Binary Artificial Algae Algorithm for Multidimensional Knapsack Problems

Xuedong Zhang[a], Changzhi Wu[b], Jing Li[c], Xiangyu Wang[b,d], Zhijing Yang[e,*], Jae-Myung Lee[f], Kwang-Hyo Jung[f]

[a]*School of Management Science and Engineering,*
Anhui University of Finance & Economics, Bengbu 233000, China
[b]*Australasian Joint Research Centre for Building Information Modelling,*
School of Built Environment, Curtin University, Perth, WA 6845, Australia
[c]*Information and Intelligence Engineering Department,*
Anhui Vocational College of Electronics & Information Technology, Bengbu 233000, China
[d]*Department of Housing and Interior Design, Kyung Hee University, Seoul, Korea*
[e]*School of Information Engineering, Guangdong University of Technology, Guangzhou, 510006, China*
[f]*Department of Naval Architecture and Ocean Engineering, Pusan National University, Busan, Korea*

Abstract

The multidimensional knapsack problem (MKP) is a well-known NP-hard optimization problem. Various meta-heuristic methods are dedicated to solve this problem in literature. Recently a new meta-heuristic algorithm, called artificial algae algorithm (AAA), was presented, which has been successfully applied to solve various continuous optimization problems. However, due to its continuous nature, AAA cannot settle the discrete problem straightforwardly such as MKP. In view of this, this paper proposes a binary artificial algae algorithm (BAAA) to efficiently solve MKP. This algorithm is composed of discrete process, repair operators and elite local search. In discrete process, two logistic functions with different coefficients of curve are studied to achieve good discrete process results. Repair operators are performed to make the solution feasible and increase the efficiency. Finally, elite local search is introduced to improve the quality of solutions. To demonstrate the efficiency of our proposed algorithm, simulations and evaluations are carried out with total of 94 benchmark problems and compared with other bio-inspired state-of-the-art algorithms in the recent years including MBPSO, BPSOTVAC, CBPSOTVAC, GADS, bAFSA, and IbAFSA. The results show the superiority of BAAA to many compared existing algorithms.

Keywords: Artificial algae algorithm; Multidimensional knapsack problem; Pseudo-utility ratio; Elite local search

---

[*] Corresponding author
Email address: `yzhj@gdut.edu.cn` (Zhijing Yang)

1. Introduction

Knapsack problems are found in many science and engineering applications such as finite word length filter design problems [1]. The decision vectors are discrete valued. One common approach to address this issue is to approximate the problems by the optimization problems with continuous valued decision vectors and some advanced techniques [2, 3, 4, 5] are applied to find the solution of these problems. To address the original optimization with the discrete valued decision vectors, the 0-1 multidimensional knapsack problem (MKP) is a well-known NP-hard optimization problem [6]. Given a set of items with non-negative weights and values (profits), MKP is to select some of the items to put into knapsack with specified capacity constraints such that the profit is maximized without violating the constraints. A standard MKP is given as follows [7]:

$$\begin{aligned} &\max\ f(x) = \sum_{i=1}^{d} p_i x_i, \quad i = 1, 2..., d, \\ \text{s.t.} \qquad &\sum_{i=1}^{d} c_{ij} x_i \leq b_j, \quad i = 1, 2...d,\ j = 1, 2..., m, \\ &x_i \in \{0, 1\}, \quad i = 1, 2..., d, \end{aligned} \tag{1}$$

where d is the number of items and m is the number of knapsack constraints; p_i is the profit of ith item if it is put into knapsack; x_i is either 1 or 0, where 1 denotes the ith item being stored into the knapsack and 0 denotes ith item being discarded, respectively; c_{ij} is the consumption of jth resource while putting the ith item into knapsack and b_j is the total capacity of jth resource. Without loss of generality, it is assumed that $p_i > 0$, $0 \leq c_{ij} < b_j$ and $\sum_{i=1}^{d} c_{ij} > b_j$.

In nature, MKP is a typical integer programming problem with d variables and m constraints. In the past decades, MKP has been investigated and applied in cutting stock, loading problem, project selection and resource allocation [8]. Plenty of methods were introduced to solve MKP in recent years including deterministic and approximate algorithms [9]. Some exact algorithms like dynamic programming [7, 10], branch and bound algorithm [11] and hybrid algorithms [12, 13] can solve small-scaled and medium-scaled problems within endurable time. As the number of items and constraints increase, the performance of exact algorithm declines rapidly and becomes intolerable. With the development of intelligent computing, many new approximate methods emerge such as heuristic and meta-heuristic algorithms. This type of algorithms can find optimal, sub-optimal or at least satisfactory solutions in most cases, although the optimum is not guaranteed. Such algorithms include genetic algorithm [14, 15, 16], tabu search [17], simulated annealing [18], particle swarm optimization [19, 20], firefly algorithm [21], harmony search [22, 23] and artificial fish swarm algorithm [24, 25], etc. Evolutionary computation and bio-inspired algorithms are the fastest developing type of algorithms. The basic idea of them is that from an initial population of individuals, solution vectors, individuals evolve by some way to produce new better individuals and keep better ones in the next generation(iteration), whereas the worse individuals are discarded in the next generation. A satisfactory solution will be obtained after updating some generations. More details can be found in [26, 27].

In [14], genetic algorithm was utilized to solve MKP. This method has been further improved by Djannaty in [15] where initial population created by Dantzig algorithm and penalty function to increase the rate of convergence of MKP were introduced. In [28], a binary version of PSO is introduced by Kennedy to solve discrete optimization problems. In [20], a modified binary particle swarm optimization (MBPSO) algorithm is proposed for 0-1 knapsack problem and multidimensional knapsack problem. MBPSO introduced a new probability function to improve the diversity and made it more effective than simple binary version of PSO. In [29], binary PSO with time-varying acceleration coefficients (BPSOTVAC) and chaotic binary PSO with time-varying acceleration coefficients (CBPSOTVAC) were proposed. Through introducing the time-varying inertia weight and time-varying learning factors, the performance of the solution had been improved significantly. In [30], a particle swarm optimization with self-adaptive check and repair operator (SACRO) was presented to improve the efficiency of PSO, where SACRO will change the alternative pseudo-utility ratio dynamically. In [25], a binary version of the artificial fish swarm algorithm was proposed where a decoding scheme was introduced to transform infeasible solutions to be feasible for multidimensional knapsack problem. In [23], an effective hybrid algorithm based on harmony search (HHS) was presented to solve multidimensional knapsack problems. HHS developed a novel harmony improvisation mechanism with modified memory consideration rule and global-best pitch adjustment scheme. In addition, the fruit fly optimization (FFO) scheme was integrated as a local search strategy. Compared with an improved adaptive binary harmony search algorithm (ABHS) [31] and a novel global harmony search algorithm (NGHS) [32], HHS demonstrated the effectiveness and robustness.

In the recent years, a new meta-heuristic algorithm, artificial algae algorithm (AAA), was presented [33]. Similar to other bio-inspired algorithms, AAA was inspired by the lifestyles of algae. AAA has been successfully applied in the optimization of benchmark functions with various dimensions in CEC'05 [34] and implemented on the pressure vessel problem. However, due to its continuous nature, AAA cannot settle the discrete problem straightforwardly such as MKP. In view of this problem, this paper proposes a binary artificial algae algorithm (BAAA) to solve MKP. Compared with many bio-inspired binary version algorithms in well-known benchmarks for MKP, BAAA achieves better performance in terms of robustness as well as the best solution obtained.

2. Introduction to Artificial Algae Algorithm (AAA) in [33]

In the recent years, a new artificial algorithm, named as artificial algae algorithm (AAA), is proposed to solve continuous optimization problems [33]. AAA simulates real algae to survive by finding and moving to the appropriate environment, and reproduce next generation. In this section, we will review AAA briefly. More details on AAA can be found in [33].

Denote the algae population which comprises of a number of algal colonies as below:

$$Population\ of\ algal\ colony = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \cdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix} \tag{2}$$

Set $x_i = (x_{i1},\ x_{i2}, \cdots,\ x_{id}),\ i = 1, 2, ..., n$, where each x_i represents a feasible solution in solution space. Each algal colony contains a group of algal cells which are regarded as the elements of a solution. All the algal cells in an algal colony are considered as a whole to move together towards a suitable place with abundant resources. As the colony reaches a ideal position, optimum solution is obtained.

In the artificial algae algorithm, there are three key parts which are helical movement, evolutionary process and adaptation. The algal colony tries to move to a optimal position through moving, evolving and adapting itself. It is worth to mention that a crucial concept in AAA is the size of algal colony of *ith* algal colony denoted as $S_i,\quad i = 1, 2, ..., n$. Similar to the real algae, under perfect living condition, the algal colony will reproduce and grow to a bigger size. Living in a bad environment will lead to death of algal cells and shrink of algal colony. S_i is set as 1 at the initial stage, and altered with the change of the fitness value of the *ith* algal colony, i.e. the value of objective function. The better the objective function $f(x_i)$ is, the bigger S_i is. S_i is updated according to the biological growth process given as follows:

$$S_i = size(x_i) \tag{3}$$

$$\mu_i = \frac{S_i + 4f(x_i)}{S_i + 2f(x_i)} \tag{4}$$

$$S_i^{t+1} = \mu_i S_i^t, \quad i = 1, 2, ..., n \tag{5}$$

where $f(x_i)$ is the objective function, μ_i is the update coefficient of S_i, t represents the current generation.

2.1. Helical movement

Algae make instinctive movement to the water areas which have adequate light and other nutrients. In AAA, each algal colony moves towards the best algal colony which has the biggest size or optimal objective function value. Similar to the movement in three dimensions of the object in real world, algal colony moves in three dimensions as well. However, this movement is simulated by selecting three distinct algal cells randomly and changing their positions. Eq. (6) represents the movement in the first dimension and can be used for one-dimensional problems. Eqs. (7) and (8) indicate movement in other two dimensions.

$$x_{im}^{t+1} = x_{im}^t + (x_{jm}^t - x_{im}^t)(sf - \omega_i)p \tag{6}$$

$$x_{ik}^{t+1} = x_{ik}^t + (x_{jk}^t - x_{ik}^t)(sf - \omega_i)\cos\alpha \tag{7}$$

$$x_{il}^{t+1} = x_{il}^t + (x_{jl}^t - x_{il}^t)(sf - \omega_i)\sin\beta \tag{8}$$

where m, k and l are random integers uniformly generated between 1 and d, x_{im}, x_{ik} and x_{il} simulate x, y and z coordinates of the ith algal colony, j indicates the index of a neighbor algal colony and is obtained by tournament selection, p is an independent random real-valued number between -1 and 1, α and β are random degrees of arc between 0 and 2π, sf is shear force which exists as viscous drag, ω_i is the friction surface area of ith algal colony which is proportional to the size of algal colony. Due to the spherical shape of algal colony, friction surface is deduced as the surface area of the hemisphere which can wrap up the algal colony. ω_i is calculated as follows:

$$\omega_i = 2\pi r_i^2 \tag{9}$$

$$r_i = (\sqrt[3]{\frac{3S_i}{4\pi}}) \tag{10}$$

where r_i represents the radius of the hemisphere of the ith algal colony, and S_i is its size.

2.2. Evolutionary process

In natural environment, algal colony with adequate nutrient source grows rapidly and that with scarce nutrient source will wither to die. Similarly, in AAA, algal colony x_i becomes bigger if it moves to an ideal position and obtains more feasible solution. While a iteration terminates, the smallest algal colony withers and an algal cell of the smallest algal colony is substituted by an algal cell of the biggest algal colony. This process is simulated as the following equations:

$$biggest = arg\ max\{size(x_i)\}, \quad i = 1, 2, ..., n \tag{11}$$

$$smallest = arg\ min\{size(x_i)\}, \quad i = 1, 2, ..., n \tag{12}$$

$$smallest_j = biggest_j, \quad j = 1, 2, \ldots, d. \tag{13}$$

where *biggest* and *smallest* represent the biggest and smallest algal colony, respectively, j is the index of a randomly selected algal cell.

2.3. Adaptation

In the growing process, algal colony suffers from starvation under insufficient light and nutrient. Adaptation is the process in which starved algal colony tries to move towards the biggest colony and adapts itself to the environment. Starvation value is set to zero from beginning, and increases with the helical movement. The movement makes the fitness of algal colony either better or worse. Thus, the objective function value becomes superior or inferior to the value after movement. If the objective function gets better value, the corresponding algal colony remains its starvation level unchanged. Otherwise, the starvation value increases by one. After movement of algal colony ends in an iteration, the algal colony that has the highest starvation value (Eq. (14)) adapts itself to the biggest algal colony with a probability A_p. In the adaptation phase of original AAA [33], the adaptation of the algal colony was implemented by adapting every single algal cell. For the sake of clarity, we introduce Eq. (15) to illustrate this process:

$$x_s = arg\ max\{starvation(x_i)\}, \quad i = 1, 2, ..., n \tag{14}$$

$$x_{sj}^{t+1} = \begin{cases} x_{sj}^t + (biggest_j - x_{sj}^t) \times rand1, & if\ rand2 < A_p; \\ x_{sj}^t, & otherwise. \end{cases} \quad j = 1, 2, \ldots, d \tag{15}$$

where s is the index of algal colony which has the highest starvation value, and $starvation(x_i)$ measures the starvation level of algal colony x_i, j is the index of algal cell, $rand1$ and $rand2$

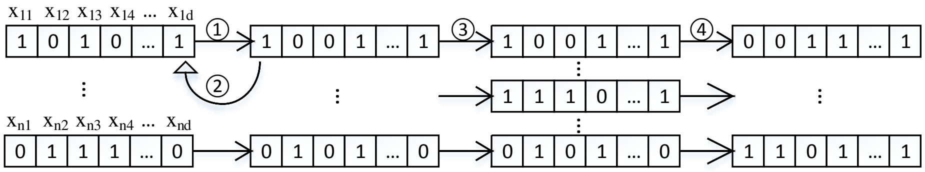

Figure 1: Encoding example of BAAA.

generate stochastic real-valued numbers between 0 and 1, A_p is the adaptation probability which decides whether adaptation occurs or not, A_p is a constant usually being set between 0.3 and 0.7.

3. Binary artificial algae algorithm (BAAA)

AAA was initially proposed to solve continuous nonlinear optimization problems. Therefore, all computation in AAA, such as helical movement, evolutionary process and adaptation are continuous. However, MKP is a typical discrete optimization problem. AAA cannot be applied directly. Here we will introduce a binary version of AAA, namely BAAA, to solve MKP. At the initialization stage, algal colony x_i is initialized as a binary string of length d with 0 or 1. Each algal cell x_{ij} is generated according to the following equation:

$$x_{ij} = \begin{cases} 0, & if\ rand < 0.5; \\ 1, & otherwise. \end{cases} \tag{16}$$

Then, the population of algal colony is encoded as n binary strings and each string is a candidate solution for MKP. An encoding example is illustrated in Fig. 1 which demonstrates the changing process of population in one iteration. In Fig. 1, ① denotes each algal colony is transformed into a new binary string through helical movement. ② indicates algal colony moves until its energy runs out. ③ represents the evolutionary process which leads to the inversion of one bit in a specified binary string. ④ means each binary string adapts itself according to the adaptation probability.

3.1. Discrete process

Due to its continuous nature of AAA, the intermediate results tend to be real-valued number and cannot be applied to MKP straightforwardly. Discrete method should be introduced to transfer real number into binary number 0 or 1. Sigmoid function is a type of mathematical function which is defined for all real input values with bound outputs ranging from 0 to 1.

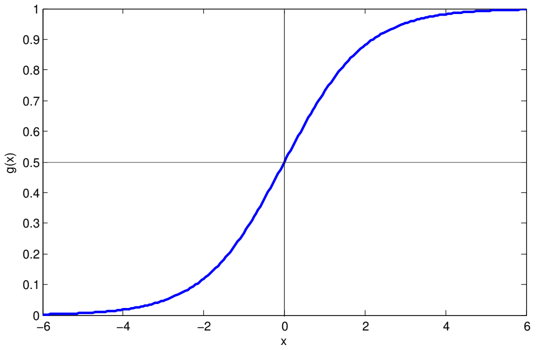

Figure 2: Sigmoid curve of logistic function.

Logistic function is the special case of sigmoid function (see Eq. (17)) and its figure is shown in Fig. 2.

$$g(x) = \frac{1}{e^{-x} + 1} \tag{17}$$

$$x_{ij} = \begin{cases} 0, & if\ g(x) < rand; \\ 1, & otherwise. \end{cases} \tag{18}$$

In real applications, two variants of logistic function, called $Tanh(x)$ and $Sig(x)$, are often used. Here $Tanh(x)$ and $Sig(x)$ are defined as:

$$g(x) = Tanh(x) = \frac{e^{\tau|x|} - 1}{e^{\tau|x|} + 1} \tag{19}$$

$$g(x) = Sig(x) = \frac{1}{e^{-\tau x} + 1} \tag{20}$$

where τ is a controlling parameter which determines the changing trend of the curve. Combined with Eq. (18), a discrete value 0 or 1 is produced through comparing g(x) with a random distributed value between 0 and 1. Fig. 3 illustrates the figure of $Tanh(x)$ and $Sig(x)$ with different τ. As seen in Fig. 3, the smaller τ is, the less steepness of the curves have. When τ is very small, the curve tends to be a horizontal line. Taking $Sig(x)$ as an example, when $\tau = 0.1$, the values of function are close to 0.5 which makes the discrete procedure like a random selection. As a result, the algorithm is led to poor exploitation and easy to fall into local optimum. On the other hand, when τ is large, the curve becomes much steep which leads to low diversity and poor exploration. For example, if $x > 5$ and $\tau = 3.5$, then g(x) is very close to 1. For this case, Eq. (18) has little chance to produce 0. This clearly shows that proper τ is crucial for the discrete procedure. An experiment is carried out in the next section for the selection of τ.

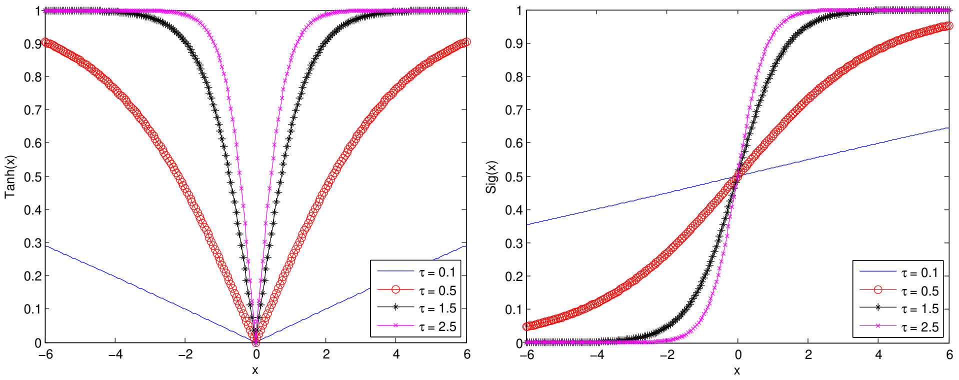

Figure 3: Comparison of $Tanh(x)$ and $Sig(x)$ with different τ.

3.2. Repair operator

In the initialization and discrete process, the solution vectors with 0 or 1 are produced without considering their feasibility. However, they are likely to be infeasible solutions in spite of their high fitness values, and they may mislead the search into hopeless situation. As is known to all, as the solution of MKP, the binary string should satisfy all the constraints. Therefore, each candidate solution must be checked and modified to meet every constraint. Moreover, total fitness value is to be enhanced as high as possible. This idea can be realized by two stages. The first stage is to adjust the infeasible solution to feasible one by discarding some items from the knapsack and setting the responding item value from 1 to 0. The second stage is to utilize the remainder space of the knapsack completely by putting some items into the knapsack and setting the responding item value from 0 to 1. In order to choose appropriate items for previous operation, a selection mechanism must be determined. Several techniques were proposed in the literatures. [35] first introduced the pseudo-utility in the surrogate duality approach. The pseudo-utility of each variable was given below:

$$\delta_i = \frac{p_i}{\sum_{j=1}^{m} w_j c_{ij}}, \quad i = 1, 2, ..., d \tag{21}$$

where w_j is surrogate multiplier between 0 and 1 which can be viewed as shadow prices of the jth constraint in the linear programming (LP) relaxation of the original MKP. Obviously, w_j is a key value to determine the selection of items. An optimal set of surrogate multipliers can effectively measure the consumption level of resources for each item, and improve the final repair effect. However, it is hard to find the optimal set of w_j, especially when $m + n$ is very large. To overcome this drawback, [36] presented a new metric called relative mean resource occupation defined as:

$$\delta_i = \frac{\sum_{j=1}^{m} \frac{c_{ij}}{m \cdot b_j}}{p_i}, \quad i = 1, 2, ..., d \tag{22}$$

In addition, another two common used pseudo-utilities [30], i.e. profit/weight utility and relative profit density, are:

$$\overline{\delta_i} = min\{\frac{p_i}{c_{ij}}\}, \quad i = 1, 2, ..., d, \ j = 1, 2, ..., m \tag{23}$$

$$\widetilde{\delta_i} = min\{\frac{p_i \cdot b_j}{c_{ij}}\}, \quad i = 1, 2, ..., d, \ j = 1, 2, ..., m \tag{24}$$

Eq. (23) calculates the ratio of profit and weight. The greater the ratio is, the more possible the item being selected into knapsack. Considering $c_{ij}, j = 1, 2, ..., m$ have m values for item $\overline{\delta_i}$, only the smallest value of the ratios is adopted to measure the pseudo-utility. Compared with Eq. (23), Eq. (24) not only takes profit/weight into account but also introduces the capacities in each dimension, i.e. profit density. Three different measures of pseudo-utility ratios produce different ranking of ratios and lead to various packing sequence. An experimental comparison among them will be implemented in Section 4.

After pseudo-utility ratios are calculated, the pseudo-utilities are ranked to ascending order. Then, two repair operators are performed for making the solution feasible and improving the quality of solution, respectively. The first is DROP operator in which some items will be removed from the knapsack if the solution is infeasible. The DROP operator selects the item from the knapsack with smallest value of pseudo-utility and changes the responding bit from 1 to 0 until the solution is feasible. The second is ADD operator in which some items will be added into the knapsack as much as possible. The ADD operator examines each item in the descending order of pseudo-utility, and tries to pack the item in the knapsack one by one without violating the constraints. This greedy-like procedure makes sure that the profit can be acquired as much as possible based on the pseudo-utility ratio. The DROP and ADD operators are implemented in Algorithm 1. The function $feasible(x)$ judges whether solution vector x satisfies all the constraints. It returns true if x is feasible, otherwise, it returns false. This repair method not only makes the solution feasible without violating any constraints but also packs items into knapsack with profits as much as possible.

3.3. Elite local Search

In BAAA, the best algal colony is obtained in each iteration which represents current optimal solution x^b. In order to further improve the quality of the solution x^b, an greedy local search method is adopted to exploit the neighborhood of the current best solution called *EliteLocalSearch*. The main idea of *EliteLocalSearch* is to remove an item from the knapsack and put another outside item into the knapsack for every possible pairwise items. As far as x^b is concerned, each pairwise element which contains distinct value 0 or 1 is interchanged for a higher profit. Providing that new achieved vector is a feasible solution and has better fitness value than the previous one through swap operation, then new vector will substitute for old one. This swap operation continues until all pairwise positions are examined. The algorithm is outlined as Algorithm 2 and an experiment is implemented to verify the effectiveness of this method in Section 4.

3.4. Flowchart and pseudo code of BAAA

The flowchart of BAAA is illustrated in Fig. 4. As can be seen in the flowchart, each algal colony has certain energy. How far the algal colony moves or how many times it moves in one generation (iteration) is determined by its energy. Along with the iteration, energy of each algal colony is updated in proportion to the size of algal colony S_i and transformed into a value between 0 and 1. The purpose of transformation is to make the energy values comparable and easy to handle in a controlled scope. Each movement of algal colony consumes some energy. Under the drive of energy, algal colony moves several times to a new position and achieves a new size until the energy is exhausted. After all algal colonies use up their energy, the helical movement ends and is followed by the evolutionary process and adaptation. This process is described in Algorithm 3 with details. In Algorithm 3, there are three loops. The outer loop controls the times of iteration, while the middle loop deals with each algal colony of population and the inner loop is the energy loop which controls the movement of algal colony until its energy is used up. Each movement consumes $eloss$ or $eloss/2$ energy which depends on whether this movement achieves better result.

4. Experimental study

In order to verify the effectiveness and robustness of the proposed BAAA algorithm for optimization problems, BAAA is evaluated on the well-known MKP benchmarks which come from the OR-Library[1]. The benchmark datasets are divided into two groups: low-dimensional knapsack problems and high-dimensional knapsack problems. The first group totally has 54 instances including “Sento”, “Hp”, “Pb”, “Pet”, “Weing” and “Weish”, in which the number of decision variables (d) ranges from 10 to 105 and the number of constraints (m) ranges from 2 to 30. The second group covers 10 medium-scaled problems and 30 large-scaled problems with 500 items and 5 constraints. Among the latter 30 instances, three tightness ratios exist

---

[1] OR-Library (Download on 2015-7-6):http://www.brunel.ac.uk/∼mastjjb/jeb/orlib/mknapinfo.html

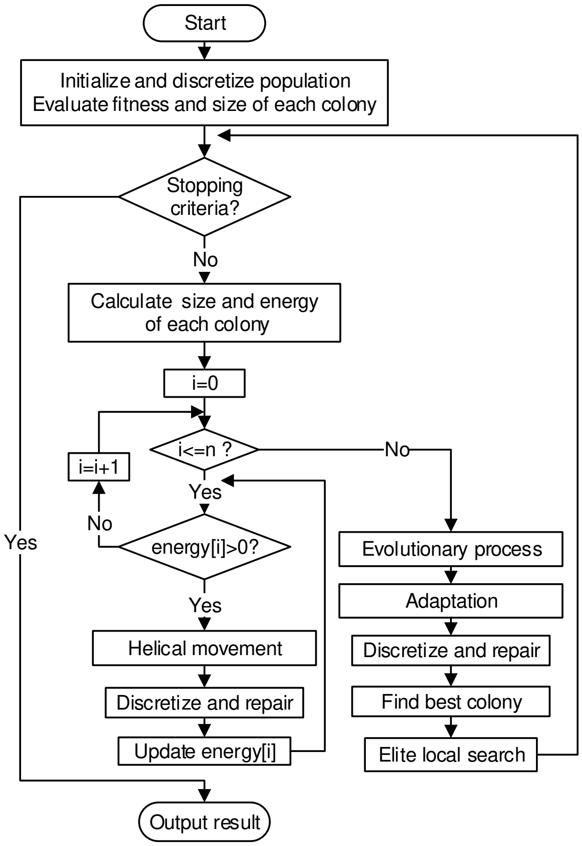

Figure 4: The flowchart of BAAA.

which are 0.25, 0.50 and 0.75, respectively. For the sake of clarity, the instances are named as cb.m.d-s_n, where m is the number of constraints, d is the number of items, s is the tightness ratio and n is the index of instances. The control parameters in BAAA are predefined for all runs. The shear force sf is set as 2, energy loss $eloss$ is 0.3, and the adaptation probability A_p is 0.5. The size of population is experience-based which is set as 100. In fact, too small size decreases the diversity of population, while too big size increases the computation complexity and leads to memory overflow. As can been seen in Algorithm 3, the parameter T_{max} controls the maximum number of iterations. Based on our extensive numerical experience, T_{max} is set to be 35000. However, it does not mean that the algorithm iterates so many times. The algorithm terminates in many other situations. Firstly, in the inner loop t increases itself as algal colony moves until its energy is used up or iteration variable t reaches T_{max}. Secondly, since the optimal solutions Opt are available, the algorithm terminates once the Opt has been obtained.

The proposed algorithm is implemented in C++ within Microsoft Visual Studio 2010 using a PC with Intel Core (TM) 2 Duad CPU Q9300 @2.5 GHz, 4 GB RAM and 64-bit Windows 7 operating system. The point-estimator of digits is studied in [37]. Here we will use standard truncation method to report numerical results. If the error between the true optimal and that of obtained by our algorithm is less than 10^{-8}, we say that our algorithm has successfully found the solution.

As mentioned above, the selection of τ is a key step for the balance of search ability between exploitation and exploration. To clarify the influence of τ on BAAA, a comparison test is implemented using different τ on the instance Sento1 which has 60 items and 30 constraints. The comparison results are depicted in Figs. 5-7. In the experiment, ten different τ between 0.1 and 3.5 are used in the algorithm for 30 independent runs. BAAA with $Tanh(x)$ and $Sig(x)$ are named as BAAA-Tanh and BAAA-Sig, respectively. The comparison is performed based on three performance measures: average iteration number (AIT), average fitness value (AVG), and success rate (SR). AIT reflects the speed of finding optimal solution. It is worth to mention that AIT only indicates the number of running the outer loop in BAAA. SR indicates the ratio of the number of finding the optimal solution and the total running times (30). From Figs. 5-7, we can observe that based on the function $Tanh(x)$, BAAA obtains best result when τ is 1.5 in terms of AIT, AVG and SR. As far as function $Sig(x)$ is concerned, best results are obtained when τ is 2. The comparison results confirm that too small or too large values of τ can downgrade the performance of algorithm. Fig. 5, Fig. 6 and Fig. 7 depict the

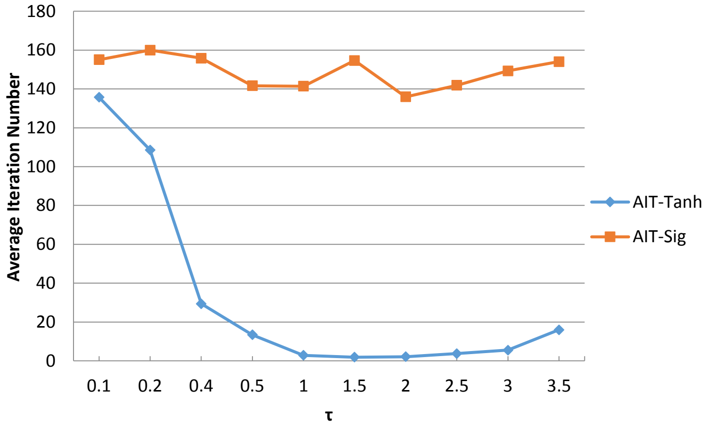

Figure 5: Comparison of AIT of $Tanh(x)$ and $Sig(x)$ on Sento1.

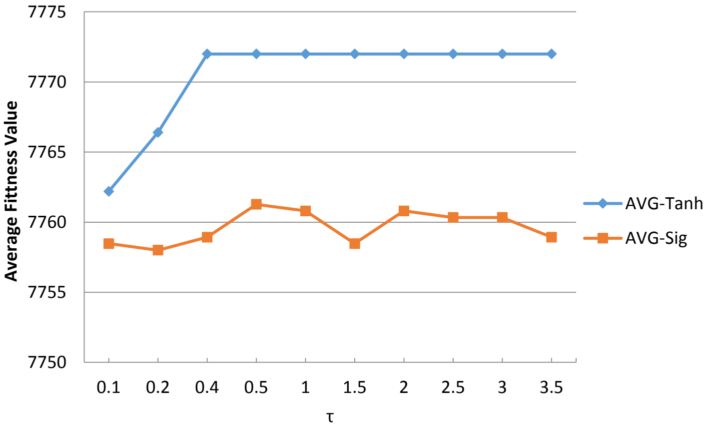

Figure 6: Comparison of AVG of $Tanh(x)$ and $Sig(x)$ on Sento1.

variations of AIT, AVG and SR in terms of τ, respectively. Based on these observations, we set τ as 1.5 and 2 for BAAA-Tanh and BAAA-Sig, respectively, in the following experiments.

Moreover, it is clear that BAAA-Tanh performs much better than BAAA-Sig in all respects. The success rate of BAAA-Tanh almost reaches 100%, except for the two smallest values of τ, whereas BAAA-Sig cannot achieve 100% success rate no matter what τ is. For further analysis, more comprehensive and complex comparisons between BAAA-Tanh and BAAA-Sig are implemented on more datasets which include 24 instances. The results are illustrated in Table 1. Through running 30 times of two algorithms on each instance, and we can observe that BAAA-Tanh outperforms BAAA-Sig. BAAA-Tanh obtains optimal solutions in 18 instances out of 24 instances with 100% success rate, whereas BAAA-Sig fails to achieve 100% success rate in 9 instances. In addition, SR of BAAA-Tanh is much higher than that of BAAA-Sig even if it can not reach 100%, and BAAA-Sig can not succeed in finding optimal solution at all in "Pet6" instance. The responding AVG prefers BAAA-Tanh

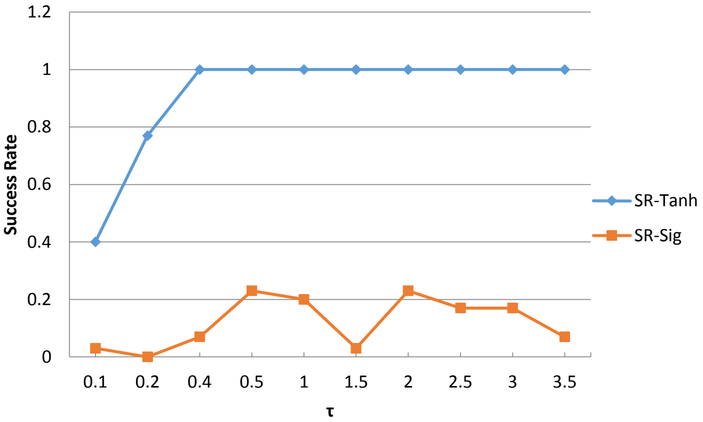

Figure 7: Comparison of SR of $Tanh(x)$ and $Sig(x)$ on Sento1.

in the same way, since BAAA-Tanh obtains higher average fitness values than BAAA-Sig. According to the comparison results, $Tanh(x)$ is applied in BAAA for further tests.

In BAAA, repair operators play a significant role in improving the maximal profit of the knapsack. The DROP and ADD operators utilize the ranked pseudo-utility ratios to discard and receive items. Eqs. (22-24) present three pseudo-utility ratios: $\overline{\delta_i}$, $\widetilde{\delta_i}$ and δ_i, i.e. profit/weight utility, relative profit density and relative mean resource occupation. In order to verify the effects of the three pseudo-utility ratios on the algorithm, an experiment is conducted and the results are depicted in Figs. 8-11. Standard deviation (SD) and SR are considered to measure the performance of algorithm with different pseudo-utility ratios. The tests are based on 54 instances and each instance is solved by 30 times. The instances from weish1 to weish17 are left out in Fig. (11) where all runs are able to find optimal solutions at 100% success rate. From these figures, it is difficult to confirm which one is more appropriate than others. In terms of SR, $\overline{\delta_i}$ fails to find optimal solutions at 100% success rate for 11 instances, while $\widetilde{\delta_i}$ and δ_i are 8 and 6, respectively. It seems that δ_i performs better, but its success rates are 0 for “Pet6” and “Pet7” and the success rates are very low only about 0.1 for “Hp2”, “Pb2” and “Weing7”. As far as SD is concerned, $\widetilde{\delta_i}$ obtains less SD than $\overline{\delta_i}$ and δ_i for “Hp1”, ”Pet6”and “Pet7”. However, in other cases it is not true. In general, $\widetilde{\delta_i}$ and δ_i outperform $\overline{\delta_i}$, and each has its own strong point. We adopt relative profit density in BAAA to compare with other swarm-based algorithms.

Elite local search is a greedy local search method which can improve the solution quality significantly. However, it may take more computational cost for its greedy character to search better neighbors. In order to gain insight into its effect on the algorithm, a comparison experiment is implemented on 10 hard problems which have 100 items and 10 constraints.

Table 1: Comparative results of Tanh(x) and Sig(x)

Problems	d×m	Opt	BAAA-Tanh		BAAA-Sig	
			SR	AVG	SR	AVG
Sento1	60×30	7772	**1**	**7772**	0.3	7762.2
Sento2	60×30	8722	**1**	**8722**	0.7	8721.7
Hp1	28×4	3418	**0.8**	**3415.2**	0.6	3412.4
Hp2	35×4	3186	**0.27**	**3161.1**	0.13	3160.6
Pet2	10×10	87061	1	87061	1	87061
Pet3	15×10	4015	1	4015	1	4015
Pet4	20×10	6120	1	6120	1	6120
Pet5	28×10	12400	1	12400	1	12400
Pet6	39×5	10618	**0.3**	**10598.8**	0	10597
Pet7	50×5	16537	**0.8**	**16531.9**	0.1	16492.4
Pb1	27×4	3090	1	3090	1	3090
Pb2	34×4	3186	**1**	**3186**	0.3	3170.1
Pb4	29×2	95168	1	95168	1	95168
Pb5	20×10	2139	1	2139	1	2139
Pb6	40×30	776	1	776	1	776
Pb7	37×30	1035	1	1035	1	1035
Weing1	28×2	141278	1	141278	1	141278
Weing2	28×2	130883	1	130883	1	130883
Weing3	28×2	95677	1	95677	1	95677
Weing4	28×2	119337	1	119337	1	119337
Weing5	28×2	98796	1	98796	1	98796
Weing6	28×2	130623	1	130623	1	130623
Weing7	105×2	1095445	**0.6**	**1095419.75**	0.1	1095388.25
Weing8	105×2	624319	**0.93**	**624178.7**	0.2	623459

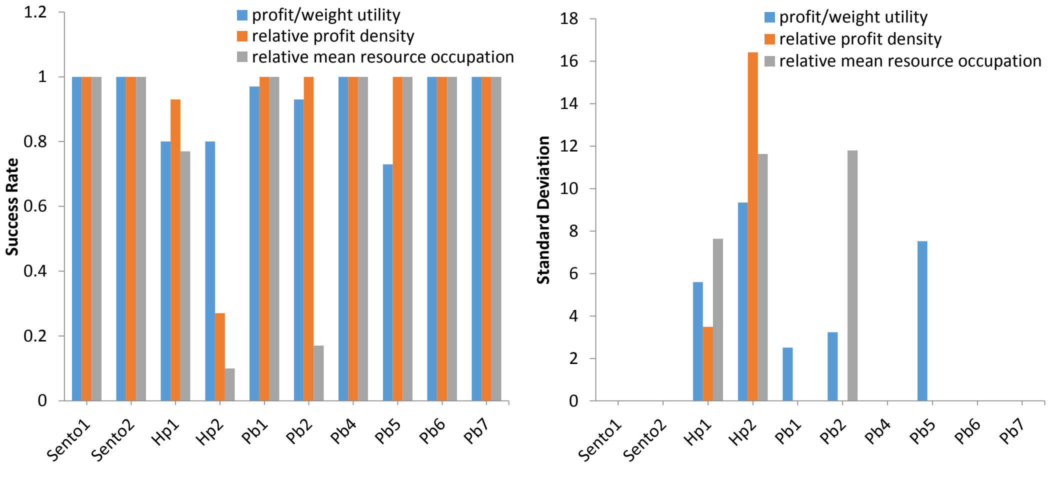

Figure 8: Comparison of SR and SD with three pseudo-utility ratios.

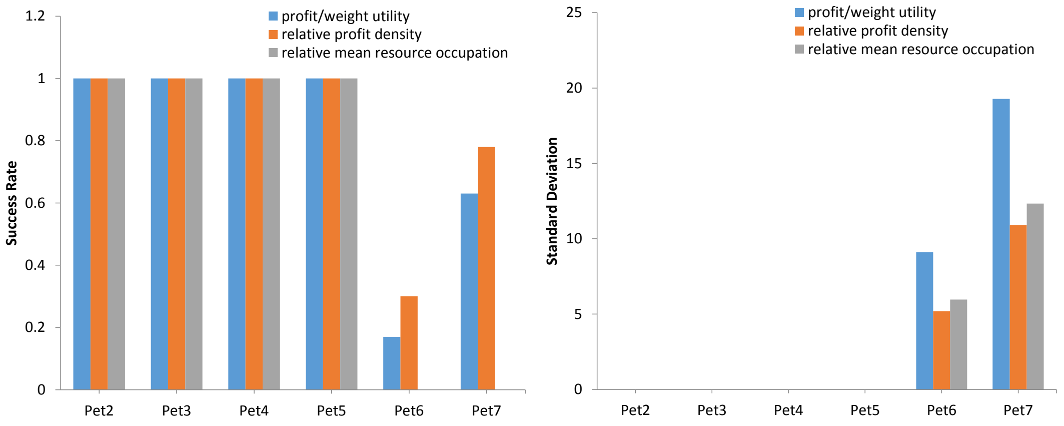

Figure 9: Comparison of SR and SD with three pseudo-utility ratios.

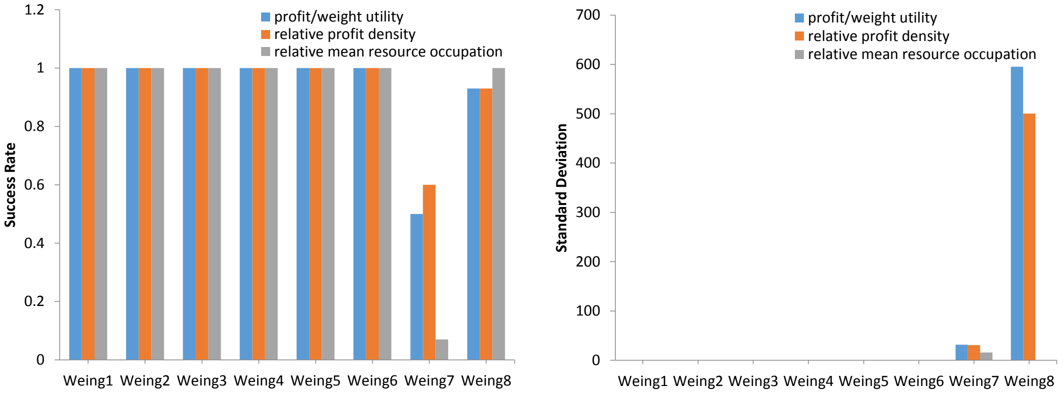

Figure 10: Comparison of SR and SD with three pseudo-utility ratios.

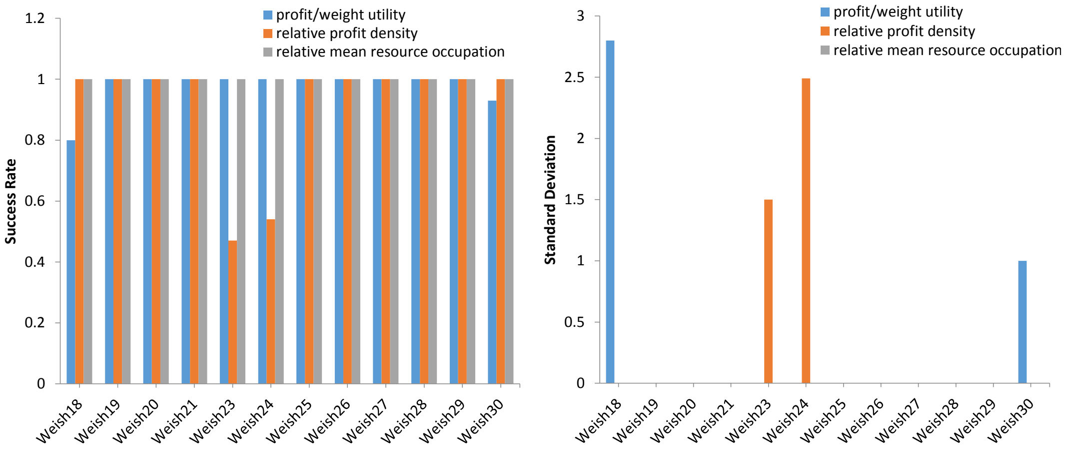

Figure 11: Comparison of SR and SD with three pseudo-utility ratios.

Considering elite local search is a built-in feature of BAAA, BAAA without elite local search is named as BAAA-noelite. The Comparative results based on 100 independent runs are shown in Table 2. SR denotes the ratio of the running times reaching the best-known value of 100 runs. AT is the average computational time (in seconds). It is quite clear that BAAA obtains better AVG and higher SR than BAAA-noelite. However, AT denotes BAAA costs more computational time than BAAA-noelite, because extra computation is needed to complete elite local search.

Table 2: Comparative results of BAAA and BAAA-noelite

Problems	Best known	BAAA			BAAA-noelite		
		AVG	SR	AT	AVG	SR	AT
10.100.00	23064	**23043.28**	**0**	17.499	22859.55	0	**4.085**
10.100.01	22801	**22750.15**	**0.30**	17.023	22659.25	0.25	**3.471**
10.100.02	22131	**22091.14**	**0.12**	13.483	21928.10	0.02	**4.726**
10.100.03	22772	**22645.65**	**0.06**	17.809	22433.55	0.01	**4.664**
10.100.04	22751	**22635.30**	**0.03**	14.178	22408.25	0	**4.228**
10.100.05	22777	**22710.95**	**0**	17.412	22405.90	0	**4.917**
10.100.06	21875	**21822.20**	**0.25**	13.073	21742.50	0.10	**4.052**
10.100.07	22635	**22530.65**	**0.16**	17.993	22350.30	0.01	**5.368**
10.100.08	22511	**22412.88**	**0.01**	19.156	22316.20	0	**4.746**
10.100.09	22702	**22650.50**	**0.45**	15.581	22569.05	0.35	**3.823**

In order to verify the superiority of the algorithm, BAAA is further compared with other population-based algorithms, including the modified binary particle swarm optimization algorithnm (MBPSO [20]), particle swarm optimization with time-varying acceleration coefficients (BPSOTVAC and CBPSOTVAC [29]), genetic algorithms with double strings (GADS [16]), binary artificial fish swarm algorithm (bAFSA [25]) and improved binary artificial fish swarm algorithm (IbAFSA [24]). Table 3 summarizes the comparison among MBPSO, BPSOTVAC, CBPSOTVAC and BAAA based on four different performance criteria, namely, SR, average error (AE), mean absolute deviation (MAD) and SD. AE is calculated as the average of the difference between the values and corresponding optimum solutions. Whereas MAD is the average of the absolute difference between the values and their mean. The data of MBPSO, BPSOTVAC and CBPSOTVAC are collected from original literatures. For the sake of consistency, 100 independent runs of BAAA are carried out for 48 instances. The experimental results show that BAAA performs much better than other three algorithms in terms of SR except for “Hp2”, “Weish23” and “Weish24”. It is worth mentioning that BAAA finds optimal solutions for all the instances and succeeds at 100% success rate for 42 instances. AE, MAD and SD are the measures to evaluate the stability of the algorithms from different angles. Based on the observation from Table 3, most values of AE, MAD and SD obtained by BAAA are less than corresponding values obtained by other three algorithms. In general, BAAA is superior to MBPSO, BPSOTVAC and CBPSOTVAC in terms of effectiveness and robustness.

Table 3: Comparative results of BAAA with MBPSO, BPSOTVAC, and CBPSOTVAC.

Problems	MBPSO			BPSOTVAC			CBPSOTVAC			BAAA			
	SR	AE	SD	SR	MAD	SD	SR	MAD	SD	SR	AE	MAD	SD
Sento1	0.52	9.96	15.1195	0.57	8.74	11.52	0.39	136.28	357.78	**1**	**0**	**0**	**0**
Sento2	0.44	5.4	6.6333	0.27	9.42	7.04	0.2	53.53	101.03	**1**	**0**	**0**	**0**
Hp1	0.45	10.85	12.0982	0.38	11.44	10.69	0.29	14.1	13.69	**0.93**	**0.93**	**1.74**	**3.49**
Hp2	0.65	**7.27**	**11.7217**	**0.67**	**6.51**	13.95	0.59	12.39	21.35	0.27	29.88	10.39	13.2
Pb1	0.40	102.86	108.55	0.46	9	9.44	0.4	10.26	10.52	**1**	**0**	**0**	**0**
Pb2	0.36	22	22.1418	0.73	4.5	7.68	0.51	14.45	18.73	**1**	**0**	**0**	**0**
Pb4	0.59	8.95	14.0224	0.91	228.1	797.1	0.84	304.33	875.1	**1**	**0**	**0**	**0**
Pb5	0.44	5.19	5.8969	0.84	2.72	6.26	0.8	3.4	6.83	**1**	**0**	**0**	**0**
Pb6	0.48	10.96	13.5033	0.5	8.7	9.99	0.54	17.74	40.17	**1**	**0**	**0**	**0**
Pb7	0.58	10.51	16.9555	0.47	5.43	5.71	0.4	13.05	24.25	**1**	**0**	**0**	**0**
Weing1	1	0	0	1	0	0	0.92	51.25	281.98	**1**	**0**	**0**	**0**
Weing2	0.99	1.6	15.9198	1	0	0	0.88	123.19	545.5	**1**	**0**	**0**	**0**
Weing3	0.37	347.86	373.721	0.92	6.42	25.53	0.75	173.07	672.42	**1**	**0**	**0**	**0**
Weing4	0.99	27.15	270.139	1	0	0	0.97	42.83	378.58	**1**	**0**	**0**	**0**
Weing5	0.86	384.4	1131.66	1	0	0	0.94	85.62	572.82	**1**	**0**	**0**	**0**
Weing6	0.74	101.4	171.067	0.97	11.7	66.86	0.87	91.71	343.45	**1**	**0**	**0**	**0**
Weing7	0.41	38.33	33.9594	0	281.23	383.74	0	11272.9	30020	**0.58**	**32.76**	**31.45**	**31.48**
Weing8	0.89	**0.11**	**0.3129**	0.35	1872.44	2000.9	0.20	27128.4	75169	**0.93**	133.46	**239.91**	500.4
Weish1	1	0	0	1	0	0	0.94	5.45	32.81	**1**	**0**	**0**	**0**
Weish2	0.80	1	2	0.64	1.8	2.41	0.66	4.12	23.12	**1**	**0**	**0**	**0**
Weish3	0.98	0.72	6.3231	0.99	0.63	6.3	0.95	9.21	52.69	**1**	**0**	**0**	**0**
Weish4	1	0	0	1	0	0	0.99	8.59	85.9	**1**	**0**	**0**	**0**
Weish5	1	0	0	1	0	0	0.98	8.11	74.45	**1**	**0**	**0**	**0**
Weish6	0.80	3.25	6.5869	0.59	6.68	8.19	0.53	23.21	79.28	**1**	**0**	**0**	**0**
Weish7	0.99	0.18	1.791	0.96	0.7	3.45	0.78	19.17	71.95	**1**	**0**	**0**	**0**
Weish8	0.95	0.1	0.4359	0.79	0.42	0.82	0.68	8.84	42.81	**1**	**0**	**0**	**0**
Weish9	1	0	0	1	0	0	0.85	13.01	65.7	**1**	**0**	**0**	**0**
Weish10	0.98	0.81	5.9828	0.91	1.43	9.56	0.67	57.16	188.63	**1**	**0**	**0**	**0**
Weish11	0.41	41.337	200.864	0.88	7.42	25.72	0.62	110.85	403.03	**1**	**0**	**0**	**0**
Weish12	0.99	0.01	0.0995	0.89	0.29	1.91	0.71	107.5	304.43	**1**	**0**	**0**	**0**
Weish13	0.95	0.7917	7.7162	1	0	0	0.85	38.62	180.04	**1**	**0**	**0**	**0**
Weish14	0.88	2.2842	8.0989	0.98	0.62	4.36	0.79	116.23	364.66	**1**	**0**	**0**	**0**
Weish15	0.97	1.29	7.8145	1	0	0	0.8	161.45	554.35	**1**	**0**	**0**	**0**
Weish16	0.91	0.9	7.3668	0.54	1.16	1.71	0.43	143.29	367.29	**1**	**0**	**0**	**0**
Weish17	1	0	0	1	0	0	0.72	85.29	227.16	**1**	**0**	**0**	**0**
Weish18	0.85	1.78	5.285	0.75	2.79	5.25	0.53	99.14	275.53	**1**	**0**	**0**	**0**

(Continued on next page)

(Continued Table 3)

Problems	MBPSO			BPSOTVAC			CBPSOTVAC			BAAA			
	SR	AE	SD	SR	MAD	SD	SR	MAD	SD	SR	AE	MAD	SD
Weish19	0.51	13.568	22.9474	0.65	4.9	7.13	0.62	169.45	489.37	**1**	**0**	**0**	**0**
Weish20	0.96	0.86	5.284	0.78	3.78	7.53	0.69	117.89	410.74	**1**	**0**	**0**	**0**
Weish21	0.77	8.0851	17.6838	0.74	6.06	10.41	0.67	125.78	378.38	**1**	**0**	**0**	**0**
Weish22	0.45	12.071	17.1277	0.16	15.12	6.63	0.17	172.8	486.71	**1**	**0**	**0**	**0**
Weish23	0.10	25.052	42.3526	**0.85**	**1.11**	5.11	0.58	179	437.23	0.45	**1.74**	1.46	**1.48**
Weish24	**0.90**	**0.5**	**1.5**	0.7	3.04	6.44	0.55	113.72	295.79	0.54	2.3	**2.48**	2.49
Weish25	0.52	7.84	8.2894	0.49	4.54	7.09	0.32	112.43	361.88	**1**	**0**	**0**	**0**
Weish26	0	587.49	27.567	0.36	11.44	12.81	0.28	270.13	710.77	**1**	**0**	**0**	**0**
Weish27	0.77	20.337	90.701	0.99	0.39	3.9	0.83	211.46	640.43	**1**	**0**	**0**	**0**
Weish28	0.10	149	140	0.87	2.99	7.77	0.62	368.74	887.33	**1**	**0**	**0**	**0**
Weish29	0	586	0	0.86	3.19	10.09	0.48	384.5	854.5	**1**	**0**	**0**	**0**
Weish30	0.72	1.73	4.7241	0.87	0.52	1.35	0.63	203.79	491.81	**1**	**0**	**0**	**0**

The comparison with other bio-inspired algorithms are further carried out. Table 4 indicates the experimental results of GADS, IbAFSA and BAAA in terms of AIT, AIT*, Nopt, AT and ASR. AIT is the average iteration number, and AIT* is the average iteration number only considering successful runs. Nopt is the number of instances which optimal solutions are found at least one time from 30 runs. AT is the average computational time (in seconds). ASR is the average of the success rate (in %) of all instances in one set. For a fair comparison, we run BAAA 30 times independently like other two algorithms. As far as AIT and AIT* are concerned, the iteration times of our proposed BAAA are smaller than those of GADS and IbAFSA. However, BAAA is not always superior to other algorithms in AT because of the different computational complexity of each iteration in different algorithms. Considering Nopt, except for GADS, they are able to solve all instances to optimality at least one time out of 30 runs. Meanwhile, the ASR of BAAA is greater than or equal to those of other algorithms in “Pb”, “Pet”, “Sento” and “Weing”.

Table 4: Comparative results of BAAA with GADS and IbAFSA.

Problem	GADS					IbAFSA					BAAA				
sets	AIT	AIT*	Nopt	AT	ASR	AIT	AIT*	Nopt	AT	ASR	AIT	AIT*	Nopt	AT[a]	ASR
Hp	399	235	2	**0.22**	76.67	189	176	2	0.40	**98.33**	**107.15**	**70.22**	**2**	0.57	58
Pb	352	183	6	0.25	78.33	77	77	6	**0.17**	100.00	**22.18**	**22.18**	**6**	0.21	**100**
Pet	335	70	5	**0.24**	71.43	262	123	7	0.83	76.19	**49.01**	**36.23**	**7**	0.53	**84.6**
Sento	1959	1379	1	3.03	6.67	43	43	2	**0.28**	100.00	**5.05**	**5.05**	**2**	1.03	**100**
Weing	665	184	6	0.76	70.33	543	266	8	3.11	78.75	**24.57**	**18.41**	**8**	**0.15**	**92.13**
Weish	1312	493	17	1.38	33.33	109	89	30	**0.56**	**98.44**	**9.38**	**4.57**	**30**	0.85	95.66

[a] AT is not comparable due to different CPU, operation system and programming language.

In order to verify the stability of our algorithm, BAAA is compared with HHS [23], ABHS [31] and NGHS [32] in terms of AVG, Min.Dev, Ave.Dev and Var.Dev. Min.Dev is the minimum percentage deviations from best-known values. Ave.Dev denotes the average percentage deviations from best-known values. Var.Dev represents the variance of the deviations. The experiment is based on a medium-scaled instances which have 100 items and 10 constraints. For consistency with other algorithms, the algorithm is run 20 times independently for each instance. The comparative results are shown in Table 5. From Table 5, we can confirm BAAA is stable in obtaining acceptable solutions because BAAA can achieve minimal Min.Dev, Ave.Dev and Var.Dev, although AVG of BAAA is sometimes inferior to that of HHS.

To further reveal the performance of BAAA, we test BAAA on large-scaled problems which have 500 items and 5 constraints with different tightness ratios. The simulation results are compared with those of state-of-the-art algorithms: SACRO-BPSO-TVAC and SACRO-CBPSO-TVAC [30]. This is because [30] is published in the recent and the method in [30] shows its superior to many existing algorithms. Table 6 summarizes the comparative results based on 30 independent runs. We can observe from the results that BAAA performs better than SACRO-BPSO-TVAC and SACRO-CBPSO-TVAC in terms of best obtained value (BEST) in 23 out of 30 instances. BAAA performs worse than SACRO-BPSO-TVAC or SACRO-CBPSO-TVAC in 6 instances in terms of BEST, and the results of instance 'cb.5.500-0.50_5' are not available in the reference [30] which are denoted as '-'. With respect to AVG and SD, BAAA outperforms SACRO-BPSO-TVAC and SACRO-CBPSO-TVAC clearly. In summary, in contrast to other algorithms, BAAA is more robust and competitive in low-dimensional problems as well as high-dimensional problems.

Table 6: Comparative results of BAAA with SACRO-BPSO-TVAC and SACRO-CBPSO-TVAC.

Problems		Optimal	SACRO-BPSO-TVAC	SACRO-CBPSO-TVAC	BAAA
cb.5.500-0.25_1	BEST	120148	119867	120009	**120066**
	AVG		119725.8	119761.9	**120013.66**
	SD		119.61	114.51	**21.57**
cb.5.500-0.25_2	BEST	117879	117681	117699	**117702**
	AVG		117470.8	117512.1	**117560.47**
	SD		146.32	115.72	**111.4**
cb.5.500-0.25_3	BEST	121131	120951	120923	**120951**
	AVG		120759.7	120741.2	**120782.87**
	SD		102.67	111.11	**87.96**
cb.5.500-0.25_4	BEST	120804	120450	120563	**120572**
	AVG		120282.5	120284.2	**120340.57**

(Continued on next page)

(Continued Table 6)

Problems		Optimal	SACRO-BPSO-TVAC	SACRO-CBPSO-TVAC	BAAA
	SD		**100.74**	119.82	106.01
cb.5.500-0.25_5	BEST	122319	122037	122054	**122231**
	AVG		121908.1	121922.9	**122101.84**
	SD		82.73	67.86	**56.95**
cb.5.500-0.25_6	BEST	122024	121918	121901	**121957**
	AVG		121691.5	121690	**121741.84**
	SD		103.44	104.34	**84.33**
cb.5.500-0.25_7	BEST	119127	118771	118846	**119070**
	AVG		118528.5	118530.7	**118913.37**
	SD		130.12	109.38	**63.01**
cb.5.500-0.25_8	BEST	120568	120364	120376	**120472**
	AVG		120136.6	120147.6	**120331.23**
	SD		150.23	146.64	**69.09**
cb.5.500-0.25_9	BEST	121586	**121201**	121185	121052
	AVG		120926.3	**120933.6**	120683.60
	SD		**114.39**	120.72	834.88
cb.5.500-0.25_10	BEST	120717	120471	120453	**120499**
	AVG		120285	120276.6	**120296.30**
	SD		102.94	**81.74**	110.06
cb.5.500-0.50_1	BEST	218428	**218291**	218269	218185
	AVG		**218136.9**	218116.6	217984.67
	SD		**116.41**	141.28	123.94
cb.5.500-0.50_2	BEST	221202	**221025**	221007	220852
	AVG		**220795.2**	220786.7	220527.53
	SD		**115.93**	181.32	169.16
cb.5.500-0.50_3	BEST	217542	217337	**217398**	217258
	AVG		217125.2	**217172.8**	217056.7
	SD		151.13	166.07	**104.95**
cb.5.500-0.50_4	BEST	223560	223429	223450	**223510**
	AVG		223232.4	223265.1	**223450.94**
	SD		118.43	137.67	**26.02**
cb.5.500-0.50_5	BEST	-	-	-	218811
	AVG		-	-	218634.27
	Std		-	-	97.52
cb.5.500-0.50_6	BEST	220530	220337	220428	**220429**
	AVG		220045.6	220052.1	**220375.86**
	SD		226.15	230.24	**31.86**
cb.5.500-0.50_7	BEST	219989	219686	219734	**219785**
	AVG		219407.3	219524.5	**219619.27**
	SD		204.01	192.09	**93.01**
cb.5.500-0.50_8	BEST	218215	218094	**218096**	218032
	AVG		217930.6	**217980.8**	217813.20
	SD		72.61	**56.6**	115.37
cb.5.500-0.50_9	BEST	216976	216785	216851	**216940**

(Continued on next page)

(Continued Table 6)

Problems		Optimal	SACRO-BPSO-TVAC	SACRO-CBPSO-TVAC	BAAA
	AVG		216595	216586.1	**216862.03**
	SD		143.86	192.49	**32.51**
cb.5.500-0.50_10	BEST	219719	219561	219549	**219602**
	AVG		219404.2	**219438.5**	219435.14
	SD		77.03	55.51	**54.45**
cb.5.500-0.75_1	BEST	295828	295346	295309	**295652**
	AVG		294980.4	295026.4	**295505.00**
	Std		140.29	147.36	**76.30**
cb.5.500-0.75_2	BEST	308086	307666	**307808**	307783
	AVG		307421	307461.1	**307577.50**
	SD		145.05	**120.78**	135.94
cb.5.500-0.75_3	BEST	299796	299292	299393	**299727**
	AVG		299053.2	299069	**299664.09**
	SD		144.29	145.76	**28.81**
cb.5.500-0.75_4	BEST	306480	305915	305992	**306469**
	AVG		305692.6	305680.2	**306385.00**
	SD		147.27	145.85	**31.64**
cb.5.500-0.75_5	BEST	300342	299810	299947	**300240**
	AVG		299662.7	299769.5	**300136.66**
	SD		104.49	99.74	**51.84**
cb.5.500-0.75_6	BEST	302571	302132	302156	**302492**
	AVG		301926.1	301959.6	**302376**
	SD		105.84	115.18	**53.94**
cb.5.500-0.75_7	BEST	301339	300905	300854	**301272**
	AVG		300586.3	300575.9	**301158**
	SD		150.19	144.78	**44.3**
cb.5.500-0.75_8	BEST	306454	306132	306069	**306290**
	AVG		305878.7	305922.4	**306138.41**
	SD		164.62	97.26	**84.56**
cb.5.500-0.75_9	BEST	302828	302436	302447	**302769**
	AVG		302182.8	302188.1	**302690.06**
	SD		130.53	157.72	**34.11**
cb.5.500-0.75_10	BEST	299910	299456	299558	**299757**
	AVG		299205.5	299207.5	**299702.28**
	SD		165.58	149.91	**31.66**

Furthermore, a non-parametric test, Wilcoxon signed-rank test (W-test) is carried out to determine whether the results from BAAA and those from other algorithms have significant difference or not. Table 7 shows the Wilcoxon signed-rank test results on AVG of BAAA against other algorithms, including ABHS, NGHS, HHS, SACRO-BPSO-TVAC and SACRO-CBPSO-TVAC. R- or R+ is the sum of ranks based on the absolute value of the difference between sample data from two algorithms. R- indicates the sum of the ranks corresponding

Table 5: Comparative results of BAAA with ABHS, NGHS and HHS.

Problems	Best known	Algorithms	AVG	Min.Dev(%)	Ave.Dev(%)	Var.Dev(%)
10.100.00	23064	ABHS	23023.35	0.0304	0.1762	0.1625
		NGHS	22971.20	0.0607	0.4024	0.2927
		HHS	23041.00	0.0304	0.0997	0.0974
		BAAA	**23044.25**	**0.0006**	**0.0049**	**0.0027**
10.100.01	22801	ABHS	22725.00	0.2237	0.3333	0.1291
		NGHS	22711.65	0.2105	0.3919	0.2207
		HHS	22739.55	0	0.2695	0.1161
		BAAA	**22751.25**	**0**	**0.0054**	**0.0027**
10.100.02	22131	ABHS	22070.41	0	0.2738	0.1624
		NGHS	22011.50	0	0.5399	0.2066
		HHS	**22096.25**	**0**	0.1570	0.1435
		BAAA	22090.60	0.003	**0.0050**	**0.0016**
10.100.03	22772	ABHS	22719.70	**0**	0.2297	0.3042
		NGHS	22647.15	0.0395	0.5483	0.2128
		HHS	**22753.85**	0.0395	0.0797	0.0928
		BAAA	22648.55	0.0027	**0.0098**	**0.0033**
10.100.04	22751	ABHS	22625.90	**0**	0.5499	0.2137
		NGHS	22598.55	0.2373	0.6701	0.3116
		HHS	**22657.05**	0.2373	0.4129	0.1941
		BAAA	22634.00	0.0043	**0.0095**	**0.0034**
10.100.05	22777	ABHS	22628.30	0.2678	0.6529	0.1882
		NGHS	22618.05	0.2678	0.6979	0.2342
		HHS	**22717.42**	**0**	0.2616	0.1107
		BAAA	22714.75	0.007	**0.0115**	**0.0029**
10.100.06	21875	ABHS	21774.25	0.2469	0.4606	0.1777
		NGHS	21782.45	0.3200	0.4230	0.1577
		HHS	21814.90	0.1853	0.2747	0.0941
		BAAA	**21823.10**	**0**	**0.0047**	**0.0033**
10.100.07	22635	ABHS	22523.35	0.3711	0.4933	0.0745
		NGHS	22469.70	0.4109	0.7303	0.2280
		HHS	22518.70	0.3711	0.5138	0.0327
		BAAA	**22533.20**	**0.0037**	**0.0089**	**0.0025**
10.100.08	22511	ABHS	22397.35	0.3909	0.5049	0.0764
		NGHS	22369.45	0.5153	0.6288	0.1193
		HHS	**22416.75**	0.3243	0.4187	0.0557
		BAAA	22412.25	**0.0052**	**0.0071**	**0.0013**
10.100.09	22702	ABHS	22551.35	0	0.6636	0.2524
		NGHS	22496.95	0.0176	0.9032	0.2411
		HHS	22645.78	0	0.2476	0.0789
		BAAA	**22650.50**	**0**	**0.0045**	**0.0045**

to the negative difference and R+ indicates the sum of the ranks corresponding to positive difference, respectively. pValue is significant difference between the AVG values of two algorithms, which is calculated by the software SPSS statistics 22. A null hypothesis is assumed that there is no significant difference between the two samples and an alternative hypothesis is assumed that there is a significant difference between the two samples, at 0.05 significance level. According to the relationship between pValue and 0.05 significance level, we obtain the result which is represented by three signs: "+", "-" or "≈". "+" or "-" denotes the first algorithm is significantly better or worse than the second one, i.e. there is a significant difference. And "≈" denotes there is no significant difference between the two algorithms. It can be seen from Table 7 that BAAA is superior to ABHS, NGHS, SACRO-BPSO-TVAC and SACRO-CBPSO-TVACGA, and nearly equivalent to HHS.

Table 7: Wilcoxon signed-rank test results on AVG of BAAA against other algorithms.

Algorithm	Better	Equal	Worse	R-	R+	pValue	Result
BAAA to ABHS	9	0	1	8	47	0.047	+
BAAA to NGHS	10	0	0	0	55	0.005	+
BAAA to HHS	5	0	5	28	27	0.959	≈
BAAA to SACRO-BPSO-TVAC	24	0	5	23	412	0.000	+
BAAA to SACRO-CBPSO-TVAC	23	0	6	64	371	0.001	+

5. Conclusions

In this paper, a binary artificial algae algorithm is proposed for solving MKPs. Two logistic functions with different coefficients of curve are studied in discrete process. Three types of pseudo-utility ratios are presented and compared as well for repair operation so as to increase the efficiency of BAAA. In addition, an elite local search is introduced into our algorithm to improve the quality of solutions. Comparing with the existing algorithms, our algorithm is more robust and achieves better numerical performance. The comparisons of BAAA with other bio-inspired state-of-the-art algorithms available in the literatures are carried out with total of 94 benchmark problems. The numerical experiments demonstrate that BAAA is efficient and competitive comparing with the binary versions of the HS, PSO, GA and AFSA. Further research will focus on improving the model structure of AAA to decrease the computational efforts. Moreover, to extend the proposed algorithm for general purposes, BAAA must be applied in other binary test problems, especially in real applications, such as project scheduling and resource allocation.

Acknowledgements

This paper was partially supported by Australian Research Council Linkage Program LP130100451, a grant from Korean Research Foundation, Natural Science Foundation of China (Nos. 61473326 and 61471132), Natural Science Foundation of Chongqing (cstc2013jcyjA00029 and cstc2013jjB0149).

Reference

[1] B. W.-K. L. Ling, C. Y.-F. Ho, J. Cao, Q. Dai, Efficient complex-valued finite word length allpass rational iir pcls filter design via functional inequality constrained integer programming with bit plane searching technique, Mediterranean Journal of Electronics and Communications 9 (2013) 588–593.

[2] B. W.-K. L. Ling, N. Tian, C. Y.-F. Ho, W.-C. Siu, K.-L. Teo, Q. Dai, Maximally decimated paraunitary linear phase fir filter bank design via iterative svd approach, IEEE Transactions on Signal Processing 63 (2015) 466–481.

[3] B. W.-K. L. Ling, C. Y.-F. Ho, K.-L. Teo, W.-C. Siu, , J. Cao, Q. Dai, Optimal design of cosine modulated nonuniform linear phase fir filter bank via both stretching and shifting frequency response of single prototype filter, IEEE Transactions on Signal Processing 62 (2014) 2517–2530.

[4] S. R. Subramaniam, B. W.-K. L. Ling, A. Georgakis, Filtering in rotated time-frequency domains with unknown noise statistics, IEEE Transactions on Signal Processing 60 (2012) 489–493.

[5] B. W.-K. L. Ling, C. Y.-F. Ho, S. R. Subramaniam, A. Georgakis, J. Cao, Q. Dai, Optimal design of hermitian transform and vectors of both mask and window coefficients for denoising applications with both unknown noise characteristics and distortions, Signal Processing 98 (2014) 1–22.

[6] A. Fréville, The multidimensional 0–1 knapsack problem: An overview, European Journal of Operational Research 155 (2004) 1–21.

[7] D. Bertsimas, R. Demir, An approximate dynamic programming approach to multidimensional knapsack problems, Management Science 48 (2002) 550–565.

[8] J. Puchinger, G. R. Raidl, U. Pferschy, The multidimensional knapsack problem: Structure and algorithms, INFORMS Journal on Computing 22 (2010) 250–265.

[9] M. J. Varnamkhasti, Overview of the algorithms for solving the multidimensional knapsack problems, Advanced Studies in Biology 4 (2012) 37–47.

[10] S. Balev, N. Yanev, A. Fréville, R. Andonov, A dynamic programming based reduction procedure for the multidimensional 0–1 knapsack problem, European Journal of Operational Research 186 (2008) 63–76.

[11] V. C. Li, Y.-C. Liang, H.-F. Chang, Solving the multidimensional knapsack problems with generalized upper bound constraints by the adaptive memory projection method, Computers & Operations Research 39 (2012) 2111–2121.

[12] M. Vasquez, J.-K. Hao, et al., A hybrid approach for the 0-1 multidimensional knapsack problem, in: IJCAI, 2001, pp. 328–333.

[13] J. E. Gallardo, C. Cotta, A. J. Fernández, Solving the multidimensional knapsack problem using an evolutionary algorithm hybridized with branch and bound, in: Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, Springer, 2005, pp. 21–30.

[14] P. C. Chu, J. E. Beasley, A genetic algorithm for the multidimensional knapsack problem, Journal of heuristics 4 (1998) 63–86.

[15] F. Djannaty, S. Doostdar, A hybrid genetic algorithm for the multidimensional knapsack problem, International Journal of Contemporary Mathematical Sciences 3 (2008) 443–456.

[16] M. Sakawa, K. Kato, Genetic algorithms with double strings for 0–1 programming problems, European Journal of Operational Research 144 (2003) 581–597.

[17] S. Hanafi, A. Freville, An efficient tabu search approach for the 0–1 multidimensional knapsack problem, European Journal of Operational Research 106 (1998) 659–675.

[18] F. Qian, R. Ding, Simulated annealing for the 0/1 multidimensional knapsack problem, Numerical Mathematics - English Series 16 (2007) 320.

[19] F. Hembecker, H. S. Lopes, W. Godoy Jr, Particle swarm optimization for the multidimensional knapsack problem, in: Adaptive and Natural Computing Algorithms, Springer, 2007, pp. 358–365.

[20] J. C. Bansal, K. Deep, A modified binary particle swarm optimization for knapsack problems, Applied Mathematics and Computation 218 (2012) 11042–11061.

[21] A. Baykasoğlu, F. B. Ozsoydan, An improved firefly algorithm for solving dynamic multidimensional knapsack problems, Expert Systems with Applications 41 (2014) 3712–3725.

[22] X. Kong, L. Gao, H. Ouyang, S. Li, Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm, Computers & Operations Research 63 (2015) 7–22.

[23] B. Zhang, Q.-K. Pan, X.-L. Zhang, P.-Y. Duan, An effective hybrid harmony search-based algorithm for solving multidimensional knapsack problems, Applied Soft Computing 29 (2015) 288–297.

[24] M. A. K. Azad, A. M. A. Rocha, E. M. Fernandes, Improved binary artificial fish swarm algorithm for the 0–1 multidimensional knapsack problems, Swarm and Evolutionary Computation 14 (2014) 66–75.

[25] M. A. K. Azad, A. M. A. Rocha, E. M. Fernandes, Solving multidimensional 0–1 knapsack problem with an artificial fish swarm algorithm, in: Computational Science and Its Applications–ICCSA 2012, Springer, 2012, pp. 72–86.

[26] X.-S. Yang, Z. Cui, R. Xiao, A. H. Gandomi, M. Karamanoglu, Swarm intelligence and bio-inspired computation: theory and applications, Newnes, 2013.

[27] S. Binitha, S. S. Sathya, A survey of bio inspired optimization algorithms, International Journal of Soft Computing and Engineering 2 (2012) 137–151.

[28] J. Kennedy, R. C. Eberhart, A discrete binary version of the particle swarm algorithm, in: Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on, volume 5, IEEE, 1997, pp. 4104–4108.

[29] M. Chih, C.-J. Lin, M.-S. Chern, T.-Y. Ou, Particle swarm optimization with time-varying acceleration coefficients for the multidimensional knapsack problem, Applied Mathematical Modelling 38 (2014) 1338–1350.

[30] M. Chih, Self-adaptive check and repair operator-based particle swarm optimization for the multidimensional knapsack problem, Applied Soft Computing 26 (2015) 378–389.

[31] L. Wang, R. Yang, Y. Xu, Q. Niu, P. M. Pardalos, M. Fei, An improved adaptive binary harmony search algorithm, Information Sciences 232 (2013) 58–87.

[32] D. Zou, L. Gao, S. Li, J. Wu, Solving 0–1 knapsack problem by a novel global harmony search algorithm, Applied Soft Computing 11 (2011) 1556–1564.

[33] S. A. Uymaz, G. Tezel, E. Yel, Artificial algae algorithm (aaa) for nonlinear global optimization, Applied Soft Computing 31 (2015) 153–171.

[34] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization, KanGAL report 2005005 (2005).

[35] H. Pirkul, A heuristic solution procedure for the multiconstraint zero-one knapsack problem, Naval Research Logistics 34 (1987) 161–172.

[36] H. O. S. L. Xiangyong Konga, Liqun Gaoa, Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm, Computers and Operations Research 63 (2015) 7–22.

[37] W. T. Song, B. W. Schmeiser, Omitting meaningless digits in point estimates: The probability guarantee of leading-digit rules, Operations research 57 (2009) 109–117.

Algorithm 1 DROP and ADD procedure

```
Input:
    a candidate solution $x$
Output:
    a repaired solution $x$
compute $\delta_i$, i=1,2,...,d
initialize s(i)=i, i=1,2,...,d
sort s(i) rendering $\delta_{s(i)}$ be in ascending order
//DROP phase
if(not $feasible(x)$)
    for i=1 to d do
        if($x_{s(i)}$=1)
            $x_{s(i)} = 0$
            if($feasible(x)$)  break
        end if
    end for
end if
//ADD phase
for i=d to 1 do
    if($x_{s(i)}$=0)
        $x_{s(i)} = 1$
        if(not $feasible(x)$)  $x_{s(i)} = 0$
    end if
end for
return  $x$.
```

Algorithm 2 *EliteLocalSearch* procedure

Input:

a current best solution x^b

Output:

an improved solution x^b

```
for i=1 to d do
    for j=1 to d do
        if (i!=j and $x_i^b != x_j^b$)
            $x$=swap($x^b$, i, j) //exchange the ith and jth elements of the solution vector
            if ($fitness(x) > fitness(x^b)$)  $x^b = x$
        end if
    end for
end for
return  $x^b$.
```

Algorithm 3 Binary artificial algae algorithm

```
Input:
    $c$, $p$, $b$
Output:
    the maximized profit of knapsack
define $n,\ sf,\ eloss,\ A_p$
initialize population of algal colony $x_i$ and repair $x_i,\ i = 1, 2, ..., n$
$starvation_i = 0,\ i = 1, 2, ..., n$
while $(t < T_{max})$
    calculate energy $E_i$ and friction surface $\omega_i$ according to size of $x_i,\ i = 1, 2, ..., n$
    for i=1 to n do
        isstarve=true
        while $(E_i > 0\ and\ t < T_{max})$
            calculate j through tournament selection method
            choose distinct k, l, m randomly between 1 and d
            produce $\alpha, \beta, p$ randomly where $\alpha$ and $\beta$ are in the range [0,2$\pi$], $p$ is between -1 and 1
            $x_{im} = x_{im} + (x_{jm} - x_{im})(sf - \omega_i))p$
            $x_{ik} = x_{ik} + (x_{jk} - x_{ik})(sf - \omega_i)) \cos\alpha$
            $x_{il} = x_{il} + (x_{jl} - x_{il})(sf - \omega_i)) \sin\beta$
            discretize and repair $x_i$
            $E_i = E_i - eloss/2$
            if (new fitness value of $x_i$ is better than old one)
                accept $x_i$ and update corresponding fitness value
                isstarve=false
            else
                $E_i = E_i - eloss/2$
            end if
            t=t+1
        end while
        if (isstarve) $starvation_i = starvation_i + 1$
    end for
    the $r_{th}$ dimension of smallest algal colony is replaced by that of biggest one, where $r$ is selected randomly between 1 to d
    if $(A_p > rand)$
        select the most starving algal colony $x_s$, and $x_s = x_s + (biggest - x_s) * rand$
        discretize and repair $x_s$
    end if
    best=findBest($x$)
    ebest=eliteLocalSearch(best)
end while
return ebest
```