

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]:

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

where d is the number of items and m is the number of knapsack constraints; p_i is the profit of i th item if it is put into knapsack; x_i is either 1 or 0, where 1 denotes the i th item being stored into the knapsack and 0 denotes i th item being discarded, respectively; c_{ij} is the consumption of j th resource while putting the i th item into knapsack and b_j is the total capacity of j th 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],

29 firefly algorithm [21], harmony search [22, 23] and artificial fish swarm algorithm [24, 25],
30 etc. Evolutionary computation and bio-inspired algorithms are the fastest developing type of
31 algorithms. The basic idea of them is that from an initial population of individuals, solution
32 vectors, individuals evolve by some way to produce new better individuals and keep better
33 ones in the next generation(iteration), whereas the worse individuals are discarded in the next
34 generation. A satisfactory solution will be obtained after updating some generations. More
35 details can be found in [26, 27].

36 In [14], genetic algorithm was utilized to solve MKP. This method has been further im-
37 proved by Djannaty in [15] where initial population created by Dantzig algorithm and penalty
38 function to increase the rate of convergence of MKP were introduced. In [28], a binary version
39 of PSO is introduced by Kennedy to solve discrete optimization problems. In [20], a modified
40 binary particle swarm optimization (MBPSO) algorithm is proposed for 0-1 knapsack prob-
41 lem and multidimensional knapsack problem. MBPSO introduced a new probability function
42 to improve the diversity and made it more effective than simple binary version of PSO. In
43 [29], binary PSO with time-varying acceleration coefficients (BPSOTVAC) and chaotic binary
44 PSO with time-varying acceleration coefficients (CBPSOTVAC) were proposed. Through in-
45 troducing the time-varying inertia weight and time-varying learning factors, the performance
46 of the solution had been improved significantly. In [30], a particle swarm optimization with
47 self-adaptive check and repair operator (SACRO) was presented to improve the efficiency of
48 PSO, where SACRO will change the alternative pseudo-utility ratio dynamically. In [25], a
49 binary version of the artificial fish swarm algorithm was proposed where a decoding scheme
50 was introduced to transform infeasible solutions to be feasible for multidimensional knap-
51 sack problem. In [23], an effective hybrid algorithm based on harmony search (HHS) was
52 presented to solve multidimensional knapsack problems. HHS developed a novel harmony
53 improvisation mechanism with modified memory consideration rule and global-best pitch ad-
54 justment scheme. In addition, the fruit fly optimization (FFO) scheme was integrated as a
55 local search strategy. Compared with an improved adaptive binary harmony search algorithm
56 (ABHS) [31] and a novel global harmony search algorithm (NGHS) [32], HHS demonstrated
57 the effectiveness and robustness.

58 In the recent years, a new meta-heuristic algorithm, artificial algae algorithm (AAA), was
59 presented [33]. Similar to other bio-inspired algorithms, AAA was inspired by the lifestyles
60 of algae. AAA has been successfully applied in the optimization of benchmark functions with
61 various dimensions in CEC'05 [34] and implemented on the pressure vessel problem. However,

62 due to its continuous nature, AAA cannot settle the discrete problem straightforwardly such
 63 as MKP. In view of this problem, this paper proposes a binary artificial algae algorithm
 64 (BAAA) to solve MKP. Compared with many bio-inspired binary version algorithms in well-
 65 known benchmarks for MKP, BAAA achieves better performance in terms of robustness as
 66 well as the best solution obtained.

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

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

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

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

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

78 In the artificial algae algorithm, there are three key parts which are helical movement,
 79 evolutionary process and adaptation. The algal colony tries to move to a optimal position
 80 through moving, evolving and adapting itself. It is worth to mention that a crucial concept
 81 in AAA is the size of algal colony of i th algal colony denoted as S_i , $i = 1, 2, \dots, n$. Similar
 82 to the real algae, under perfect living condition, the algal colony will reproduce and grow to
 83 a bigger size. Living in a bad environment will lead to death of algal cells and shrink of algal
 84 colony. S_i is set as 1 at the initial stage, and altered with the change of the fitness value of
 85 the i th algal colony, i.e. the value of objective function. The better the objective function
 86 $f(x_i)$ is, the bigger S_i is. S_i is updated according to the biological growth process given as

87 follows:

$$S_i = size(x_i) \quad (3)$$

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

$$S_i^{t+1} = \mu_i S_i^t, \quad i = 1, 2, \dots, n \quad (5)$$

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

90 2.1. Helical movement

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

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

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

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

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

$$\omega_i = 2\pi r_i^2 \quad (9)$$

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

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

110 *2.2. Evolutionary process*

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

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

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

$$smallest_j = biggest_j, \quad j = 1, 2, \dots, d. \quad (13)$$

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

119 *2.3. Adaptation*

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

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

132

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

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

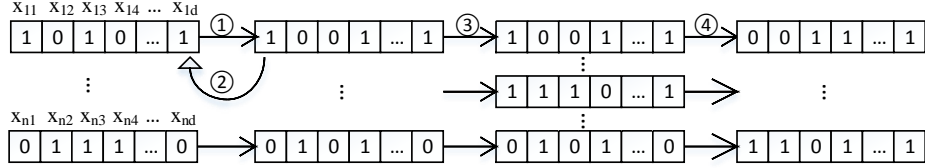


Figure 1: Encoding example of BAAA.

135 generate stochastic real-valued numbers between 0 and 1, A_p is the adaptation probability
 136 which decides whether adaptation occurs or not, A_p is a constant usually being set between
 137 0.3 and 0.7.

138 3. Binary artificial algae algorithm (BAAA)

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

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

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

152 3.1. Discrete process

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

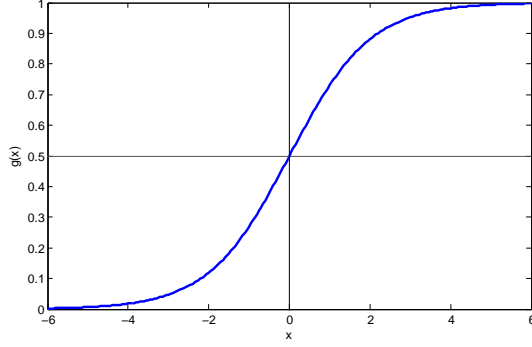


Figure 2: Sigmoid curve of logistic function.

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

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

159

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

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

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

162

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

163 where τ is a controlling parameter which determines the changing trend of the curve. Com-
 164 bined with Eq. (18), a discrete value 0 or 1 is produced through comparing $g(x)$ with a random
 165 distributed value between 0 and 1. Fig. 3 illustrates the figure of $Tanh(x)$ and $Sig(x)$ with
 166 different τ . As seen in Fig. 3, the smaller τ is, the less steepness of the curves have. When
 167 τ is very small, the curve tends to be a horizontal line. Taking $Sig(x)$ as an example, when
 168 $\tau = 0.1$, the values of function are close to 0.5 which makes the discrete procedure like a
 169 random selection. As a result, the algorithm is led to poor exploitation and easy to fall into
 170 local optimum. On the other hand, when τ is large, the curve becomes much steep which
 171 leads to low diversity and poor exploration. For example, if $x > 5$ and $\tau = 3.5$, then $g(x)$
 172 is very close to 1. For this case, Eq. (18) has little chance to produce 0. This clearly shows
 173 that proper τ is crucial for the discrete procedure. An experiment is carried out in the next
 174 section for the selection of τ .

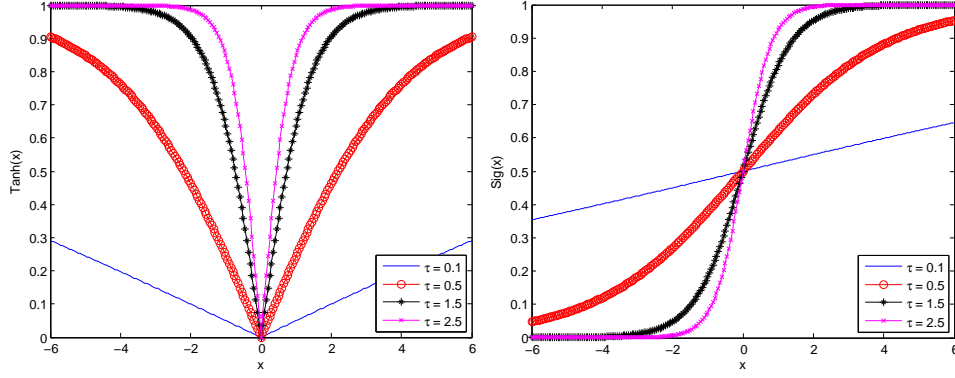


Figure 3: Comparison of $Tanh(x)$ and $Sig(x)$ with different τ .

175 3.2. Repair operator

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

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

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

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

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

$$\bar{\delta}_i = \min\left\{\frac{p_i}{c_{ij}}\right\}, \quad i = 1, 2, \dots, d, \quad j = 1, 2, \dots, m \quad (23)$$

198

$$\tilde{\delta}_i = \min\left\{\frac{p_i \cdot b_j}{c_{ij}}\right\}, \quad i = 1, 2, \dots, d, \quad j = 1, 2, \dots, m \quad (24)$$

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

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

220 3.3. Elite local Search

221 In BAAA, the best algal colony is obtained in each iteration which represents current
 222 optimal solution x^b . In order to further improve the quality of the solution x^b , an greedy
 223 local search method is adopted to exploit the neighborhood of the current best solution
 224 called *EliteLocalSearch*. The main idea of *EliteLocalSearch* is to remove an item from
 225 the knapsack and put another outside item into the knapsack for every possible pairwise

226 items. As far as x^b is concerned, each pairwise element which contains distinct value 0 or 1
227 is interchanged for a higher profit. Providing that new achieved vector is a feasible solution
228 and has better fitness value than the previous one through swap operation, then new vector
229 will substitute for old one. This swap operation continues until all pairwise positions are
230 examined. The algorithm is outlined as Algorithm 2 and an experiment is implemented to
231 verify the effectiveness of this method in Section 4.

232 3.4. Flowchart and pseudo code of BAAA

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

247 4. Experimental study

248 In order to verify the effectiveness and robustness of the proposed BAAA algorithm for
249 optimization problems, BAAA is evaluated on the well-known MKP benchmarks which come
250 from the OR-Library¹. The benchmark datasets are divided into two groups: low-dimensional
251 knapsack problems and high-dimensional knapsack problems. The first group totally has 54
252 instances including “Sento”, “Hp”, “Pb”, “Pet”, “Weing” and “Weish”, in which the number
253 of decision variables (d) ranges from 10 to 105 and the number of constraints (m) ranges from
254 2 to 30. The second group covers 10 medium-scaled problems and 30 large-scaled problems
255 with 500 items and 5 constraints. Among the latter 30 instances, three tightness ratios exist

¹OR-Library (Download on 2015-7-6):<http://www.brunel.ac.uk/~mastjjb/jeb/orlib/mknapiinfo.html>

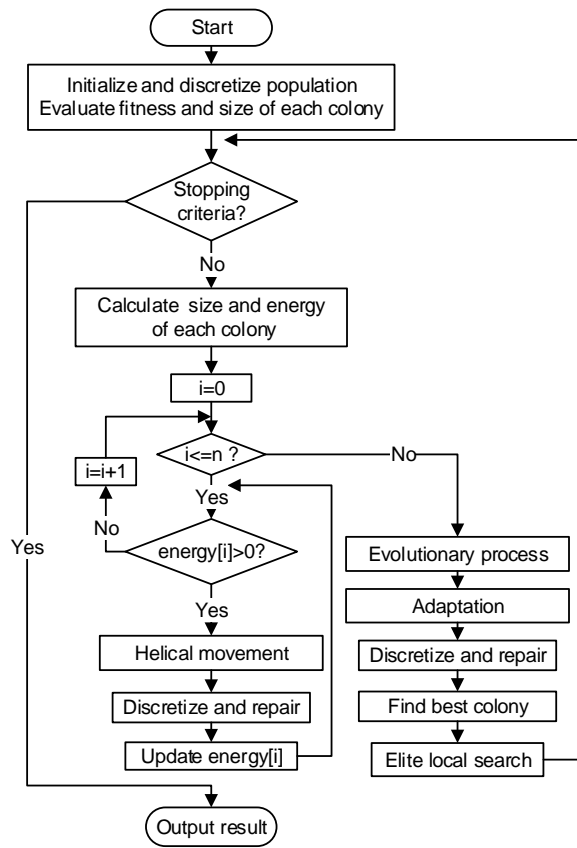


Figure 4: The flowchart of BAAA.

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

269 The proposed algorithm is implemented in C++ within Microsoft Visual Studio 2010 using
 270 a PC with Intel Core (TM) 2 Duad CPU Q9300 @2.5 GHz, 4 GB RAM and 64-bit Windows
 271 7 operating system. The point-estimator of digits is studied in [37]. Here we will use standard
 272 truncation method to report numerical results. If the error between the true optimal and that
 273 of obtained by our algorithm is less than 10^{-8} , we say that our algorithm has successfully
 274 found the solution.

275 As mentioned above, the selection of τ is a key step for the balance of search ability between
 276 exploitation and exploration. To clarify the influence of τ on BAAA, a comparison test is
 277 implemented using different τ on the instance Sento1 which has 60 items and 30 constraints.
 278 The comparison results are depicted in Figs. 5-7. In the experiment, ten different τ between
 279 0.1 and 3.5 are used in the algorithm for 30 independent runs. BAAA with $Tanh(x)$ and
 280 $Sig(x)$ are named as BAAA-Tanh and BAAA-Sig, respectively. The comparison is performed
 281 based on three performance measures: average iteration number (AIT), average fitness value
 282 (AVG), and success rate (SR). AIT reflects the speed of finding optimal solution. It is worth
 283 to mention that AIT only indicates the number of running the outer loop in BAAA. SR
 284 indicates the ratio of the number of finding the optimal solution and the total running times
 285 (30). From Figs. 5-7, we can observe that based on the function $Tanh(x)$, BAAA obtains best
 286 result when τ is 1.5 in terms of AIT, AVG and SR. As far as function $Sig(x)$ is concerned, best
 287 results are obtained when τ is 2. The comparison results confirm that too small or too large
 288 values of τ can downgrade the performance of algorithm. Fig. 5, Fig. 6 and Fig. 7 depict the

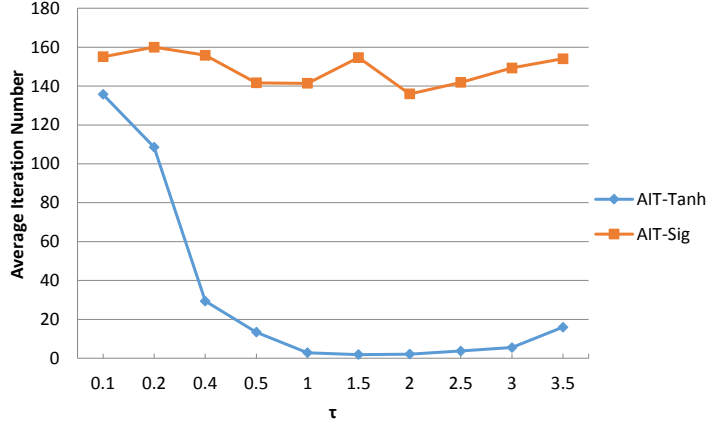


Figure 5: Comparison of AIT of $Tanh(x)$ and $Sig(x)$ on Sento1.

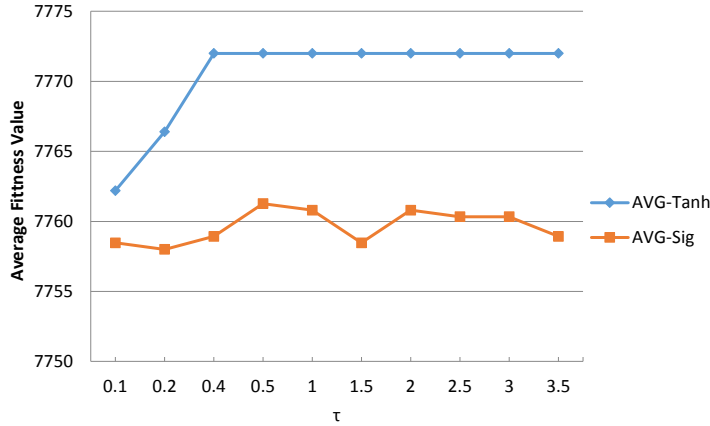


Figure 6: Comparison of AVG of $Tanh(x)$ and $Sig(x)$ on Sento1.

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

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

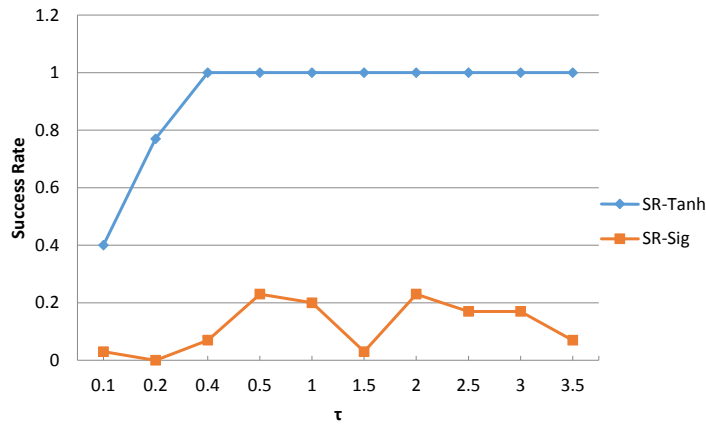


Figure 7: Comparison of SR of $Tanh(x)$ and $Sig(x)$ on Sento1.

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

304 In BAAA, repair operators play a significant role in improving the maximal profit of
 305 the knapsack. The DROP and ADD operators utilize the ranked pseudo-utility ratios to
 306 discard and receive items. Eqs. (22-24) present three pseudo-utility ratios: $\bar{\delta}_i$, $\tilde{\delta}_i$ and δ_i ,
 307 i.e. profit/weight utility, relative profit density and relative mean resource occupation. In
 308 order to verify the effects of the three pseudo-utility ratios on the algorithm, an experiment
 309 is conducted and the results are depicted in Figs. 8-11. Standard deviation (SD) and SR are
 310 considered to measure the performance of algorithm with different pseudo-utility ratios. The
 311 tests are based on 54 instances and each instance is solved by 30 times. The instances from
 312 weish1 to weish17 are left out in Fig. (11) where all runs are able to find optimal solutions at
 313 100% success rate. From these figures, it is difficult to confirm which one is more appropriate
 314 than others. In terms of SR, $\bar{\delta}_i$ fails to find optimal solutions at 100% success rate for 11
 315 instances, while $\tilde{\delta}_i$ and δ_i are 8 and 6, respectively. It seems that δ_i performs better, but its
 316 success rates are 0 for “Pet6” and “Pet7” and the success rates are very low only about 0.1
 317 for “Hp2”, “Pb2” and “Weing7”. As far as SD is concerned, $\tilde{\delta}_i$ obtains less SD than $\bar{\delta}_i$ and
 318 δ_i for “Hp1”, “Pet6” and “Pet7”. However, in other cases it is not true. In general, $\tilde{\delta}_i$ and δ_i
 319 outperform $\bar{\delta}_i$, and each has its own strong point. We adopt relative profit density in BAAA
 320 to compare with other swarm-based algorithms.

321 Elite local search is a greedy local search method which can improve the solution quality
 322 significantly. However, it may take more computational cost for its greedy character to search
 323 better neighbors. In order to gain insight into its effect on the algorithm, a comparison
 324 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

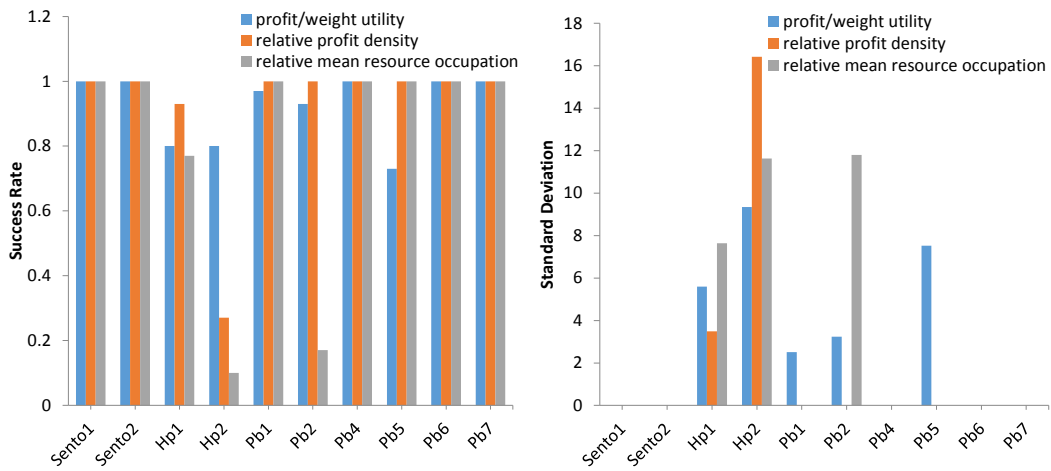


Figure 8: Comparison of SR and SD with three pseudo-utility ratios.

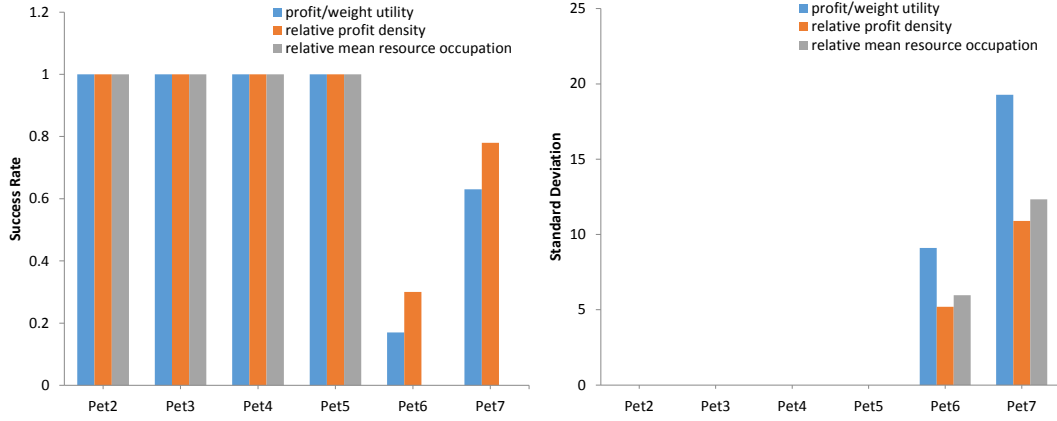


Figure 9: Comparison of SR and SD with three pseudo-utility ratios.

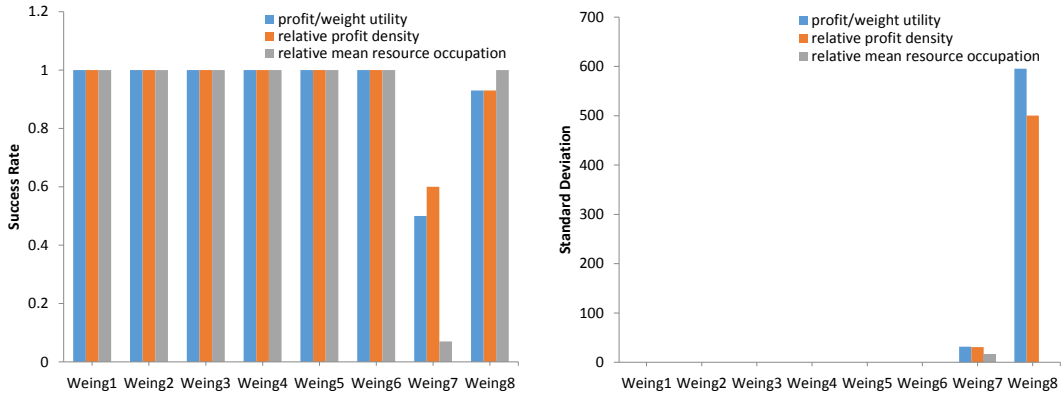


Figure 10: Comparison of SR and SD with three pseudo-utility ratios.

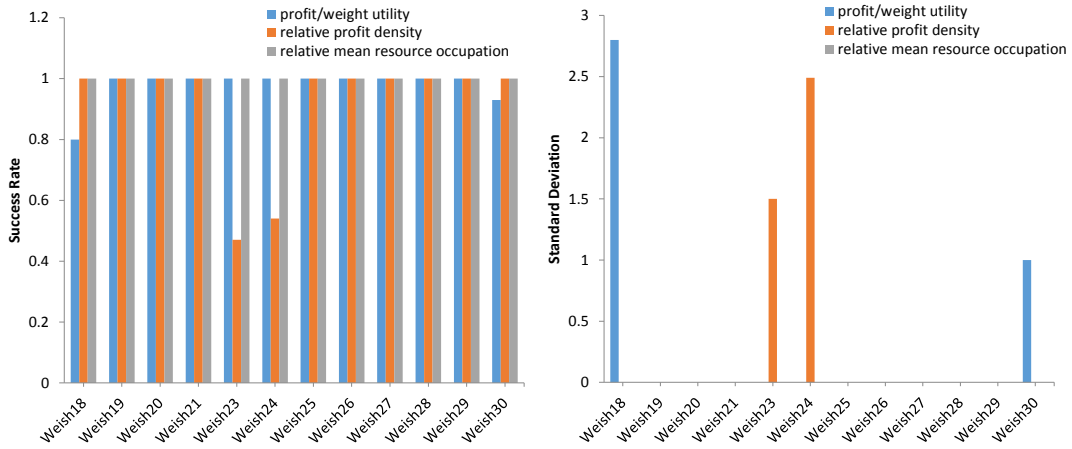


Figure 11: Comparison of SR and SD with three pseudo-utility ratios.

325 Considering elite local search is a built-in feature of BAAA, BAAA without elite local search is
326 named as BAAA-noelite. The Comparative results based on 100 independent runs are shown
327 in Table 2. SR denotes the ratio of the running times reaching the best-known value of 100
328 runs. AT is the average computational time (in seconds). It is quite clear that BAAA obtains
329 better AVG and higher SR than BAAA-noelite. However, AT denotes BAAA costs more
330 computational time than BAAA-noelite, because extra computation is needed to complete
331 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

332 In order to verify the superiority of the algorithm, BAAA is further compared with oth-
333 er population-based algorithms, including the modified binary particle swarm optimization
334 algorithm (MBPSO [20]), particle swarm optimization with time-varying acceleration coeffi-
335 cients (BPSOTVAC and CBPSOTVAC [29]), genetic algorithms with double strings (GADS
336 [16]), binary artificial fish swarm algorithm (bAFSA [25]) and improved binary artificial fish
337 swarm algorithm (IbAFSA [24]). Table 3 summarizes the comparison among MBPSO, BP-
338 SOTVAC, CBPSOTVAC and BAAA based on four different performance criteria, namely,
339 SR, average error (AE), mean absolute deviation (MAD) and SD. AE is calculated as the
340 average of the difference between the values and corresponding optimum solutions. Whereas
341 MAD is the average of the absolute difference between the values and their mean. The data
342 of MBPSO, BPSOTVAC and CBPSOTVAC are collected from original literatures. For the
343 sake of consistency, 100 independent runs of BAAA are carried out for 48 instances. The
344 experimental results show that BAAA performs much better than other three algorithms in
345 terms of SR except for “Hp2”, “Weish23” and “Weish24”. It is worth mentioning that BAAA
346 finds optimal solutions for all the instances and succeeds at 100% success rate for 42 instances.
347 AE, MAD and SD are the measures to evaluate the stability of the algorithms from different

348 angles. Based on the observation from Table 3, most values of AE, MAD and SD obtained
349 by BAAA are less than corresponding values obtained by other three algorithms. In general,
350 BAAA is superior to MBPSO, BPSOTVAC and CBPSOTVAC in terms of effectiveness and
351 robustness.

Table 3: Comparative results of BAAA with MBPSO, BPSOTVAC, and CBP-SOTVAC.

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

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

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

Problem sets	GADS				IbaAFSA				BAAA						
	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.

365 In order to verify the stability of our algorithm, BAAA is compared with HHS [23], ABHS
366 [31] and NGHS [32] in terms of AVG, Min.Dev, Ave.Dev and Var.Dev. Min.Dev is the mini-
367 mum percentage deviations from best-known values. Ave.Dev denotes the average percentage
368 deviations from best-known values. Var.Dev represents the variance of the deviations. The
369 experiment is based on a medium-scaled instances which have 100 items and 10 constraints.
370 For consistency with other algorithms, the algorithm is run 20 times independently for each in-
371 stance. The comparative results are shown in Table 5. From Table 5, we can confirm BAAA
372 is stable in obtaining acceptable solutions because BAAA can achieve minimal Min.Dev,
373 Ave.Dev and Var.Dev, although AVG of BAAA is sometimes inferior to that of HHS.

374 To further reveal the performance of BAAA, we test BAAA on large-scaled problems
375 which have 500 items and 5 constraints with different tightness ratios. The simulation results
376 are compared with those of state-of-the-art algorithms: SACRO-BPSO-TVAC and SACRO-
377 CBPSO-TVAC [30]. This is because [30] is published in the recent and the method in [30]
378 shows its superior to many existing algorithms. Table 6 summarizes the comparative re-
379 sults based on 30 independent runs. We can observe from the results that BAAA performs
380 better than SACRO-BPSO-TVAC and SACRO-CBPSO-TVAC in terms of best obtained val-
381 ue (BEST) in 23 out of 30 instances. BAAA performs worse than SACRO-BPSO-TVAC or
382 SACRO-CBPSO-TVAC in 6 instances in terms of BEST, and the results of instance ‘cb.5.500-
383 0.50_5’ are not available in the reference [30] which are denoted as ‘-’. With respect to AVG
384 and SD, BAAA outperforms SACRO-BPSO-TVAC and SACRO-CBPSO-TVAC clearly. In
385 summary, in contrast to other algorithms, BAAA is more robust and competitive in low-
386 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

387 Furthermore, a non-parametric test, Wilcoxon signed-rank test (W-test) is carried out to
388 determine whether the results from BAAA and those from other algorithms have significant
389 difference or not. Table 7 shows the Wilcoxon signed-rank test results on AVG of BAAA
390 against other algorithms, including ABHS, NGHS, HHS, SACRO-BPSO-TVAC and SACRO-
391 CBPSO-TVAC. R- or R+ is the sum of ranks based on the absolute value of the difference
392 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

393 to the negative difference and R+ indicates the sum of the ranks corresponding to positive
394 difference, respectively. pValue is significant difference between the AVG values of two algo-
395 rithms, which is calculated by the software SPSS statistics 22. A null hypothesis is assumed
396 that there is no significant difference between the two samples and an alternative hypothesis
397 is assumed that there is a significant difference between the two samples, at 0.05 significance
398 level. According to the relationship between pValue and 0.05 significance level, we obtain
399 the result which is represented by three signs: “+”, “-” or “≈”. “+” or “-” denotes the
400 first algorithm is significantly better or worse than the second one, i.e. there is a significant
401 difference. And “≈” denotes there is no significant difference between the two algorithms.
402 It can be seen from Table 7 that BAAA is superior to ABHS, NGHS, SACRO-BPSO-TVAC
403 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	+

404 5. Conclusions

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

418 **Acknowledgements**

419 This paper was partially supported by Australian Research Council Linkage Program
420 LP130100451, a grant from Korean Research Foundation, Natural Science Foundation of Chi-
421 na (Nos. 61473326 and 61471132), Natural Science Foundation of Chongqing (cstc2013jcyjA00029
422 and cstc2013jjB0149).

423 **Reference**

- 424 [1] B. W.-K. L. Ling, C. Y.-F. Ho, J. Cao, Q. Dai, Efficient complex-valued finite word
425 length allpass rational iir pcls filter design via functional inequality constrained integer
426 programming with bit plane searching technique, *Mediterranean Journal of Electronics
427 and Communications* 9 (2013) 588–593.
- 428 [2] B. W.-K. L. Ling, N. Tian, C. Y.-F. Ho, W.-C. Siu, K.-L. Teo, Q. Dai, Maximally
429 decimated paraunitary linear phase fir filter bank design via iterative svd approach,
430 *IEEE Transactions on Signal Processing* 63 (2015) 466–481.
- 431 [3] B. W.-K. L. Ling, C. Y.-F. Ho, K.-L. Teo, W.-C. Siu, , J. Cao, Q. Dai, Optimal design of
432 cosine modulated nonuniform linear phase fir filter bank via both stretching and shifting
433 frequency response of single prototype filter, *IEEE Transactions on Signal Processing* 62
434 (2014) 2517–2530.
- 435 [4] S. R. Subramaniam, B. W.-K. L. Ling, A. Georgakis, Filtering in rotated time-frequency
436 domains with unknown noise statistics, *IEEE Transactions on Signal Processing* 60 (2012)
437 489–493.
- 438 [5] B. W.-K. L. Ling, C. Y.-F. Ho, S. R. Subramaniam, A. Georgakis, J. Cao, Q. Dai,
439 Optimal design of hermitian transform and vectors of both mask and window coefficients
440 for denoising applications with both unknown noise characteristics and distortions, *Signal
441 Processing* 98 (2014) 1–22.
- 442 [6] A. Fréville, The multidimensional 0–1 knapsack problem: An overview, *European Journal
443 of Operational Research* 155 (2004) 1–21.
- 444 [7] D. Bertsimas, R. Demir, An approximate dynamic programming approach to multidi-
445 mensional knapsack problems, *Management Science* 48 (2002) 550–565.

- 446 [8] J. Puchinger, G. R. Raidl, U. Pferschy, The multidimensional knapsack problem: Struc-
447 ture and algorithms, *INFORMS Journal on Computing* 22 (2010) 250–265.
- 448 [9] M. J. Varnamkhasti, Overview of the algorithms for solving the multidimensional knap-
449 sack problems, *Advanced Studies in Biology* 4 (2012) 37–47.
- 450 [10] S. Balev, N. Yanev, A. Fréville, R. Andonov, A dynamic programming based reduc-
451 tion procedure for the multidimensional 0–1 knapsack problem, *European Journal of*
452 *Operational Research* 186 (2008) 63–76.
- 453 [11] V. C. Li, Y.-C. Liang, H.-F. Chang, Solving the multidimensional knapsack problems
454 with generalized upper bound constraints by the adaptive memory projection method,
455 *Computers & Operations Research* 39 (2012) 2111–2121.
- 456 [12] M. Vasquez, J.-K. Hao, et al., A hybrid approach for the 0-1 multidimensional knapsack
457 problem, in: *IJCAI, 2001*, pp. 328–333.
- 458 [13] J. E. Gallardo, C. Cotta, A. J. Fernández, Solving the multidimensional knapsack prob-
459 lem using an evolutionary algorithm hybridized with branch and bound, in: *Artificial*
460 *Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, Springer,
461 2005, pp. 21–30.
- 462 [14] P. C. Chu, J. E. Beasley, A genetic algorithm for the multidimensional knapsack problem,
463 *Journal of heuristics* 4 (1998) 63–86.
- 464 [15] F. Djannaty, S. Doostdar, A hybrid genetic algorithm for the multidimensional knapsack
465 problem, *International Journal of Contemporary Mathematical Sciences* 3 (2008) 443–
466 456.
- 467 [16] M. Sakawa, K. Kato, Genetic algorithms with double strings for 0–1 programming prob-
468 lems, *European Journal of Operational Research* 144 (2003) 581–597.
- 469 [17] S. Hanafi, A. Freville, An efficient tabu search approach for the 0–1 multidimensional
470 knapsack problem, *European Journal of Operational Research* 106 (1998) 659–675.
- 471 [18] F. Qian, R. Ding, Simulated annealing for the 0/1 multidimensional knapsack problem,
472 *Numerical Mathematics - English Series* 16 (2007) 320.
- 473 [19] F. Hembeker, H. S. Lopes, W. Godoy Jr, Particle swarm optimization for the mul-
474 tidimensional knapsack problem, in: *Adaptive and Natural Computing Algorithms*,
475 Springer, 2007, pp. 358–365.

- 476 [20] J. C. Bansal, K. Deep, A modified binary particle swarm optimization for knapsack
477 problems, *Applied Mathematics and Computation* 218 (2012) 11042–11061.
- 478 [21] A. Baykasoğlu, F. B. Ozsoydan, An improved firefly algorithm for solving dynamic
479 multidimensional knapsack problems, *Expert Systems with Applications* 41 (2014) 3712–
480 3725.
- 481 [22] X. Kong, L. Gao, H. Ouyang, S. Li, Solving large-scale multidimensional knapsack prob-
482 lems with a new binary harmony search algorithm, *Computers & Operations Research*
483 63 (2015) 7–22.
- 484 [23] B. Zhang, Q.-K. Pan, X.-L. Zhang, P.-Y. Duan, An effective hybrid harmony search-based
485 algorithm for solving multidimensional knapsack problems, *Applied Soft Computing* 29
486 (2015) 288–297.
- 487 [24] M. A. K. Azad, A. M. A. Rocha, E. M. Fernandes, Improved binary artificial fish swarm
488 algorithm for the 0–1 multidimensional knapsack problems, *Swarm and Evolutionary*
489 *Computation* 14 (2014) 66–75.
- 490 [25] M. A. K. Azad, A. M. A. Rocha, E. M. Fernandes, Solving multidimensional 0–1 knapsack
491 problem with an artificial fish swarm algorithm, in: *Computational Science and Its*
492 *Applications–ICCSA 2012*, Springer, 2012, pp. 72–86.
- 493 [26] X.-S. Yang, Z. Cui, R. Xiao, A. H. Gandomi, M. Karamanoglu, *Swarm intelligence and*
494 *bio-inspired computation: theory and applications*, Newnes, 2013.
- 495 [27] S. Binitha, S. S. Sathya, A survey of bio inspired optimization algorithms, *International*
496 *Journal of Soft Computing and Engineering* 2 (2012) 137–151.
- 497 [28] J. Kennedy, R. C. Eberhart, A discrete binary version of the particle swarm algorithm,
498 in: *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation.*,
499 1997 IEEE International Conference on, volume 5, IEEE, 1997, pp. 4104–4108.
- 500 [29] M. Chih, C.-J. Lin, M.-S. Chern, T.-Y. Ou, Particle swarm optimization with time-
501 varying acceleration coefficients for the multidimensional knapsack problem, *Applied*
502 *Mathematical Modelling* 38 (2014) 1338–1350.
- 503 [30] M. Chih, Self-adaptive check and repair operator-based particle swarm optimization for
504 the multidimensional knapsack problem, *Applied Soft Computing* 26 (2015) 378–389.

- 505 [31] L. Wang, R. Yang, Y. Xu, Q. Niu, P. M. Pardalos, M. Fei, An improved adaptive binary
506 harmony search algorithm, *Information Sciences* 232 (2013) 58–87.
- 507 [32] D. Zou, L. Gao, S. Li, J. Wu, Solving 0–1 knapsack problem by a novel global harmony
508 search algorithm, *Applied Soft Computing* 11 (2011) 1556–1564.
- 509 [33] S. A. Uymaz, G. Tezel, E. Yel, Artificial algae algorithm (aaa) for nonlinear global
510 optimization, *Applied Soft Computing* 31 (2015) 153–171.
- 511 [34] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Prob-
512 lem definitions and evaluation criteria for the cec 2005 special session on real-parameter
513 optimization, *KanGAL report 2005005* (2005).
- 514 [35] H. Pirkul, A heuristic solution procedure for the multiconstraint zero-one knapsack
515 problem, *Naval Research Logistics* 34 (1987) 161–172.
- 516 [36] H. O. S. L. Xiangyong Konga, Liqun Gaoa, Solving large-scale multidimensional knapsack
517 problems with a new binary harmony search algorithm, *Computers and Operations*
518 *Research* 63 (2015) 7–22.
- 519 [37] W. T. Song, B. W. Schmeiser, Omitting meaningless digits in point estimates: The
520 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

```
1: compute  $\delta_i, i=1,2,\dots,d$ 
2: initialize  $s(i)=i, i=1,2,\dots,d$ 
3: sort  $s(i)$  rendering  $\delta_{s(i)}$  be in ascending order
   //DROP phase
4: if(not feasible( $x$ ))
5:   for  $i=1$  to  $d$  do
6:     if( $x_{s(i)}=1$ )
7:        $x_{s(i)} = 0$ 
8:       if(feasible( $x$ )) break
9:     end if
10:  end for
11: end if
   //ADD phase
12: for  $i=d$  to  $1$  do
13:   if( $x_{s(i)}=0$ )
14:      $x_{s(i)} = 1$ 
15:     if(not feasible( $x$ ))  $x_{s(i)} = 0$ 
16:   end if
17: end for
18: return  $x$ .
```

Algorithm 2 *EliteLocalSearch* procedure

Input:

a current best solution x^b

Output:

an improved solution x^b

```
1: for i=1 to d do
2:   for j=1 to d do
3:     if (i!=j and  $x_i^b \neq x_j^b$ )
4:        $x = \text{swap}(x^b, i, j)$  //exchange the ith and jth elements of the solution vector
5:       if ( $\text{fitness}(x) > \text{fitness}(x^b)$ )  $x^b = x$ 
6:     end if
7:   end for
8: end for
9: return  $x^b$ .
```

Algorithm 3 Binary artificial algae algorithm

Input: c, p, b **Output:**

the maximized profit of knapsack

```
1: define  $n, sf, eloss, A_p$ 
2: initialize population of algal colony  $x_i$  and repair  $x_i, i = 1, 2, \dots, n$ 
3:  $starvation_i = 0, i = 1, 2, \dots, n$ 
4: while ( $t < T_{max}$ )
5:     calculate energy  $E_i$  and friction surface  $\omega_i$  according to size of  $x_i, i = 1, 2, \dots, n$ 
6:     for  $i=1$  to  $n$  do
7:         isstarve=true
8:         while ( $E_i > 0$  and  $t < T_{max}$ )
9:             calculate  $j$  through tournament selection method
10:            choose distinct  $k, l, m$  randomly between 1 and  $d$ 
11:            produce  $\alpha, \beta, p$  randomly where  $\alpha$  and  $\beta$  are in the range  $[0, 2\pi]$ ,  $p$  is between -1 and 1
12:             $x_{im} = x_{im} + (x_{jm} - x_{im})(sf - \omega_i)p$ 
13:             $x_{ik} = x_{ik} + (x_{jk} - x_{ik})(sf - \omega_i) \cos \alpha$ 
14:             $x_{il} = x_{il} + (x_{jl} - x_{il})(sf - \omega_i) \sin \beta$ 
15:            discretize and repair  $x_i$ 
16:             $E_i = E_i - eloss/2$ 
17:            if (new fitness value of  $x_i$  is better than old one)
18:                accept  $x_i$  and update corresponding fitness value
19:                isstarve=false
20:            else
21:                 $E_i = E_i - eloss/2$ 
22:            end if
23:             $t=t+1$ 
24:        end while
25:        if (isstarve)  $starvation_i = starvation_i + 1$ 
26:    end for
27:    the  $r_{th}$  dimension of smallest algal colony is replaced by that of biggest one, where  $r$  is selected randomly
    between 1 to  $d$ 
28:    if ( $A_p > rand$ )
29:        select the most starving algal colony  $x_s$ , and  $x_s = x_s + (biggest - x_s) * rand$ 
30:        discretize and repair  $x_s$ 
31:    end if
32:    best=findBest( $x$ )
33:    ebest=eliteLocalSearch(best)
34: end while
35: return ebest
```
