commonly used benchmark problems taken from OR-Library [28] as tabulated in Table 2.

Table 2: OR-Library UFLP dataset

Problem name	Problem size	Optimal cost value
Cap71	16, 50	932615.75
Cap72	16, 50	977799.40
Cap73	16, 50	1010641.45
Cap74	16, 50	1034976.98
Cap101	25, 50	796648.44
Cap102	25, 50	854704.20
Cap103	25, 50	893782.11
Cap104	25, 50	928941.75
Cap131	50, 50	793439.56
Cap132	50, 50	851495.33
Cap133	50, 50	893076.71
Cap134	50, 50	928941.75
CapA	100, 1000	17156454.48
CapB	100, 1000	12979071.58
CapC	100, 1000	11505594.33

5. Experimental Results

UFLP has been chosen for testing the performance of ABABC, which is the algorithm proposed in this paper. Parameter setting and fine-tuning is to be reported first, and then the produced

results are evaluated in comparison with the state-of-art works. The proposed approaches are implemented and developed using C programming language to run each experiment applying $8e^5$ number of operations in order for a fair comparison.

5.1. Parameter Tuning

Table 3 presents ABABC results for CapC with a configuration of instant reward, PM and AP selection approaches. Given these circumstances, the best bespoke values are are investigated for N, p_{min}, and α parameters, where the best results are obtained with $p_{min} = 0.2$ that increases the randomness level, which pushes PM and AP rules towards lower performance.

Table 4 presents the results produced with a reward calculation based on objective function and using the same parametric configurations adopted for the experimental results tabulated in Table 3. The new results seem clearly improved, particularly, with 0.05 and 0.1 values of p_{min} that help improve, significantly. This means that the selection process allocates lower rate to exploration and higher rate to exploitation so as to smartly switch in between whenever needed. Withstanding these circumstances, the best results are observed with parametric setting of $N = 80$, $\alpha = 0.9$ and $p_{min} = 0.1$ for PM, and $N = 40$, $\alpha = 0.7$ and $p_{min} = 0.05$ for AP. The rest of experimentation will be conducted under the light-shed of these parametric settings that found best productive.

It is observed in Table 3 and 4 that better results for PM and AP obtained with objective function-based reward calculations, herewith O.V. to denote so forth. The best parametric setting fine-tuned as $N = 40$, $\alpha = 0.7$ and 0.9, $p_{min} = 0.05$ and 0.1. The next set of experimentation investigate the best window size, W from the alternative value set of $\{5, 10, 20, 50\}$, for sliding window approach to produce *average* and *extreme* rewards. All the experiments have been conducted using CapC benchmark as indicated before. The experimental results for window size are tabulated in Table 5, where the best results are obtained with $W = 50$ and *average* reward approach for PM, and $W = 50$ and *extreme* reward approach for AP. It is also observed that the sliding window approach works better for PM and AP selection methods.

Another prioritisation/selection approach is known as UCB, as a multi-bandit armed approach, which requires a parametric setting study for C via testing through CapC benchmark. The reward type chosen for these experiments is not *average* or *extreme*, but *instant* reward. The results are tabulated in Table 6, where the performance results with lower C values are not good with success rate-based reward estimation, herewith S.V. denotes so forth, but get better with growing values

Table 3: Parametric exploration for N, α and p_{min} with success rate reward function

Parameters			Probability Matching			Adaptive Pursuit		
N	α	p_{min}	*Mean*	*Std.Dev*	*Hit*	*Mean*	*Std.Dev*	*Hit*
20	0.3	0.05	11563334.96	43042.61	0	11539816.34	29538.38	0
		0.1	11516440.94	15791.23	8	11515503.20	13560.41	9
		0.2	11508039.92	2748.16	13	11511834.49	8698.81	8
	0.5	0.05	11548768.93	31881.39	2	11540272.29	29191.67	1
		0.1	11517703.10	14500.48	4	11516888.26	13344.06	5
		0.2	11511247.64	7877.58	12	11509561.29	7396.88	14
	0.7	0.05	11564366.42	40831.55	1	11532992.27	25800.14	3
		0.1	11516321.28	14242.15	10	11514113.24	12789.67	8
		0.2	11510440.33	7079.94	7	11508592.28	5373.11	13
	0.9	0.05	11549317.87	41532.51	0	11527449.16	22089.79	3
		0.1	11516378.08	11139.72	4	11513261.58	11052.49	9
		0.2	11508625.41	4394.45	14	11509131.50	6462.54	15
40	0.3	0.05	11599617.86	60970.06	0	11564559.39	39911.57	0
		0.1	11540460.14	31035.28	2	11533374.36	29771.53	1
		0.2	11513135.80	9822.15	5	11511703.93	9046.67	10
	0.5	0.05	11580583.92	42876.96	0	11556058.60	38951.08	0
		0.1	11537540.81	19361.95	2	11519918.75	14872.14	5
		0.2	11514226.67	10082.87	5	11513673.74	11054.35	6
	0.7	0.05	11594740.41	63220.02	1	11555309.88	41856.81	1
		0.1	11541724.52	27172.95	1	11524611.84	21670.23	4
		0.2	11510031.92	5126.22	6	11513671.55	10247.55	7
	0.9	0.05	11591382.36	65415.94	0	11537780.05	33631.49	6
		0.1	11537540.49	30949.45	0	11525922.70	15165.57	0
		0.2	11512992.70	7813.90	5	11510149.51	6927.57	9
80	0.3	0.05	11626170.36	62320.24	0	11589859.20	51211.10	0
		0.1	11558109.06	34198.76	0	11541427.82	34023.35	0
		0.2	11530108.86	16013.07	2	11529462.85	17072.21	1
	0.5	0.05	11604198.02	49800.73	0	11586844.34	46164.99	0
		0.1	11561410.95	34675.64	0	11549366.36	34967.73	1
		0.2	11528999.51	14250.57	0	11536128.84	24507.11	3
	0.7	0.05	11595266.19	59125.39	0	11553105.36	52644.25	7
		0.1	11561325.80	39041.08	0	11545000.57	27891.08	0
		0.2	11522240.74	14103.06	2	11529383.14	18270.58	0
	0.9	0.05	11605819.39	49001.41	0	11551884.84	50713.27	10
		0.1	11564947.39	45338.69	0	11540146.27	26467.60	1
		0.2	11527534.12	18367.37	1	11526508.05	23730.50	1

Table 4: Parametric exploration for N, α and p_{min} with objective function-based reward function

Parameters			Probability Matching			Adaptive Pursuit		
N	α	p_{min}	*Mean*	*Std.Dev*	*Hit*	*Mean*	*Std.Dev*	*Hit*
20	0.3	0.05	11507101.26	1877.16	18	11507352.42	1911.60	16
		0.1	11507032.30	3916.43	23	11507785.76	3930.13	17
		0.2	11508903.05	5680.00	13	11506975.68	1846.49	19
	0.5	0.05	11507660.19	3938.22	18	11508900.05	6319.50	15
		0.1	11507534.61	3942.15	19	11507352.42	1911.60	16
		0.2	11507226.84	1898.76	17	11508341.12	5420.71	15
	0.7	0.05	11507101.26	1877.16	18	11508341.12	5420.71	15
		0.1	11506850.11	1806.30	20	11507477.99	1915.87	15
		0.2	11506975.68	1846.49	19	11507101.26	1877.16	18
	0.9	0.05	11506724.53	1755.92	21	11506975.68	1846.49	19
		0.1	11507713.23	5486.12	20	11506724.53	1755.92	21
		0.2	11507101.26	1877.16	18	11507226.84	1898.76	17
40	0.3	0.05	11506598.95	1694.46	22	11506975.68	1846.49	19
		0.1	11507101.26	1877.16	18	11507226.84	1898.76	17
		0.2	11507477.99	1915.87	15	11507226.84	1898.76	17
	0.5	0.05	11506473.37	1620.64	23	11506724.53	1755.92	21
		0.1	11506724.53	1755.92	21	11507352.42	1911.60	16
		0.2	11507035.30	2770.62	21	11507226.84	1898.76	17
	0.7	0.05	11506598.95	1694.46	22	11506347.79	1532.69	24
		0.1	11506724.53	1755.92	21	11506975.68	1846.49	19
		0.2	11506724.53	1755.92	21	11506850.11	1806.30	20
	0.9	0.05	11506598.95	1694.46	22	11506981.14	4281.62	24
		0.1	11506724.53	1755.92	21	11506850.11	1806.30	20
		0.2	11507332.72	2492.66	18	11507226.84	1898.76	17
80	0.3	0.05	11506598.95	1694.46	22	11507352.42	1911.60	16
		0.1	11507477.99	1915.87	15	11507785.76	3930.13	17
		0.2	11507854.73	1877.16	12	11508918.97	2393.96	6
	0.5	0.05	11506473.37	1620.64	23	11507609.03	4309.28	19
		0.1	11506850.11	1806.30	20	11508193.49	4670.88	15
		0.2	11509462.00	6862.49	12	11511277.22	7985.92	9
	0.7	0.05	11507766.07	4245.31	19	11506850.11	1806.30	20
		0.1	11507477.99	1915.87	15	11508416.66	2639.62	10
		0.2	11508613.65	4153.73	11	11507603.57	1911.60	14
	0.9	0.05	11506850.11	1806.30	20	11507101.26	1877.16	18
		0.1	11506222.22	1428.00	25	11506850.11	1806.30	20
		0.2	11508165.50	2718.47	12	11509959.39	6423.93	9

Table 5: Parametric exploration for α, p_{min}, Credit Type and W objective function-based reward function

Parameters				Probability Matching			Adaptive Pursuit		
α	p_{min}	W	Credit Type	*Mean*	*Std.Dev*	*Hit*	*Mean*	*Std.Dev*	*Hit*
0.7	0.05	5	Average	11506975.68	1846.49	19	11506598.95	1694.46	22
0.7	0.05	5	Extreme	11506473.37	1620.64	23	11506850.11	1806.30	20
0.7	0.05	10	Average	11506473.37	1620.64	23	11507583.87	2494.71	16
0.7	0.05	10	Extreme	11506975.68	1846.49	19	11506850.11	1806.30	20
0.7	0.05	20	Average	11506473.37	1620.64	23	11506975.68	1846.49	19
0.7	0.05	20	Extreme	11506473.37	1620.64	23	11506222.22	1428.00	25
0.7	0.05	50	Average	11506347.79	1532.69	24	11506704.83	2369.98	23
0.7	0.05	50	Extreme	11506724.53	1755.92	21	11506473.37	1620.64	23
0.9	0.1	5	Average	11506850.11	1806.30	20	11506598.95	1694.46	22
0.9	0.1	5	Extreme	11506598.95	1694.46	22	11506975.68	1846.49	19
0.9	0.1	10	Average	11506724.53	1755.92	21	11507101.26	1877.16	18
0.9	0.1	10	Extreme	11506598.95	1694.46	22	11506347.79	1532.69	24
0.9	0.1	20	Average	11506347.79	1532.69	24	11506473.37	1620.64	23
0.9	0.1	20	Extreme	11506724.53	1755.92	21	11506724.53	1755.92	21
0.9	0.1	50	Average	11506598.95	1694.46	22	11506347.79	1532.69	24
0.9	0.1	50	Extreme	11506473.37	1620.64	23	11506096.64	1302.54	26

Table 6: Experimental results to fine tune C value

		N=20			*N=40*			*N=80*		
C	*Reward*	*Mean*	*Std.Dev*	*Hit*	*Mean*	*Std.Dev*	*Hit*	*Mean*	*Std.Dev*	*Hit*
1	O.V.	11508036.92	3901.37	15	11519031.55	25267.30	5	11543820.67	32453.10	1
	S.V.	11755508.65	110385.59	0	11727526.10	90952.56	0	11687905.10	78007.98	0
5	O.V.	11509451.62	5356.29	8	11516964.65	13163.72	8	11545538.87	33066.04	0
	S.V.	11649598.60	86777.80	0	11626647.30	74737.50	0	11628728.41	76735.98	0
10	O.V.	11508111.34	4262.94	15	11518637.12	19395.98	7	11532917.15	28054.63	1
	S.V.	11527454.20	23421.45	1	11548081.70	38257.48	0	11568191.98	48084.45	0
50	O.V.	11508291.08	2682.38	11	11514943.97	11606.26	6	11546459.68	36176.19	1
	S.V.	11507207.14	2481.79	19	11509782.90	7463.99	11	11518209.76	13918.82	2
100	O.V.	11510248.95	8250.18	9	11515405.37	9881.23	3	11544436.99	30132.55	1
	S.V.	11508827.52	7399.66	18	11508377.26	3488.81	14	11517062.91	15906.45	2
500	O.V.	11510630.99	8815.61	14	11514309.60	10452.94	5	11533447.50	25786.90	4
	S.V.	11508466.70	5398.47	14	11508823.74	5553.91	13	11511801.59	8196.15	6

Table 7: Window size, W, fine-tuning with *average* and *extreme* rewarding approaches using UCB

N	W	Reward Base	Average Reward			Extreme Reward		
20	5	O.V.	11507603.6	1911.61	14	11507101.3	1877.16	18
		S.V.	11509402.4	6180.39	11	11507478.0	1915.87	15
	10	O.V.	11507974.0	3439.93	16	11508215.5	5439.86	16
		S.V.	11507101.3	1877.16	18	11506724.5	1755.92	21
	20	O.V.	11508142.8	4201.35	16	11507478.0	1915.87	15
		S.V.	11507911.3	3917.90	16	11506724.5	1755.92	21
	50	O.V.	11508717.9	5344.58	12	11507729.1	1898.76	13
		S.V.	11507894.6	4464.75	17	11507815.3	2946.54	16
40	5	O.V.	11508878.0	5673.84	14	11507352.4	1911.61	16
		S.V.	11508990.4	4027.59	8	11507980.3	1846.49	11
	10	O.V.	11507709.5	2485.91	15	11510070.9	8569.16	17
		S.V.	11508953.1	7374.82	17	11510882.1	8718.45	12
	20	O.V.	11510804.9	9907.43	16	11506724.5	1755.92	21
		S.V.	11508039.9	2748.16	13	11507854.7	1877.16	12
	50	O.V.	11507664.0	5889.56	21	11508483.9	9583.01	20
		S.V.	11507942.3	4703.58	17	11507226.8	1898.76	17
80	5	O.V.	11508268.4	4178.79	15	11512719.4	11470.36	10
		S.V.	11512874.4	10390.86	4	11515062.1	12602.81	7
	10	O.V.	11510465.7	10109.25	14	11510437.8	19495.87	19
		S.V.	11514648.8	11936.96	5	11511488.0	7340.76	5
	20	O.V.	11506473.4	1620.64	23	11516425.4	26900.04	17
		S.V.	11512716.0	9313.44	7	11517605.8	12654.65	7
	50	O.V.	11506347.8	1532.69	24	11521711.1	33922.83	18
		S.V.	11518448.7	12364.40	1	11511859.4	8516.39	9

of C. It is observed that the best result obtained with S.V is $C = 500$, while is $C = 10$ with O.V. The best value for N is observed as 20 in these circumstances. The rest of experimentation will consider the best parameter set found so far.

Table 7 summarises the experimental results for UCB approach with *average* and *extreme* reward approaches using sliding window, where the window size, W, is looked for the best choice among the options of $\{5, 10, 20, 50\}$. The best choices are observed as $W = 50$ and $N = 20$ for both *average* and *extreme* rewarding cases, while the performance declines with growing population size.

The experimental results so far (Table 3 - 7) help derive the best configuration for parametric settings as in Table 8, which have been applied to all benchmark functions taken from OR-library

Table 8: Best Configurations

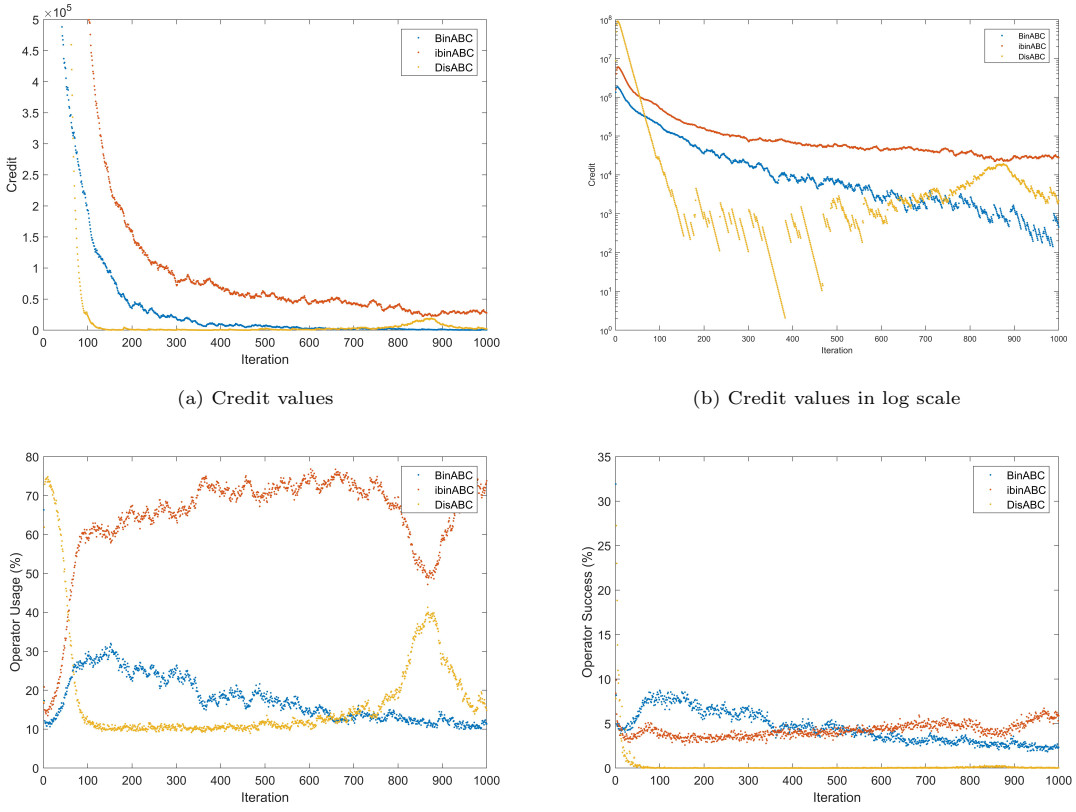
Parameters	PM	AP	UCB
Reward	O.V.	O.V.	O.V.
α	0.9	0.9	-
p_{min}	0.1	0.1	-
N	80	40	80
C	-	-	50
Credit Type	Instant	Extreme	Average
W	-	50	50

as tabulated in Table 9.

Figure 2 shows the plots of the results produced with adaptive binary ABC, averaged over 30 repetitions, for CapC benchmark problem using the best configuration identified for PM with *Instant* reward adaptively selecting one of three operators; *disABC*, *binABC* and *ibinABC*. The figures included plot different aspects; (a) plots the credit levels against the number of iterations in normal scale and (b) in log scale, while (c) presents operator usage over the iteration numbers in % and (d) plots the success of chosen operators in %. It can be observed that *disABC* produces much better results in first 100 iteration, but, then consumes its gained credit very quickly for the remaining stages until iteration 800, then starts raising credit and turns to be competitive once again, competing with *ibinABC*. This can be seen clearly in Figure 2 (a), (b) and (c). On the other hand, *binABC* performs moderately, keeps being competitive, but, the performance declines towards the end. This can be viewed from Figure 2(d) that the performance rises in the earlier iterations, but starts gradually declining after 200 iterations. This might be due to that *binABC* produces less improvable solutions, which may reduce the diversity within the operating neighbourhood. Figure 2(d) also tells us that *binABC* gains high credit while its success rate is not proportionally high. The graphs in all 4 plots in Figure 2(d) suggest that PM helps select the operations on merit base driving through quality of solutions.

The dynamic behaviour of operators selected with AP approach is experimented with CapC benchmark. The best found configuration, as reported in Table 8, has been considered and the experiments have been repeated 30 times. The results are plotted in Figure 3 to reveal the behaviours (a) in normal scale credit, (b) log-scale credit, (c) operators' usage, and (d) the success of selected operators. AP approach is found to serve better with *extreme* rewarding as suggested in previous experimentation. *disABC* is observed to produce better in earlier until 200 iterations, then it under-performs until 1400 iterations, while *binABC* competes with *ibinABC* until 400 iterations, then lessens effectivity. As suggested by all plots in Figure 3, *disABC* and *ibinABC* perform

Figure 2: Dynamic behaviours of BABC algorithms equipped with PM solving CapC benchmark problem



(a) Credit values

(b) Credit values in log scale

(c) Operator selection

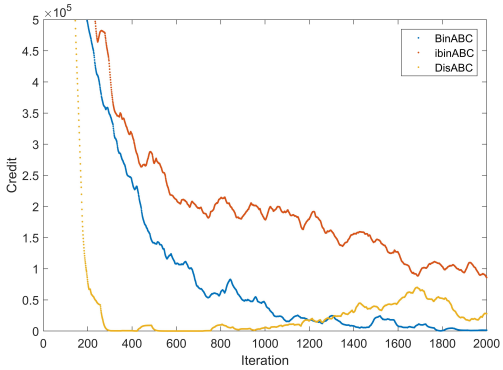
(d) Success of selected operators

comparatively while *binABC* under-performs.

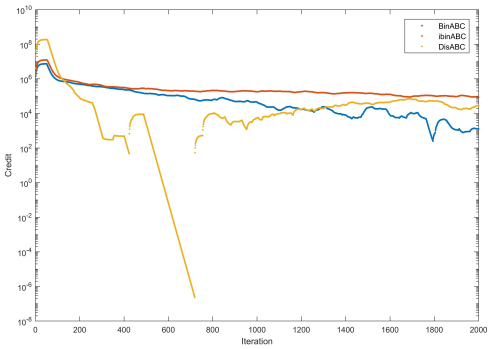
UCB approach with the best configuration to select operators is examined with experimental results collected for CapC benchmark, where each is repeated 30 times, and plotted in Figure 4, accordingly; the plots show (a) credit in normal scale, (b) log-scale credit, (c) operators' usage, and (d) the success of selected operators. It can be seen through plots, between 100 - 300 iterations, how impactful UCB can be on the balance between exploration and exploitation from the figures, especially, Figure 4(c). *disABC* performs better in the last 300 iterations, where it achieves 15% usage rate. *binABC* behaves in the same way as with the other two approaches with comparatively not performing well, but remains competitive with *ibinABC* in the race.

All three figures (Figure 2, 3, 4) help derive the characteristics of the operators. *disABC*

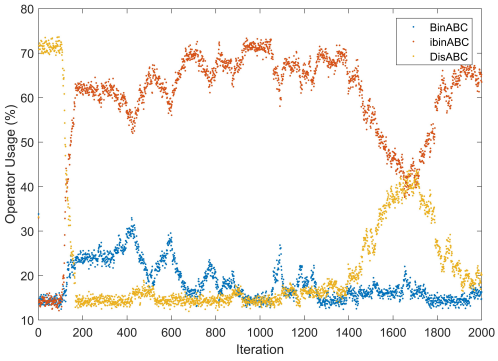
Figure 3: Dynamic behaviours of operators selected with AP solving CapC benchmark problem



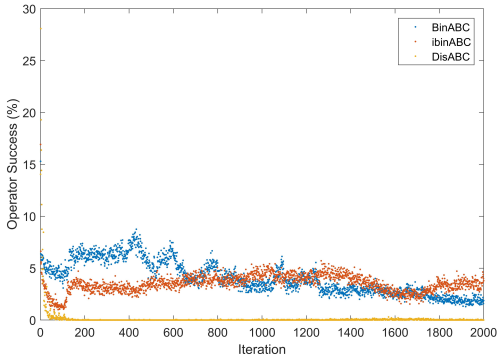
(a) Credit in normal scale



(b) Credit in log-scale

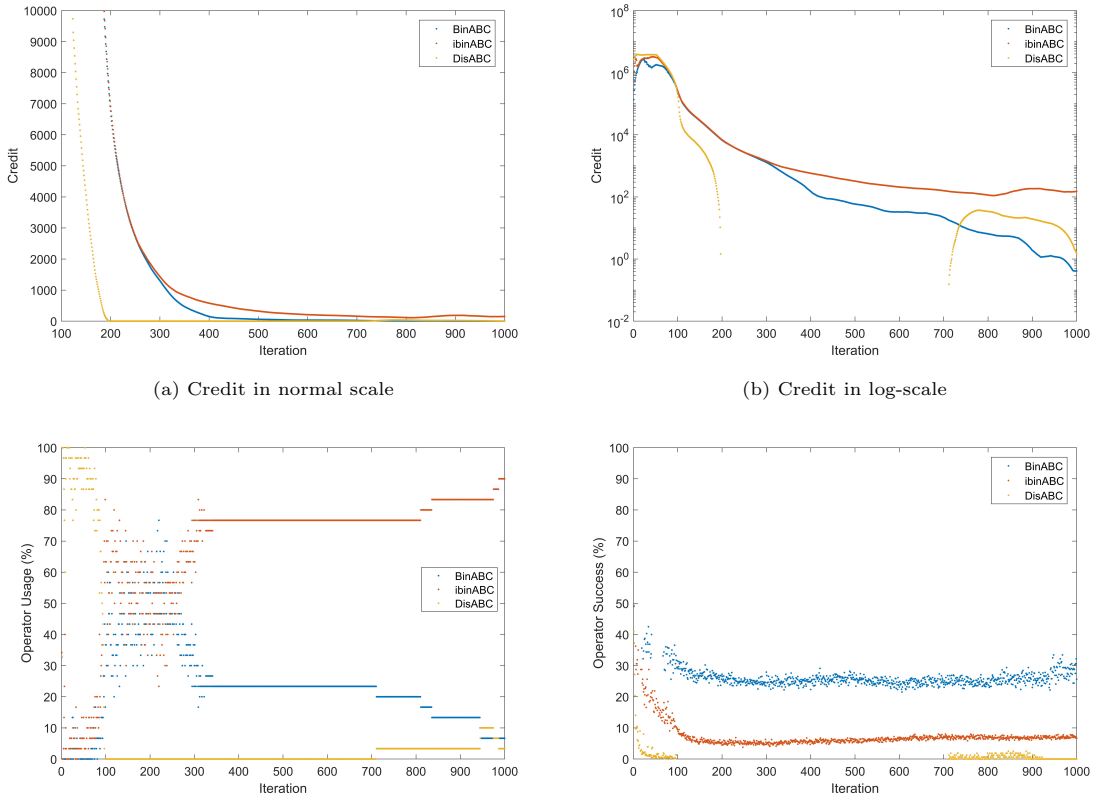


(c) Operator usage



(d) Success of selected operators

Figure 4: ABABC algorithms perform with UCB approach to solve CapC benchmark



(a) Credit in normal scale

(b) Credit in log-scale

(c) Operators' usage

(d) Success of selected operators

performs well in early stages, but, does not keep up in the later stages, while *binABC* helps improve the solution quality earlier, but steadily turns to be unproductive later, but *ibinABC* keeps better performance across the entire search process demonstrating with competitive results. PM and AP behave very similar, while operator change by AP seems sudden and sharper. UCB seems fueling competition among the operators initially, which falls in exploration phase, while promoting more promising operators in middle and later stages.

Table 9 presents the results of adaptive binary ABC algorithms equipped with either of operator selection schemes - PM, AP, or UCB - solving UFLP benchmark problems taken from OR-Library. Three configurations have been tested and tabulated, where the performance of each configuration is measured with three metrics; *Gap*, the difference between the optimum and the mean of replicates

Table 9: Comparative results with Binary ABC adopting different prioritisation approaches

	ABABC-PM			ABABC-AP			ABABC-UCB		
Instance	Gap	Std. Dev.	Hit	Gap	Std. Dev.	Hit	Gap	Std. Dev.	Hit
Cap71	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
Cap72	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
Cap73	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
Cap74	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
Cap101	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
Cap102	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
Cap103	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
Cap104	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
Cap131	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
Cap132	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
Cap133	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
Cap134	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
CapA	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
CapB	0.00	0.00	30	0.00	0.00	30	0.00	0.00	30
CapC	0.005	1428.00	25	0.004	1302.54	26	0.006	1532.69	24

in %, standard deviation denoted with *Std. Dev*, and *Hit*, the number of times optimum value is achieved. The first 14 of the benchmarks have been solved optimally by all three configurations in all 30 replications while the performances for CapC benchmark vary due to the difficulty of the benchmark, which has been used for all previous parameter fine-tuning purposes. CapC benchmark is the only problem instance helps differentiate the achievements of different configurations. As seen in Table 9, the ABABC versions with PM, AP and UCB solve the problem, CapC, optimally 25, 26 and 24 times out of 30, respectively. Apparently, the results are slightly different over *Gap* and *Hit* measures, but more distinctive in *Std.Dev*, where AP scheme seems doing better than other two. This may help conclude that configuration of ABABC with AP and *Extreme* rewarding is suggested to be adopted over the others.

5.2. Performance Evaluation

This section includes performance evaluations in comparison to relevant state-of-art literature. The comparisons include the results of ABABC-AP variant, found best performing in the previous sections, *binABC* [31], *disABC* [52] and *ibinABC* [35]. In order to keep the comparison fair, all algorithms have been run for the same number of functional evaluations and runtime. Apparently, the best performing algorithm is ABABC-AP, which solves all benchmarks optimum except CapC, which is solved to optimum 26/30 as indicated in Table 10 while the runner up algorithm is *ibinABC*, which solves 13 benchmarks to optimum, but other two do much worse in solving to optimum.

Table 11 provides performance results of a number of state-of-art non-ABC literature extracted from [53] for details. The extracted results are produced by the authors of [53] using Single point

Table 10: Comparative results between ABABC other Binary ABC variants

	binABC		disABC		ibinABC		ABABC-AP	
Instance	Gap	Std	Gap	Std	Gap	Std	Gap	Std. Dev.
Cap71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap72	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap73	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap74	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap101	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap102	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap103	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap104	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap131	0.00	0.00	0.62	2337.64	0.00	0.00	0.00	0.00
Cap132	0.00	0.00	0.095	813.37	0.00	0.00	0.00	0.00
Cap133	1.22	200.24	0.031	359.03	0.00	0.00	0.00	0.00
Cap134	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CapA	2.96	236833.50	0.152	74782.61	0.00	0.00	0.00	0.00
CapB	2.51	9143.13	3.303	109738.50	0.07	23762.93	0.00	0.00
CapC	2.58	82312.70	4.697	95778.78	0.062	11326.015	0.004	1302.54

crossover genetic algorithm, denoted with *GA-SP* (not clear taken from which reference), binary PSO, denoted as *BPSO* [54], binary artificial algae algorithm, denoted as *binAAA* [55] and XOR-based Jaya Algorithm, denoted as *JayaX* [53]. The results in Table 11 suggest that ABABC-AP, which is the best of this study, outperforms all others with solving 14 benchmarks to optimum, and CapC with the highest hits. The runner up algorithm, *JayaX* solves 12 benchmarks to optimum. It is clear that the most distinctive performances can be gained with solving CapA, CapB and CapC, where CapC is the most challenging one. It is noted that the ABABC variants without AP scheme, *binABC*, *disABC* and *ibinABC*, were under-performing in comparison to both *binAAA* and *JayaX*. This clearly demonstrates the contribution of adaptive mechanisms embedded in ABABC variants.

Table 11: Comparison with state-of-art

	GA-SP			BPSO			binAAA			JayaX			ABABC-AP		
Instance	Gap	Std. Dev.	Hit	Gap	Std. Dev.	Hit	Gap	Std. Dev.	Hit	Gap	Std. Dev.	Hit	Gap	Std. Dev.	Hit
Cap71	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
Cap72	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
Cap73	0.067	899.650	19	0.024	634.625	26	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
Cap74	0.000	0.000	30	0.009	500.272	29	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
Cap101	0.068	421.655	11	0.043	428.658	18	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
Cap102	0.000	0.000	30	0.010	321.588	28	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
Cap103	0.064	505.036	6	0.049	521.237	14	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
Cap104	0.000	0.000	30	0.041	1432.239	28	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
Cap131	0.068	720.877	16	0.171	1505.749	10	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
Cap132	0.000	0.000	30	0.058	1055.238	21	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
Cap133	0.091	685.076	10	0.083	690.192	10	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
Cap134	0.000	0.000	30	0.195	2594.211	18	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
CapA	0.046	22451.206	24	1.691	319855.431	8	0.000	0.000	30	0.000	0.000	30	0.000	0.000	30
CapB	0.584	66658.649	9	1.403	135326.728	5	0.248	39224.744	15	0.079	27033.020	26	0.000	0.000	30
CapC	0.705	51848.280	2	1.622	115156.444	1	0.295	29766.311	1	0.022	5455.940	17	0.004	1302.539	26

The experimental results presented in Table 10 and 11 demonstrate outperforming performance of the adaptive binary ABC algorithm in comparison to the recent state-of-art binary approaches including ABC variants. Although the results are self-declaring the significant performance, a statistical test is always the most secure way out to make sure the results lead to significant and sound conclusions. The results have been considered for comparisons with the performance of the state-of-art approaches using Wilcoxon signed-ranked test, which is a commonly used non-parametric statistical test approach. The tests have been conducted to see if the null-hypothesis, H_0, of "*there is no significant difference in between the results*" with confidence level of 95%. The functionalities of MATLAB 2017b has been used to calculate the test results over 30 repeats of each experimental setup.

The test results are tabulated in Table 12 showing the statistical test results for UFLP benchmark problem instances on the row and binary swarm intelligence algorithms in the column. The test results per algorithm indicate (i) probability, p-value, and (ii) wining hypothesis, H. It is observed that many of the comparisons confirm the H_0 with p-value, $p = 1.0$ meaning that adaptive binary ABC performs similar to the state-of-art approaches in solving the benchmark problems. It can also be observed in Table 11 in which results fall at the same level of performance. This is not surprising since all algorithms can solve these benchmark problems very easily. The main challenge comes through the benchmark problems labelled as CapB and CapC with which the performance of the algorithms can significantly differ. It clearly shows that the test results for the majority of CapB and all of CapC instances indicate "rejection of null hypothesis" with $H = 1$ and various p values that confirm the difference in performance levels as shown in Table 11. This concludes that adaptive binary ABC's performance significantly differs in solving the most difficult benchmarks, where it solves not only all easy and moderately difficult benchmarks to optimum similar to its rivals but also the most difficult benchmarks, i.e. CapB and CapC, to optimum. The difference in performance is found statistically different and significantly better than the others.

6. Conclusion

This paper presents a study that investigates the viability of devising ABC algorithms with multiple operators selected adaptively for solving binary problems. Adaptive selection schemes have been researched and tested in various configurations and the best performing scheme working with binary ABC is sought to solve uncapacitated facility location problem instances. Three adaptive

Table 12: The results of the Wilcoxon signed-rank test of the proposed method with the state-of-art

Benchmark	AAA		JayaX		BPSO		GA-SP		ibinABC		disABC		binABC	
	p-value	H	p-value	H	p-value	H	p-value	H	p-value	H	p-value	H	p-value	H
Cap71	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Cap72	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Cap73	1	0	1	0	1.E-01	0	1.E-03	1	1	0	1	0	1	0
Cap74	1	0	1	0	3.E-06	1	4.E-08	1	1	0	1	0	1	0
Cap101	1	0	1	0	2.E-01	0	4.E-04	1	1	0	1	0	1	0
Cap102	1	0	1	0	5.E-01	0	1	0	1	0	1	0	1	0
Cap103	1	0	1	0	1.E-06	1	1.E-06	1	1	0	1	0	1	0
Cap104	1	0	1	0	5.E-01	0	1	0	1	0	1	0	1	0
Cap131	1	0	1	0	1.E-06	1	1.E-06	1	1	0	2.E-06	1	1	0
Cap132	1	0	1	0	1.E+00	0	4.E-08	1	1	0	4.E-06	1	1	0
Cap133	1	0	1	0	2.E-06	1	1.E-06	1	1	0	1.E-06	1	1.E-07	1
Cap134	1	0	1	0	5.E-04	1	1	0	1	0	1	0	1	0
CapA	1	0	1	0	5.E-05	1	1.E-01	0	1	0	3.E-01	0	6.E-06	1
CapB	6.E-05	1	2.E-07	1	2.E-06	1	2.E-06	1	3.E-02	0	2.E-05	1	1.E-05	1
CapC	4.E-06	1	8.E-01	0	2.E-06	1	2.E-06	1	2.E-03	1	4.E-06	1	2.E-06	1

selection schemes, namely *probability matching* (PM), *adaptive pursuit* (AP) and *upper confidence bound* (UCB), have been tested and are found very competitive in performance. All three schemes are implemented with three recently developed binary operators, *binABC*, *disABC* and *ibinABC*, and tested accordingly. A number of variants of binary ABC algorithm were configured with a number of parameters required by each of three operator selection schemes and have been tested with UFLP benchmarks. The configuration tests reveal that *AP* is the best performing scheme and *extreme* rewarding remains preferable, subject to given circumstances. Hence, it is concluded that a binary ABC equipped with the three operators adaptively exploited using *AP* scheme and *Extreme* rewarding approach outperforms all competing algorithms. In addition, all three non-adaptive variants of binary ABC, *binABC*, *disABC* and *ibinABC*, perform much worse than the adaptive variants. The configured new binary ABC is named adaptive binary ABC (ABABC) and comparatively evaluated against the most relevant state-of-art approaches in the literature, where ABABC significantly outperforms all rivals/alternatives with an achievement of solving 14 out of 15 benchmark problems to optimum, and solving 26 of 30 replicates of CapC to optimum. The significance of this performance has been statistically tested and found significant in 95% confidence level, is the sound highest achievement so far.

In the next step of this research, machine learning algorithms would be used to replace adaptive selection schemes for improved efficiency. This is expected to impose smarter operator selection schemes, which can be used as smartly guided adaptive search algorithms in which the operators, and even the policies, i.e. selection schemes, can be smartly chosen among alternatives so as to prevent the search from local optima with reduced complexity.

References

[1] J. Del Ser, E. Osaba, D. Molina, X.-S. Yang, S. Salcedo-Sanz, D. Camacho, S. Das, P. N. Suganthan, C. A. C. Coello, F. Herrera, Bio-inspired computation: Where we stand and what's next, Swarm and Evolutionary Computation 48 (2019) 220–250.

[2] S. Sadeghi-Moghaddam, M. Hajiaghaei-Keshteli, M. Mahmoodjanloo, New approaches in meta-heuristics to solve the fixed charge transportation problem in a fuzzy environment, Neural Computing and Applications 31 (1) (2019) 477–497.

[3] A. E. Drake, R. E. Marks, Genetic algorithms in economics and finance: Forecasting stock market prices and foreign exchange—a review, in: Genetic algorithms and genetic programming in computational finance, Springer, Boston, MA, 2002, pp. 29–54.

[4] G. Hiermann, M. Prandtstetter, A. Rendl, J. Puchinger, G. R. Raidl, Metaheuristics for solving a multimodal home-healthcare scheduling problem, Central European Journal of Operations Research 23 (1) (2015) 89–113.

[5] X.-S. Yang, Engineering optimization: an introduction with metaheuristic applications, John Wiley & Sons, Hoboken, New Jersey, 2010.

[6] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, IEEE computational intelligence magazine 1 (4) (2006) 28–39.

[7] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95-International Conference on Neural Networks, Vol. 4, IEEE, 1995, pp. 1942–1948.

[8] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm, Journal of global optimization 39 (3) (2007) 459–471.

[9] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE transactions on evolutionary computation 1 (1) (1997) 67–82.

[10] Y. Zhang, S. Wang, G. Ji, A comprehensive survey on particle swarm optimization algorithm and its applications, Mathematical Problems in Engineering 2015 (2015) 931256.

[11] S. Das, P. N. Suganthan, Differential evolution: A survey of the state-of-the-art, IEEE transactions on evolutionary computation 15 (1) (2010) 4–31.

[12] D. Karaboga, B. Gorkemli, C. Ozturk, N. Karaboga, A comprehensive survey: artificial bee colony (abc) algorithm and applications, Artificial Intelligence Review 42 (1) (2014) 21–57.

[13] M. Düğenci, M. E. Aydin, A honeybees-inspired heuristic algorithm for numerical optimisation, Neural Computing and Applications 32 (16) (2020) 12311–12325.

[14] E. L. Lawler, The quadratic assignment problem, Management science 9 (4) (1963) 586–599.

[15] C. Ozturk, E. Hancer, D. Karaboga, Dynamic clustering with improved binary artificial bee colony algorithm, Applied Soft Computing 28 (2015) 69–80.

[16] P. Espahbodi, Identification of problem banks and binary choice models, Journal of Banking & Finance 15 (1) (1991) 53–71.

[17] S. Sahni, Approximate algorithms for the 0/1 knapsack problem, Journal of the ACM (JACM) 22 (1) (1975) 115–124.

[18] V. Chvatal, A greedy heuristic for the set-covering problem, Mathematics of operations research 4 (3) (1979) 233–235.

[19] M. Tuba, N. Bacanin, Artificial bee colony algorithm hybridized with firefly algorithm for cardinality constrained mean-variance portfolio selection problem, Applied Mathematics & Information Sciences 8 (6) (2014) 2831.

[20] A. Kumar, V. Kumar, Hybridized abc-ga optimized fractional order fuzzy pre-compensated fopid control design for 2-dof robot manipulator, AEU-International Journal of Electronics and Communications 79 (2017) 219–233.

[21] M. Düğenci, M. E. Aydin, Diversifying search in bee algorithms for numerical optimisation, in: International Conference on Computational Collective Intelligence, Springer, 2018, pp. 132–144.

[22] Y. Xue, J. Jiang, B. Zhao, T. Ma, A self-adaptive artificial bee colony algorithm based on global best for global optimization, Soft Computing 22 (9) (2018) 2935–2952.

[23] S. L. Scott, A modern bayesian look at the multi-armed bandit, Applied Stochastic Models in Business and Industry 26 (6) (2010) 639–658.

[24] J. Niehaus, W. Banzhaf, Adaption of operator probabilities in genetic programming, in: European Conference on Genetic Programming, Springer, 2001, pp. 325–336.

[25] K. Li, A. Fialho, S. Kwong, Q. Zhang, Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition, IEEE Transactions on Evolutionary Computation 18 (1) (2013) 114–130.

[26] L. Davis, Adapting operator probabilities in genetic algorithms, in: Proceedings of the third international conference on Genetic algorithms, 1989, pp. 61–69.

[27] Á. Fialho, L. Da Costa, M. Schoenauer, M. Sebag, Analyzing bandit-based adaptive operator selection mechanisms, Annals of Mathematics and Artificial Intelligence 60 (1-2) (2010) 25–64.

[28] J. E. Beasley, Or-library: distributing test problems by electronic mail, Journal of the operational research society 41 (11) (1990) 1069–1072.

[29] G. Wu, R. Mallipeddi, P. N. Suganthan, Ensemble strategies for population-based optimization algorithms–a survey, Swarm and evolutionary computation 44 (2019) 695–711.

[30] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Tech. rep., Technical report-tr06, Erciyes university, engineering faculty, computer engineering department (2005).

[31] M. S. Kiran, The continuous artificial bee colony algorithm for binary optimization, Applied Soft Computing 33 (2015) 15–23.

[32] M. H. Kashan, N. Nahavandi, A. H. Kashan, Disabc: A new artificial bee colony algorithm for binary optimization, Applied Soft Computing 12 (1) (2012) 342–352.

[33] M. S. Kiran, M. Gündüz, Xor-based artificial bee colony algorithm for binary optimization, Turkish Journal of Electrical Engineering & Computer Sciences 21 (Sup. 2) (2013) 2307–2328.

[34] C. J. Santana Jr, M. Macedo, H. Siqueira, A. Gokhale, C. J. Bastos-Filho, A novel binary artificial bee colony algorithm, Future Generation Computer Systems 98 (2019) 180–196.

[35] R. Durgut, Improved binary artificial bee colony algorithm, Frontiers of Information Technology & Electronic Engineering (in press) (2020).

[36] H. Wang, Z. Wu, S. Rahnamayan, H. Sun, Y. Liu, J.-s. Pan, Multi-strategy ensemble artificial bee colony algorithm, Information Sciences 279 (2014) 587–603.

[37] W.-f. Gao, L.-l. Huang, S.-y. Liu, F. T. Chan, C. Dai, X. Shan, Artificial bee colony algorithm with multiple search strategies, Applied Mathematics and Computation 271 (2015) 269–287.

[38] X. Chen, H. Tianfield, K. Li, Self-adaptive differential artificial bee colony algorithm for global optimization problems, Swarm and Evolutionary Computation 45 (2019) 70–91.

[39] M. S. Kiran, H. Hakli, M. Gunduz, H. Uguz, Artificial bee colony algorithm with variable search strategy for continuous optimization, Information Sciences 300 (2015) 140–157.

[40] C. Ozturk, E. Hancer, D. Karaboga, A novel binary artificial bee colony algorithm based on genetic operators, Information Sciences 297 (2015) 154–170.

[41] Y. He, H. Xie, T.-L. Wong, X. Wang, A novel binary artificial bee colony algorithm for the set-union knapsack problem, Future Generation Computer Systems 78 (2018) 77–86.

[42] D. Jia, X. Duan, M. K. Khan, Binary artificial bee colony optimization using bitwise operation, Computers & Industrial Engineering 76 (2014) 360–365.

[43] M. Sevkli, M. E. Aydin, A variable neighbourhood search algorithm for job shop scheduling problems, in: European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, 2006, pp. 261–271.

[44] M. E. Aydin, Coordinating metaheuristic agents with swarm intelligence, Journal of Intelligent Manufacturing 23 (4) (2012) 991–999.

[45] Á. Fialho, L. Da Costa, M. Schoenauer, M. Sebag, Extreme value based adaptive operator selection, in: International Conference on Parallel Problem Solving from Nature, Springer, 2008, pp. 175–184.

[46] Á. Fialho, M. Schoenauer, M. Sebag, Toward comparison-based adaptive operator selection, in: Proceedings of the 12th annual conference on Genetic and evolutionary computation, 2010, pp. 767–774.

[47] M. Aydin, T. Fogarty, A distributed evolutionary simulated annealing algorithm for combinatorial optimisation problems, Journal of Heuristics 10 (3) (2004) 269–292.

[48] K. Chan, M. Aydin, T. Fogarty, Main effect fine-tuning of the mutation operator and the neighbourhood function for uncapacitated facility location problems, Soft Computing 10 (11) (2006) 1075–1090.

[49] V. Yigit, M. Aydin, O. Turkbey, Solving large-scale uncapacitated facility location problems with evolutionary simulated annealing, International Journal of Production Research 44 (22) (2006) 4773–4791.

[50] F. Glover, S. Hanafi, O. Guemri, I. Crevits, A simple multi-wave algorithm for the uncapacitated facility location problem, Frontiers of Engineering Management 5 (4) (2018) 451–465.

[51] J. Kratica, D. Tošic, V. Filipović, I. Ljubić, Solving the simple plant location problem by genetic algorithm, RAIRO - Operations Research 35 (1) (2001) 127–142.

[52] S. Korkmaz, A. Babalik, M. S. Kiran, An artificial algae algorithm for solving binary optimization problems, International Journal of Machine Learning and Cybernetics 9 (7) (2018) 1233–1247.

[53] M. Aslan, M. Gunduz, M. S. Kiran, Jayax: Jaya algorithm with xor operator for binary optimization, Applied Soft Computing 82 (2019) 105576.

[54] J. Kennedy, R. C. Eberhart, A discrete binary version of the particle swarm algorithm, in: 1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation, Vol. 5, IEEE, 1997, pp. 4104–4108.

[55] X. Zhang, C. Wu, J. Li, X. Wang, Z. Yang, J.-M. Lee, K.-H. Jung, Binary artificial algae algorithm for multidimensional knapsack problems, Applied Soft Computing 43 (2016) 583–595.