A Simplified Binary Artificial Fish Swarm Algorithm for 0–1 Quadratic Knapsack Problems

Md. Abul Kalam Azad[a,*], Ana Maria A.C. Rocha[a,b], Edite M.G.P. Fernandes[a]

[a] *Algoritmi R&D Centre*
[b] *Department of Production and Systems*
School of Engineering, University of Minho, 4710-057 Braga, Portugal

Abstract

This paper proposes a simplified binary version of the artificial fish swarm algorithm (S-bAFSA) for solving 0–1 quadratic knapsack problems. This is a combinatorial optimization problem, which arises in many fields of optimization. In S-bAFSA, trial points are created by using crossover and mutation. In order to make the points feasible, a random heuristic drop_item procedure is used. The heuristic add_item is also implemented to improve the quality of the solutions, and a cyclic reinitialization of the population is carried out to avoid convergence to non-optimal solutions. To enhance the accuracy of the solution, a swap move heuristic search is applied on a predefined number of points. The method is tested on a set of benchmark 0–1 knapsack problems.

Keywords: 0–1 knapsack problem, heuristic, artificial fish swarm, swap move

1. Introduction

In this paper, we are particularly interested in the 0–1 quadratic knapsack problem (QKP) consisting in maximizing a quadratic objective function subject to a linear capacity constraint. This problem was introduced in [6]

*Corresponding author

Email addresses: akazad@dps.uminho.pt (Md. Abul Kalam Azad), arocha@dps.uminho.pt (Ana Maria A.C. Rocha), emgpf@dps.uminho.pt (Edite M.G.P. Fernandes)

Preprint submitted to Journal of Computational and Applied Mathematics August 7, 2013

and may be expressed as follows:

$$\begin{aligned} &\text{maximize } f(\mathbf{x}) \equiv \sum_{i=1}^{n} p_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} x_i x_j \\ &\text{subject to} \qquad \sum_{i=1}^{n} w_i x_i \le c \\ &\qquad\qquad\qquad x_i \in \{0,1\}, i = 1, 2, \ldots, n, \end{aligned} \tag{1}$$

where $\mathbf{x}$ is the n-dimensional vector of the 0/1 decision variables (items), p_i is a profit achieved if item i is selected and p_{ij} $(i = 1, 2, \ldots, n-1, j = i+1, \ldots, n)$ is a profit achieved if both items i and j $(j > i)$ are selected. w_i is the weight coefficient of item i and c is the capacity of the knapsack. p_i, p_{ij} and w_i are positive integers and c is an integer such that $\max\{w_i : i = 1, 2, \ldots, n\} \le c < \sum_{i=1}^{n} w_i$. The goal is to find a subset of n items that yields maximum profit f without exceeding knapsack capacity c. We may observe that if $p_{ij} = 0$ then the problem becomes a 0–1 linear knapsack problem (LKP).

The 0–1 QKP arises in a variety of real world applications, including finance, VLSI design, compiler construction, telecommunication, flexible manufacturing systems, location of airports, railway stations, freight handling terminals, hydrological studies. Classical graph and hypergraph partitioning problems can also be formulated as the 0–1 QKP. Several deterministic solution methods [2, 3, 4, 5, 6, 8, 9, 10, 15, 20, 23] as well as stochastic solution methods [7, 13, 17, 26] have been proposed to solve (1). Billionnet and Soutif [2] used a linear reformulation technique for the 0–1 QKP and solved them efficiently using a standard mixed integer programming tool. In [3], an exact method based on the computation of an upper bound by Lagrangian decomposition is proposed. Caprara et al. [5] investigated an exact branch and bound algorithm for the 0–1 QKP, where upper bounds are computed by considering a Lagrangian relaxation which is solvable through a number of (continuous) knapsack problems. Létocart et al. [15] presented reoptimization techniques for improving the efficiency of the preprocessing phase of the 0–1 quadratic knapsack resolution. In [20], an exact algorithm which makes usage of aggressive reduction techniques to decrease the size of the instance to a manageable size is introduced. An exact solution method based on a new linearization scheme is proposed in Rodrigues et al. [23].

The deterministic and exact methods are suitable for small dimensional problems. However, when the dimension increases, they cannot solve the problems within a reasonable time period. This is the main motivation to develop stochastic methods and heuristics for solving QKP. In the context of constrained problems, the widely used approach is based on penalty functions. In this approach, a penalty term is added to the objective function aiming to penalize constraint violation. The penalty function method can be applied to any type of constraints, but the performance of penalty-type method is not always satisfactory due to the choice of appropriate penalty parameter values. Hence, other alternative constraint handling techniques have emerged in the last decades.

Examples of stochastic population-based methods to solve the 0–1 QKP follow. Glover and Kochenberger [7] reformulated the 0–1 QKP to unconstrained binary quadratic problem and solved using Tabu search. In [13], a hybridization of the genetic algorithm with greedy heuristic based on the *absolute-profit to weight* ratio is proposed. Here, the capacity constraint is handled by never generating chromosomes whose solutions violate it. Narayan and Patvardhan [17] introduced a novel quantum evolutionary algorithm for the 0–1 QKP and Xie and Liu [26] presented an agent-based mini-swarm algorithm using the *absolute-profit to weight* ratio to repair and improve the solutions.

Unlike the stochastic methods, the outcome of a deterministic method does not depend on pseudo random variables. In general, its performance depends heavily on the structure of the problem since the design relies on the mathematical attributes of the optimization problem. In comparison with the deterministic methods, the implementation of stochastic algorithms is often easier. A survey of different methods for solving the 0–1 QKP is found in [19].

The artificial fish swarm algorithm (AFSA) is an example of a stochastic method that has recently appeared to solve continuous and engineering design optimization problems [11, 12, 24, 25]. When applied to an optimization problem, a 'fish' represents an individual point in a population. The algorithm simulates the behavior of a fish swarm inside water. At each iteration, trial points are generated from the current ones using either a chasing behavior, a swarming behavior, a searching behavior or a random behavior. Each trial point competes with the corresponding current and the one with best fitness is passed to the next iteration as current point. There are in the scientific literature different versions and hybridizations of AFSA [18, 21, 22].

This paper presents a simplified binary version of AFSA for solving the 0–1 QKP. A previous binary version of AFSA, denoted by bAFSA, is presented in [1], where a set of small 0–1 multidimensional knapsack problems were successfully solved. Nevertheless, the computational effort required by bAFSA when solving large dimensional problems is not satisfactory. To create the trial points from the current ones in a population, bAFSA chooses each point/fish behavior according to the number of points inside its 'visual scope', i.e., inside a closed neighborhood centered at the point. To identify those points, the Hamming distance between pairs of points is used. When the chasing behavior is chosen, the trial point is created after performing an uniform crossover between the individual point and the best point inside the 'visual scope'. On the other hand, when the swarming behavior is chosen, a uniform crossover between the individual point and the central point of the 'visual scope' is performed to create the trial point. When the searching behavior is chosen, the trial point is created by performing a uniform crossover between the individual point and a randomly chosen point from the 'visual scope'. Finally, in the random behavior, the trial point is created by randomly setting a binary string of 0/1 bits of length n. Past experience has shown that the time related with the computation of the 'visual scope' of all points, at each iteration, is $\mathcal{O}(Nn^2)$, where N is the number of points in the population.

The purpose of the herein presented study is to simplify the procedures that are used to choose which behavior is to be performed to each current point in order to create the corresponding trial point. The main goal is to reduce the computational requirements, in terms of number of iterations and execution time, to reach the optimal solution. This is a new simplified binary version of AFSA, henceforth denoted by S-bAFSA. Briefly, for all points of the population, except the best, random, searching and chasing behavior are randomly chosen using two target probability values $0 \leq \tau_1 \leq \tau_2 \leq 1$, and thereafter an uniform crossover is operated to create the trial points. A simple 4-flip mutation is performed in the best point of the population to generate the corresponding trial point. To make the points feasible, the new S-bAFSA uses a random heuristic drop_item procedure followed by an add_item operation aiming to increase the profit throughout the adding of more items in the knapsack. Furthermore, to improve the accuracy of the solutions obtained by the algorithm, a swap move heuristic search [14] and a cyclic reinitialization of the population are implemented. A benchmark set of 0–1 knapsack problems is used to test the performance of the S-bAFSA.

The organization of this paper is as follows. The proposed simplified binary version of the artificial fish swarm algorithm is described in Section 2. Section 3 describes the experimental results and finally we draw the conclusions of this study in Section 4.

2. The Proposed S-bAFSA

In the previous binary version of AFSA [1], each trial point is created from the current one by using the original concept of 'visual scope' of a point. To identify the points inside the 'visual scope' of each individual point, the Hamming distance is used. For points of equal bits length, this distance is the number of positions at which the corresponding bits are different. The computational requirement of this procedure grows rapidly with problem's dimension. Furthermore, in some cases the population stagnates and the algorithm converges to a non-optimal solution.

To address these issues, we present a simplified binary version with the following properties.

- The concept of 'visual scope' of an individual point is discarded.
- The selection of each fish/point behavior does not depend on the number of points in the neighborhood of that point but rather on two target probability values.
- The swarming behavior is never performed since the central point may not depict the center of the distribution of solutions.
- A random heuristic drop_item procedure to make infeasible solutions to feasible ones, and an add_item operation, are combined to further improve the feasible solutions.
- A simple heuristic search based on swap moves is implemented on a predefined number of points randomly selected from the population, aiming to obtain more accurate solutions.
- The population is randomly reinitialized to diversify the search and avoid convergence to a non-optimal solution.

Details of the proposed S-bAFSA to solve the 0–1 knapsack problem (1) are described in the following. The first step of S-bAFSA is to design a suitable representation scheme of an individual point in a population for solving the 0–1 QKP. Since we consider the 0–1 knapsack problem, N individual points, $\mathbf{x}^k, k = 1, \ldots, N$, each represented by a binary string of 0/1 bits of length n, are randomly generated [1, 16]. We note that there are at most 2^n possible different solutions of binary strings of 0/1 bits of length n. The pseudocode of the herein proposed S-bAFSA for solving the 0–1 QKP (1) is shown in Algorithm 1.

Algorithm 1 S-bAFSA

Require: $T_{\max}$ and f_{opt} and other values of parameters
Set $t := 1$. Initialize population $\mathbf{x}^k, k = 1, 2, \ldots, N$
Perform random drop_item and add_item, evaluate the population and identify $\mathbf{x}^{\text{best}}$ and f_{best}
while 'termination conditions are not met' **do**
    if $\text{MOD}(t, R) = 0$ **then**
        Reinitialize population $\mathbf{x}^k, k = 1, 2, \ldots, N-1$
        Perform random drop_item and add_item, evaluate population and identify $\mathbf{x}^{\text{best}}$ and f_{best}
    end if
    for $k = 1$ to N **do**
        if $k = \text{best}$ **then**
            Perform 4 flip-bit mutation to create trial point $\mathbf{y}^k$
        else
            if $rand(0,1) \leq \tau_1$ **then**
                Perform random behavior to create trial point $\mathbf{y}^k$
            else if $rand(0,1) \geq \tau_2$ **then**
                Perform chasing behavior to create trial point $\mathbf{y}^k$
            else
                Perform searching behavior to create trial point $\mathbf{y}^k$
            end if
        end if
    end for
    Perform random drop_item and add_item to get $\mathbf{y}^k, k = 1, 2, \ldots, N$ and evaluate them
    Select the population of next iteration $\mathbf{x}^k, k = 1, 2, \ldots, N$
    Perform the swap move heuristic search
    Identify $\mathbf{x}^{\text{best}}$ and f_{best}
    Set $t := t + 1$
end while

Generating trial points in S-bAFSA After initializing N individual points, crossover and mutation are performed to create trial points in successive iterations based on the fish behavior of random, searching and chasing. We introduce the probabilities $0 \leq \tau_1 \leq \tau_2 \leq 1$ in order to perform the movements of random, searching and chasing. The fish behavior in S-bAFSA that create the trial points are outlined as follows.

In random behavior, a fish with no other fish in its neighborhood to follow, moves randomly looking for food in another region. This behavior is implemented when a uniformly distributed random number $rand(0,1)$[1] is less than or equal to τ_1. In this behavior the trial point $\mathbf{y}^k$ is created by randomly setting 0/1 bits of length n.

The chasing behavior is implemented when a fish, or a group of fish in the swarm, discover food and the others find the food dangling quickly after it. This behavior is implemented when $rand(0,1) \geq \tau_2$ and it is related to the movement towards the best point found so far in the population, $\mathbf{x}^{\text{best}}$. Here, the trial point $\mathbf{y}^k$ is created using a uniform crossover between $\mathbf{x}^k$ and $\mathbf{x}^{\text{best}}$. In uniform crossover, each bit of the trial point is created by copying the corresponding bit from one or the other current point with equal probability.

When fish discovers a region with more food, by vision or sense, it goes directly and quickly to that region. This is the searching behavior and is related to the movement towards a point $\mathbf{x}^{\text{rand}}$ where 'rand' is an index randomly chosen from the set $\{1, 2, \ldots, N\}$. This behavior is implemented in S-bAFSA when $\tau_1 < rand(0,1) < \tau_2$. A uniform crossover between $\mathbf{x}^{\text{rand}}$ and $\mathbf{x}^k$ is performed to create the trial point $\mathbf{y}^k$.

In S-bAFSA, the three fish behavior previously described are implemented to create $N-1$ trial points; the best point $\mathbf{x}^{\text{best}}$ is treated separately. A mutation is performed in the point $\mathbf{x}^{\text{best}}$ to create the corresponding trial point $\mathbf{y}$. In mutation, a 4 flip-bit operation is performed, i.e., four positions are randomly selected and the bits of the corresponding positions are changed from 0 to 1 or vice versa.

Making feasible solutions There are a number of standard ways of dealing with constraints in binary represented population-based methods. In S-bAFSA, we use a random heuristic procedure called drop_item in order to make the solutions feasible. At first, a set $\mathbf{i} = \{i_1, i_2, \ldots, i_n\}$ is defined with n randomly generated indices. Then the drop_item is performed on $\mathbf{x}^k$ using the set $\mathbf{i}$ to make the point feasible. Following the sequence of indices in the set $\mathbf{i}$, one item is dropped (changing bit 1 to 0) each time from the knapsack, if with this item the point does not satisfy the constraint. This procedure is continued until the feasible solution is reached. The advantage of this procedure is that dropping an item starts from any index and randomly continues selecting an index until the feasible solution is reached, aiming to obtain a promising solution.

[1] We note that the procedure used to generate a random number in C (rand()/(RAND_MAX + 1)) may give a zero number but will never give a one.

After making the point feasible, a greedy-like heuristic called add_item is implemented to each feasible individual point aiming to improve that point without violating the knapsack constraint. This heuristic procedure uses the information of the *absolute-profit to weight* ratio, δ_i, which is defined as the ratio of the sum of all profit associated with the item i to its weight [13], i.e., $\delta_i = (p_i + \sum_{j \neq i} p_{ij})/w_i$. The greater the ratio, the higher the chance of inclusion of that item in the knapsack. In S-bAFSA, all δ_i are sorted in decreasing order and a set $\mathbf{j} = \{j_1, j_2, \ldots, j_n\}$ is defined with the indices of the δ_i in decreasing order. One item is added (changing bit 0 to 1) each time in the knapsack, if with this item the point does not violate the constraint following the sequence of indices in the set $\mathbf{j}$. This procedure is continued until the entire sequence of indices has been used.

The *absolute-profit to weight* ratios $\delta_i, i = 1, 2, \ldots, n$ can also be used in order to make the points feasible. In this case, all δ_i are sorted in increasing order and one item is dropped from the knapsack, if with this item the point does not satisfy the constraint. This procedure is continued until the feasible solution is reached.

Selection of the new population At each iteration, each trial point $\mathbf{y}^k$ competes with the current $\mathbf{x}^k$, in order to decide which one should become a member of the population in the next iteration. Hence, if $f(\mathbf{y}^k) \geq f(\mathbf{x}^k)$, then the trial point becomes a member of the population in the next iteration, otherwise the current point is preserved to the next iteration.

Swap move heuristic search A heuristic search is often important to improve a current solution. It searches for a better solution in the neighborhood of the current solution. If such solution is found then it replaces the current solution. In S-bAFSA, we implement a simple heuristic search based on swap moves [14] after the selection procedure. In this search, the swap moves change the value of a 0 bit of an individual point to 1 and simultaneously another 1 bit to 0, so that the total number of items in the knapsack does not change. Here, the swap move heuristic search method has two parameters: N_{loc}, which gives the number of points selected randomly from the population to perform the heuristic search and n_{swap}, which sets the number of positions selected randomly in a point to perform the swap moves. They are defined as follows: $N_{\text{loc}} = \tau_3 N$ with $\tau_3 \in (0, 1)$ and $n_{\text{swap}} = \tau_4 N_{\text{bit\_0}}$, where $\tau_4 \in (0, 1)$ and $N_{\text{bit\_0}}$ is the number of 0 bits in a point. After performing the swap move heuristic search, the new points are made feasible by using the random drop_item algorithm and thereon the add_item. Then they become members of the population if they improve the objective function value with respect to the corresponding current points.

Termination conditions Let $T_{\max}$ be the maximum number of iterations. Let f_{best} be the maximum objective function value attained at iteration t and f_{opt} be the known optimal value available in the literature. The proposed S-bAFSA terminates when the known optimal solution is reached within a tolerance $\epsilon > 0$, or $T_{\max}$ is exceeded, i.e., when

$$t > T_{\max} \text{ or } |f_{\text{best}} - f_{\text{opt}}| \leq \epsilon \tag{2}$$

holds. However, if the optimal value of the given problem is not known, the algorithm may use another condition, for example, one based on the total number of function evaluations or the computational time since the start of the algorithm.

Reinitialization of the population When testing bAFSA [1], it was noticed that, in some cases, the points in a population converge to a non-optimal point. To diversify the search, we propose to randomly reinitialize the population, every R iterations, keeping the best solution found so far. In practical terms, this technique has greatly improved the quality of the solutions.

3. Experimental Results

We code S-bAFSA in C and compile with Microsoft Visual Studio 10.0 compiler in a PC having 2.5 GHz Intel Core 2 Duo processor and 4 GB RAM. We consider 80 benchmark 0–1 QKP test instances[2] with $n = 100$ and 200 items, and density $d = 0.25$, 0.50, 0.75 and 1.00. The density means that the non-zeros in the profit coefficients should be $100d$ percentage. These instances are widely used for the measurement of effectiveness of an algorithm in the optimization community. Since they are benchmark instances, the optimal solution, f_{opt}, is known and the termination condition (2) can be used to terminate the algorithm.

Firstly, we analyze the performance of S-bAFSA with different values of τ_1 and τ_2. We consider 10 instances with $n = 100, d = 0.25$ and 10 instances with $n = 200, d = 1.00$. We set $N = n$, $T_{\max} = 10n$ and $\epsilon = 10^{-4}$. After several experiments, we set the parameter $R = 100$ for the reinitialization

[2] (`http://cedric.cnam.fr/~soutif/QKP/`)

of the population. The results are analyzed for four combinations of τ_1 and τ_2: i) $\tau_1 = 0.0, \tau_2 = 0.0$, ii) $\tau_1 = 0.0, \tau_2 = 1.0$, iii) $\tau_1 = 0.1, \tau_2 = 0.9$ and iv) $\tau_1 = 1.0, \tau_2 = 1.0$. Fifty independent runs were carried out for each instance with each combination of τ_1 and τ_2. If the algorithm finds the optimal solution (or near optimal according to an error tolerance) to an instance in a run, then the run is considered to be a successful one. Table 1 contains the acronyms of the performance criteria used in this paper.

Table 1: Acronyms of the performance criteria

AIT	– average number of iterations among 50 runs and successful runs
aAIT	– average of AIT over 10 instances
T	– computational time (in seconds)
aT	– average of T over 10 instances
AT	– average computational time (in seconds) among 50 runs and successful runs
aAT	– average of AT over 10 instances
BT	– best computational time to reach best solution among 50 runs
aBT	– average of BT over 10 instances
Nsr	– number of successful runs among 50 runs
aNsr	– average of Nsr over 10 instances
SR	– percentage of successful runs among 50 runs
aSR	– average of SR over 10 instances
f_{avg}	– average objective function value among 50 runs

The results obtained among 50 runs and among successful (succ.) runs of the two sets of problems are summarized in Table 2. From the table, it is

Table 2: Results of different values for τ_1 and τ_2 of S-bAFSA

			50 runs			succ. runs	
Prob.	τ_1	τ_2	aAIT	aAT	aNsr	aAIT	aAT
100 ($d = 0.25$)	0.0	0.0	541	6.78	27	197	2.44
	0.0	1.0	289	3.55	40	180	2.19
	0.1	0.9	267	3.29	40	124	1.53
	1.0	1.0	766	10.91	13	–	–
200 ($d = 1.00$)	0.0	0.0	947	112.69	31	414	40.57
	0.0	1.0	87	11.31	50	87	11.31
	0.1	0.9	31	3.13	50	31	3.13
	1.0	1.0	1739	224.09	9	–	–

– No successful run in some test instances shown that based on all performance criteria, S-bAFSA with $\tau_1 = 0.1, \tau_2 = 0.9$ gives better performance. Although S-bAFSA with $\tau_1 = 0.0, \tau_2 = 1.0$ gives similar performance based on 'aNsr', it takes more iterations and computational time (among 50 runs and among successful runs) than the version with $\tau_1 = 0.1$ and $\tau_2 = 0.9$. When $\tau_1 = 1.0, \tau_2 = 1.0$, some instances in the set of problems were not solved to optimality. According to the algorithm (See Algorithm 1), when $\tau_1 = 0.0, \tau_2 = 0.0$ S-bAFSA performs chasing behavior mostly (never performing searching), when $\tau_1 = 0.0, \tau_2 = 1.0$ S-bAFSA performs searching behavior mostly (never performing chasing) and when $\tau_1 = 1.0, \tau_2 = 1.0$ S-bAFSA performs random behavior only. Hereafter S-bAFSA will be tested with $\tau_1 = 0.1, \tau_2 = 0.9$.

We now aim to analyze the effect of different types of crossover (used to create trial points in chasing and searching behavior) on the performance of S-bAFSA. They are: i) uniform crossover, ii) one point crossover, iii) two point crossover and iv) two point uniform crossover. The first three types are usually used in evolutionary algorithms. The proposed two point uniform crossover, with equal probability, aims to combine the bit grouping of two point crossover with the randomness of uniform crossover. It proceeds as follows. Taking two points, two positions are randomly selected to make three groups of bits in each point. Then, each group of bits in a trial point will be copied from the corresponding group from one or the other point, with equal probability. This procedure is repeated for the other groups of bits. We note that in uniform and two point uniform crossover, one trial point is created from two points, whereas in one point and two point crossover, two trial points are created from two points, and the best (based on the objective function values) is selected. We consider the above mentioned 20 instances and 50 independent runs were carried out for each instance with each type of crossover. The parameters were maintained as previously defined. Table 3 shows the obtained results among 50 runs and among successful runs. We observe that, based on all performance criteria, S-bAFSA using uniform crossover gives the best performance when solving the 0–1 QKP.

Secondly, we compare S-bAFSA with bAFSA to evaluate their performances. Here also we consider 10 instances with $n = 100, d = 0.25$ and 10 instances with $n = 200, d = 1.00$. We set in both algorithms, $N = n$, $T_{\max} = 10n$, $R = 100$ and $\epsilon = 10^{-4}$. The parameter values in bAFSA are set as suggested in [1]. Fifty independent runs were carried out with each instance using each algorithm. Figure 1 shows the comparison based on 'Nsr',

Table 3: Results of different types of crossover used in S-bAFSA

Prob.	Crossover	50 runs aAIT	50 runs aAT	50 runs aNsr	succ. runs aAIT	succ. runs aAT
100 ($d = 0.25$)	uniform	267	3.29	40	124	1.53
	one point	452	7.59	31	214	3.48
	two point	569	9.39	26	232	3.87
	two point uniform	541	7.34	27	194	2.59
200 ($d = 1.00$)	uniform	31	3.13	50	31	3.13
	one point	857	126.26	33	389	54.46
	two point	1109	156.58	27	589	70.52
	two point uniform	918	107.99	31	407	45.94

'AT', and 'BT'. Both bAFSA and S-bAFSA solved all the problems with $n = 200$ to optimality in all runs. We observe that S-bAFSA performs better than the bAFSA, in particular with the largest problems.

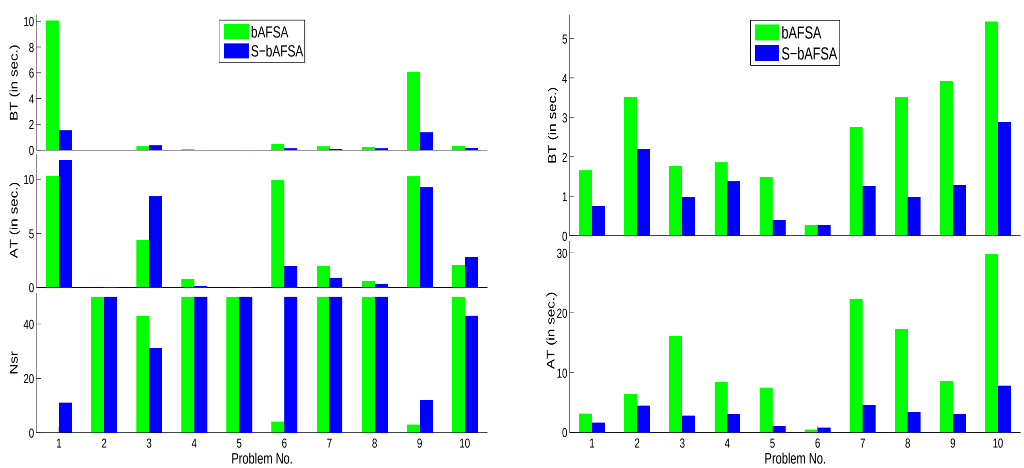

Figure 1: Comparison of bAFSA and S-bAFSA (on the left $n = 100, d = 0.25$ and on the right $n = 200, d = 1.00$)

Thirdly, we compare S-bAFSA with the greedy version of the genetic algorithm (GGA) [13]. We note that GGA and S-bAFSA have in common the use of two operators from the evolutionary algorithms (to create new points): crossover and mutation. It should be noted that S-bAFSA was run with $T_{\max} = 10n$ (value set in GGA). The comparative results are shown in Table 4. S-bAFSA performs rather well when solving the largest problems, with $n = 200$ and $d = 1.00$, and has a surprisingly bad performance on the smallest problems when compared with GGA.

Table 4: Comparative results of GGA and S-bAFSA

Prob.			GGA					S-bAFSA				
n	No.	f_{opt}	Nsr	f_{avg}	AIT	AT	BT	Nsr	f_{avg}	AIT	AT	BT
100	1	18558	50	18558.0	45	0.37	0.08	12	18535.2	814	10.63	0.53
($d =$	2	56525	50	56525.0	3	0.03	0.02	50	56525.0	3	0.02	0.00
0.25)	3	3752	36	3742.2	184	3.53	0.12	34	3740.8	480	8.36	0.22
	4	50382	23	50368.5	96	3.92	0.03	50	50382.0	11	0.09	0.01
	5	61494	50	61494.0	1	0.01	0.01	50	61494.0	1	0.01	0.00
	6	36360	50	36360.0	26	0.21	0.06	50	36360.0	159	1.63	0.14
	7	14657	50	14657.0	9	0.09	0.05	50	14657.0	55	0.79	0.05
	8	20452	50	20452.0	8	0.08	0.05	50	20452.0	21	0.29	0.09
	9	35438	37	35419.4	235	3.10	0.04	11	35381.4	873	8.22	1.26
	10	24930	50	24930.0	11	0.10	0.07	42	24917.5	249	2.84	0.16
Average			45		62	1.14	0.05	40		267	3.29	0.25
200	1	937149	50	937149.0	469	22.72	0.80	50	937149.0	27	1.27	0.45
($d =$	2	303058	50	303058.0	103	6.12	1.95	50	303058.0	33	4.54	2.34
1.00)	3	29367	50	29367.0	19	1.37	0.90	50	29367.0	12	2.50	0.48
	4	100838	50	100838.0	20	1.47	0.92	50	100838.0	19	3.45	1.42
	5	786635	50	786635.0	49	2.61	0.95	50	786635.0	14	0.92	0.50
	6	41171	50	41171.0	13	1.01	0.83	50	41171.0	4	0.68	0.26
	7	701094	50	701094.0	196	10.25	1.12	50	701094.0	69	5.23	1.17
	8	782443	6	782398.1	1571	98.23	54.50	50	782443.0	48	3.08	1.20
	9	628992	50	628992.0	66	3.65	0.98	50	628992.0	30	2.51	1.25
	10	378442	50	378442.0	179	10.31	2.47	50	378442.0	57	7.09	2.23
Average			46		269	15.77	6.54	50		31	3.13	1.13

We also compare S-bAFSA with the algorithms B&B and Mini-Swarm described in [3, 26] respectively, using the entire set of 80 instances. The comparison with the B&B algorithm is included to show the difficulty in solving even moderately sized instances, in terms of computational time. On the other hand, the Mini-Swarm algorithm is a heuristic that also relies on operators from evolutionary algorithms, and it is probably one of the most effective for solving QKP. Table 5 summarizes the results in terms of average over the 10 instances of each set. Besides 'aT', 'aAT', the table also depicts 'aSR', and 'aBT'. We observe that S-bAFSA is outperformed in both criteria 'aSR' and 'aAT' by Mini-Swarm, in particular when solving the set of largest problems. We note that, during this comparison, $T_{\max}$ was set to 500 and an extra condition was added to the termination conditions in S-bAFSA to match those reported in [26]: the algorithm stops if there is no improvement in f throughout 100 consecutive iterations. Consequently, the percentage of successful runs has decreased when compared with the results of Table 4, although the average time has improved.

Table 5: Comparative results of B&B, Mini-Swarm and S-bAFSA

Prob.		B&B	Mini-Swarm		S-bAFSA 50 runs		S-bAFSA Succ. runs	
n	d	aT	aAT	aSR	aAT	aSR	aAT	aBT
100	0.25	117	0.442	93.9	0.702	71.4	0.549	0.336
	0.50	82	0.406	94.2	0.583	79.0	0.506	0.129
	0.75	120	0.363	97.5	0.376	90.6	0.335	0.098
	1.00	190	0.225	100.0	0.231	96.8	0.209	0.059
200	0.25	3602	1.430	90.3	12.323	55.8	12.986	9.478
	0.50	1690	1.805	92.4	8.882	61.8	7.085	3.046
	0.75	–	2.165	90.9	7.270	81.2	6.463	2.695
	1.00	–	1.197	100.0	3.047	99.4	3.013	1.266

– Not solved within 30000 sec. [3]

Finally, we compare S-bAFSA with a novel global harmony search algorithm, NGHS [27], using a set of ten 0–1 LKP (see Table 6). NGHS used the penalty function method for handling the knapsack constraint. Problems data and results of NGHS are described in [27]. For a fair comparison with NGHS, we set in S-bAFSA $N = 5$ and $T_{\max} = 10000$. We may observe that S-bAFSA shows very competitive results when compared with NGHS.

4. Conclusions

In this paper, a new binary version of the artificial fish swarm algorithm for solving 0–1 quadratic knapsack problems as well as problems with linear objective function is presented. In the new version, denoted by S-bAFSA, random, searching and chasing behavior are used to move the points according to two target probability values. To create the trial points, crossover

Table 6: Comparative results of NGHS and S-bAFSA

Prob.			NGHS		S-bAFSA		Prob.			NGHS		S-bAFSA	
No.	n	f_{opt}	AIT	AT	AIT	AT	No.	n	f_{opt}	AIT	AT	AIT	AT
f_1	10	295	263	0.0093	28	0.0017	f_6	10	52*	235	0.0052	1	0.0001
f_2	20	1024	754	0.0293	67	0.0058	f_7	7	107	325	0.0087	19	0.0010
f_3	4	35	11	0.0005	1	0.0000	f_8	23	9767	1727	0.0617	89	0.0075
f_4	4	23	13	0.0006	3	0.0002	f_9	5	130	29	0.0023	1	0.0001
f_5	15	481.07	579	0.0210	11	0.0008	f_{10}	20	1025	831	0.0307	64	0.0056

* NGHS reported optimal value 50

and mutation are implemented. A random heuristic drop_item algorithm and an add_item operation are used to make the points feasible and improve the quality of the solutions. To enhance the search for an optimal solution, a swap move heuristic search and a cyclic reinitialization of the population are also implemented. Numerical experiments (with a set of well-known 0–1 QKP and LKP) show that our proposals to reduce computational effort in terms of number of iterations and execution time need further developments. Some work remains to be done in order to accelerate convergence and reduce time. Since the performance of S-bAFSA is very competitive when solving 0–1 LKP, a linearization technique that involves the addition of new variables and linking constraints may be applied to the QKP and then hybridized with the heuristic S-bAFSA. This type of formulation has been successfully tested in the past, see for example [2, 23], although our goal is to address the mixed integer linear programming problem using S-bAFSA.

Furthermore, work is already under way for using a strategy related to vanishing points throughout a few iterations and re-creating them again later on in a different place of the search space, so that computational requirements could be reduced. Future work will consider using S-bAFSA to solve multidimensional knapsack problems effectively. Other NP-hard challenging combinatorial optimization problems, like the uncapacitated facility location problem and the resource-constrained project scheduling problem will be also addressed in the future.

Acknowledgements

The authors wish to thank the reviewers for their valuable comments and suggestions. The first author acknowledges Ciência 2007 of FCT (Foundation for Science and Technology), Portugal for the fellowship grant: C2007-UMINHO-ALGORITMI-04. Financial support from FEDER COMPETE (Operational Programme Thematic Factors of Competitiveness) and FCT under project: FCOMP-01-0124-FEDER-022674 is also acknowledged.

References

[1] M.A.K. Azad, A.M.A.C. Rocha, E.M.G.P Fernandes, Solving multidimensional 0-1 knapsack problem with an artificial fish swarm algorithm, in: B. Murgante et al. (Eds.), Computational Science and Its Applications, ICCSA 2012, Part III, LNCS, vol. 7335, Springer-Verlag, Heidelberg, pp. 72–86.

[2] A. Billionnet, Éric Soutif, Using a mixed integer programming tool for solving the 0–1 quadratic knapsack problem, INFORMS J. Comput., 16 (2004) 188–197.

[3] A. Billionnet, Éric Soutif, An exact method based on Lagrangian decomposition for the 0–1 quadratic knapsack problem, Eur. J. Oper. Res., 157 (2004) 565–575.

[4] A. Billionnet, A. Faye, Éric Soutif, A new upper bound for the 0–1 quadratic knapsack problem, Eur. J. Oper. Res., 112 (1999) 664–672.

[5] A. Caprara, D. Pisinger, P. Toth, Exact solution of the quadratic knapsack problem, INFORMS J. Comput., 11 (1999) 125–137.

[6] G. Gallo, P. L. Hammer, B. Simeone, Quadratic knapsack problems, Math. Program. Study, 12 (1980) 132–149.

[7] F. Glover, G. Kochenberger, Solving quadratic knapsack problems by reformulation and tabu search, single constraint case, in: P.M. Pardalos et al. (eds.) Combinatorial and Global Optimization, vol. 14, World Scientific, Singapore, pp. 111–121 (2002).

[8] S. Gueye, Ph. Michelon, Miniaturized linearizations for quadratic 0/1 problems, Ann. of Oper. Res., 140 (2005) 235–261.

[9] P.L. Hammer, D.J. Rader Jr., Efficient methods for solving quadratic 0–1 knapsack problems, INFOR, 35(3) (1997) 170–182.

[10] C. Helmberg, F. Rendl, Solving quadratic (0,1) problems by semidefinite programs and cutting planes, Math. Program., 82 (1998) 291–315.

[11] M. Jiang, N. Mastorakis, D. Yuan, M. A. Lagunas, Image segmentation with improved artificial fish swarm algorithm, in: N. Mastorakis et al. (eds.) ECC 2008, LNEE, vol. 28, Springer-Verlag, Heidelberg, pp. 133–138.

[12] M. Jiang, Y. Wang, S. Pfletschinger, M. A. Lagunas, D. Yuan, Optimal multiuser detection with artificial fish swarm algorithm, in: D.-S. Huang et al. (eds.), Advanced Intelligent Computing Theories and Applications–ICIC 2007, Part 22, CCIS vol. 2, Springer-Verlag, Heidelberg, pp. 1084–1093.

[13] B. A. Julstrom, Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem, in: Proceedings of the GECCO'05, pp. 607–614 (2005).

[14] M.H. Kashan, N. Nahavandi, A.H. Kashan, *DisABC*: A new artificial bee colony algorithm for binary optimization, Appl. Soft Comput. 12 (2012) 342–352.

[15] L. Létocart, A. Nagih, G. Plateau, Reoptimization in Lagrangian methods for the 0–1 quadratic knapsack problem, Comput. Oper. Res. 39 (2012) 12–18.

[16] Z. Michalewicz, Genetic Algorithms+Data Structures=Evolution Programs, Springer, Berlin, 1996.

[17] A. Narayan, C. Patvardhan, A novel quantum evolutionary algorithm for quadratic knapsack problem, in: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics–SMC , pp. 1388-1392, (2009).

[18] M. Neshat, G. Sepidnam, M. Sargolzaei, A.N. Toosi, Artificial fish swarm algorithm: a survey of the state-of-the-art, hybridization, combinatorial and indicative applications, Artif. Intell. Rev. (2012). DOI:10.1007/s10462-012-9342-2

[19] D. Pisinger, The quadratic knapsack problem–a survey, Discrete Appl. Math. 155 (2007) 623–648.

[20] W. D. Pisinger, A.B. Rasmussen, R. Sandvik, Solution of large quadratic knapsack problems through aggressive reduction, INFORMS J. Comput., 19 (2007) 280–290.

[21] A. M. A. C. Rocha, T. F. M. C. Martins, E. M. G. P. Fernandes, An augmented Lagrangian fish swarm based method for global optimization, J. Comput. Appl. Math., 235 (2011) 4611–4620.

[22] A. M. A. C. Rocha, E. M. G. P. Fernandes, T. F. M. C. Martins, Novel fish swarm heuristics for bound constrained global optimization problems, in: B. Murgante et al. (eds.) Computational Science and Its Applications–ICCSA 2011, Part III, LNCS, vol. 6784, Springer-Verlag, Heidelberg, pp. 185–199.

[23] C. D. Rodrigues, D. Quadri, P. Michelon, S. Gueye, 0–1 quadratic knapsack problems: An exact approach based on a t-linearization, SIAM J. Optim., 22(4) (2012) 1449–1468.

[24] C.-R. Wang, C.-L. Zhou, J.-W. Ma, An improved artificial fish swarm algorithm and its application in feed-forward neural networks, in: Proceedings of the 4th ICMLC, pp. 2890–2894 (2005).

[25] X. Wang, N. Gao, S. Cai, M. Huang, An artificial fish swarm algorithm based and ABC supported QoS unicast routing scheme in NGI, in: G. Min et al. (eds.) ISPA 2006, LNCS, vol. 4331, Springer-Verlag, Heidelberg, pp. 205–214.

[26] X. -F. Xie, J. Liu, A Mini-swarm for the quadratic knapsack problem, in: Proceedings of the IEEE Swarm Intelligence Symposium, pp. 190–197 (2007).

[27] D. Zou, L. Gao, S. Li, Z. Wu, Solving 0–1 knapsack problem by a novel global harmony search algorithm. Appl. Soft Comput. 11 (2011) 1556–1564.