International Journal of Innovative
Computing, Information and Control
Volume **13**, Number **6**, December **2017**

 ISSN 1349-4198
pp. **2039–2049**

A NOVEL BINARY SOCIAL SPIDER ALGORITHM FOR 0-1 KNAPSACK PROBLEM

Phuong Hoai Nguyen[1,2], Dong Wang[1] and Tung Khac Truong[3,4,*]

[1]College of Computer Science and Electronic Engineering
Hunan University
Lushan South Road, Yuelu Dist., Changsha 410082, P. R. China
phuongvn2999@gmail.com; d.wang@hnu.edu.cn

[2]Department of Computer Science
University of Labour and Social Affairs
43 Tran Duy Hung Street, Trung Hoa Ward, Cau Giay Dist., Hanoi 100000, Vietnam

[3]Division of Computational Mathematics and Engineering
Institute for Computational Science
[4]Faculty of Civil Engineering
Ton Duc Thang University
19 Nguyen Huu Tho Street, Tan Phong Ward, District 7, Ho Chi Minh City 700000, Vietnam
[*]Corresponding author: truongkhactung@tdt.edu.vn

Received March 2017; revised July 2017

Abstract. *The 0-1 knapsack problem (KP01) is a well-known NP-hard optimization problem. Recently a new metaheuristic algorithm, called social spider algorithm, was proposed, which has been successfully applied to solving various continuous optimization problems. This paper proposes a binary social spider algorithm to solve KP01 efficiently. This algorithm is composed of discrete process and constraint handling process. In discrete process, a popular sigmoid function is used to achieve good discrete process result. Two constraint handling techniques are utilized. The repair operator with ADD phase and DROP phase is executed to treat infeasibility and improve the efficiency. The experimental results have proven the superior performance of BSSA compared to genetic algorithm and particle swarm optimization.*
Keywords: Combinatorial optimization, Greedy strategy, 0-1 knapsack problem, SSA

1. **Introduction.** The 0-1 knapsack problem (KP01) is known to be a combinatorial optimization problem. The knapsack problem has a variety of practical applications such as cutting stock problems, portfolio optimization, scheduling problems [1] and cryptography [2, 3]. The knapsack appears as a sub-problem in many complex mathematical models of real-world problems. In a given set of n items, each of them has an integer weight w_i and an integer profit p_i. The problem is to select a subset from the set of n items such that the overall profit is maximized without exceeding a given weight capacity C. It is an NP-hard problem and hence it does not have a polynomial time algorithm unless $P = NP$. The problem may be mathematically modelled as follows:

$$\text{Maximize} \quad \sum_{i=1}^{n} x_i p_i; \tag{1}$$

$$\text{Subject to} \quad \sum_{i=1}^{n} x_i w_i \leq C,\ x_i \in \{0, 1\},$$

$\forall i \in \{1, 2, \ldots, n\}$, where x_i takes values either 1 or 0 which represents the selection or rejection of the ith item.

In early literature, the exact algorithms are often used, for example, an exact algorithm for solving the bi-level KP01. The linear relaxation approach is used to calculate feasible solutions [4]. Moreover, a dynamic programming procedure is used to find integer optimal solution. The proposed approach shows superiority when compared with other state-of-the-art approaches. Furthermore, utilizing dynamic programming and device discounted KP01 into sub problems conducted potential results [5].

In recent years, many new nature-inspired metaheuristic algorithms have been proposed to solve KP01. The main advantage of metaheuristics is that approximate solutions can usually be found within reasonable time. The genetic algorithm (GA) has been one of the most popular approaches for this problem, for instance, dual population GA by introducing greedy approach and sub-group competition to solve the KP01. Nonetheless, the tests are not extensive enough to conclude the efficacy of the proposed methodology.

The other metaheuristics used for the KP01 include, but not limited to the ant colony optimization [6], quantum-inspired evolutionary algorithm [7], schema-guiding evolutionary algorithm [8], global harmony search algorithm [9], artificial chemical reaction optimization [10], complex-valued encoding bat algorithm [11], monarch butterfly optimization [12], amoeboid organism algorithm [13], Cohort intelligence algorithm [14], monkey algorithm [15], and complex-valued encoding wind driven optimization [16].

Although variety of methods have been proposed to solve KP01, it is still meaningful in theoretical research and practical application. Due to the difficulty of an NP-hard problem, the proposed methods can work well for small dimensional KP01 problems. For large-scale KP01 problems, there still exists the gap between the found solutions and the optimal solutions of the problems. Development novel algorithm to solve KP01 efficiently is necessary.

Social spider algorithm (SSA) is a new algorithm proposed by Yu and Li for global optimization [17]. SSA inspired the foraging behaviour of the social spider that can be described as the cooperative movement of the spiders towards the food source position. SSA has outperformed other state-of-the-art metaheuristics on many benchmark functions.

In this paper, a novel binary social spider algorithm is proposed to solve KP01. The main contribution of this study is that the first binary social spider algorithm combining with two constraint handling techniques for KP01 is proposed. The proposed algorithm integrated the exploration of SSA and the exploitation of a repair operator to solve KP01. The simulation results on five state-of-the-art benchmark instances and strongly correlated data sets demonstrate that the proposed algorithm has superior performance compared with previous algorithms.

The rest of this paper is organized in sections. Section 2 briefly gives the original framework of social spider algorithm. Section 3 presents the binary social spider algorithm. We survey the behavior of binary social spider algorithm and compare the simulated results of the BSSA in Section 4. We conclude this paper and suggest potential future work in Section 5.

2. **Social Spider Algorithm.**

SSA [17] is a metaheristic inspired by the behavior of social spider. In SSA, the solution of an optimization problem is a simulation by a position of an artificial spider on the hyper-dimension spider web. The spiders move on the web while they share the position information via vibration of it. Depending on the received vibration from other ones, guide the spider forward to the optimal position. Details of SSA will be described in the following subsections.

2.1. **Spider.** The artificial spiders are the basic operating agents of SSA. Each spider possesses a position on the hyper-dimension spider web, and the fitness value of this position is assigned to the spider. Each spider hold a memory is storing its status as well as optimization parameters, namely, its current position, a current fitness value, following vibration at previous iteration, inactive degree, previous movements, and dimension mask. All these characters guide the spider to search for the optimal solution.

2.2. **Vibration.** Vibration is a very important concept in SSA. It is one of the main characteristics that distinguish SSA from other metaheuristics. In SSA, we use two properties to define a vibration, namely, the source position and the source intensity of the vibration.

The source vibration intensity is calculated by the following equation:

$$I(s) = \log\left(\frac{1}{f(s) - C} + 1\right) \tag{2}$$

where $f(s)$ is the fitness value of spider s, and C is a given constant such that minimum fitness values are larger than C.

Vibration attenuation when transmitting from spider s to spider s' is calculated as the following equation:

$$I(s, s') = I(s) \times \exp\left(\frac{D(s, s')}{\overline{\sigma} \times r_a}\right) \tag{3}$$

where $r_a \in (0; \infty)$ is a user-controlled parameter. This parameter controls the attenuation rate of the vibration intensity over distance. $\overline{\sigma}$ is the standard deviation of all spider positions along each dimension.

2.3. **Search pattern.** SSA manipulates a population of artificial spiders via a series of optimization step. Specifically, each iteration of SSA can be divided into the following steps.

1) Fitness Evaluation: At the beginning of each iteration, the fitness values of the positions possessed of all spiders in the population are reevaluated. These fitness values are later utilized in the vibration generation and propagation process.

2) Vibration Generation: In the beginning, new vibration is generated for each spider. Vibration is then propagated to all the other spiders in the population with attenuation. Depend on the receipt vibrations; the largest attenuated vibration intensity is selected, and compare it with the previous one. The larger intensity vibration is stored. If the spider changes its stored vibration, the inactive degree is increased by one. Otherwise, the inactive degree is assigned to zero. This degree helps the algorithm get of local optima.

3) Mask Changing: After following vibration of all spiders is calculated, the position of spiders is updated. In this step, a dimension mask is used. Each spider holding mask which is a binary vector with length is the solution dimension of the optimization problem. In each iteration, each spider has a probability $1 - p_c^{N_{in}}$ to change its mask. The N_{in} is the inactive number of the spider. When a mask is considered to change, each bit has probability p_m to get a one, and a probability $1 - p_m$ to get a zero.

4) Random Walk: After following vibration and dimension mask are calculated, each spider performs a random walk to update their positions. A more detailed formulation can be found in [17].

3. **Proposed Social Spider Algorithm (BSSA) for KP01.** SSA is designed for the real domain. For 0/1 knapsack problem, the solution is presented in a binary vector. So SSA is modified to work for discrete binary space.

3.1. **Transfer function.** In this algorithm, the sigmoid function is used to convert real values to binary values, after the random walk performing the position in the real vector is converted to binary vector by Equation (4).

$$X_{s,i}(t+1) = \begin{cases} 0 & \text{if } rand() \geq S(P_{s,i}(t+1)) \\ 1 & \text{if } rand() < S(P_{s,i}(t+1)) \end{cases} \tag{4}$$

where $S(.)$ is the sigmoid function for transforming the velocity to the probability as the following expression:

$$S(P_{s,i}(t+1)) = \frac{1}{1+e^{(P_{s,i}(t+1))}} \tag{5}$$

Figure 1 shows the sigmoid function using in BSSA.

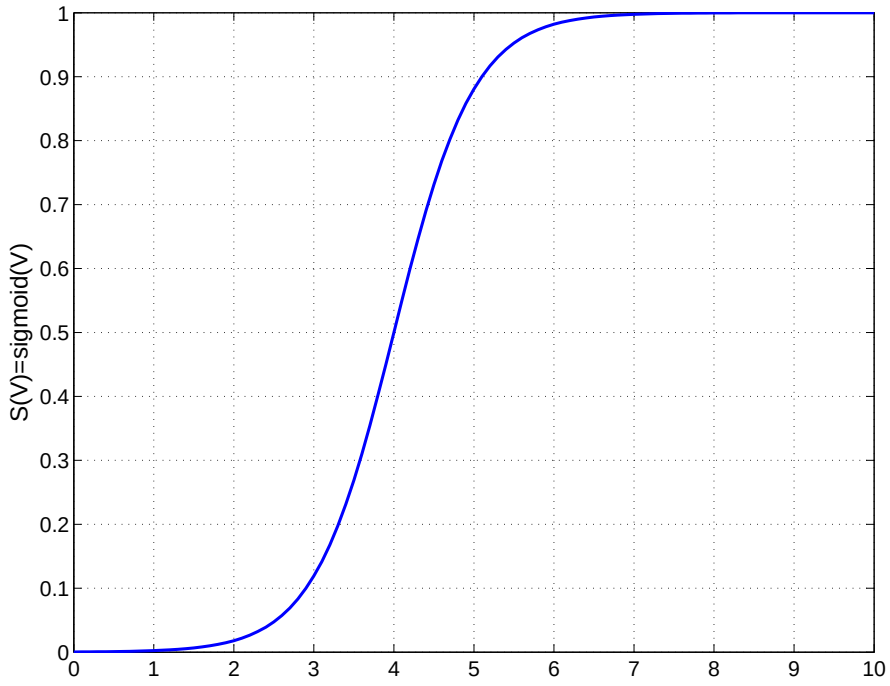

FIGURE 1. Sigmoid function used in BSSA

3.2. **Fitness function and handling constraints.** The KP01 is maximization problem. However, SSA is designed for a minimization problem. To convert KP01 problem to minimization problem a large position constant number Ω is used, and the fitness is set as follows:

$$\text{Fitness} = \Omega - \sum_{i=1}^{n} x_i p_i \tag{6}$$

The present by binary string sometimes makes the solution violate the constraint. There are two common techniques that are penalty and repair function which are used to handle it. In the first method, a penalty coefficient ratio with violated value is used to add to the fitness value. Through the iterations, the solutions with larger fitness have more change to reproduce and otherwise [9]. Although this method can help the algorithm find the sufficient solution, it does not help improve the quality of the solution. Following, two techniques are presented in detail.

3.2.1. *Penalty function.* The value of the position is equal to $\sum_{i=1}^{n} x_i p_i$ when the solution is not violated. Otherwise, a penalty factor γ is used to decrease the fitness of the violate position. The penalty factor γ is selected by experience. When we choose a large γ, the violate solution receives large probability of eliminating. In this research, we use $\gamma = 100$. The fitness function is described in Algorithm 1.

Algorithm 1: Fitness function

Input: Solution x
Output: *Fitness*
$Fitness = \sum_{i=1}^{n} x_i p_i - \gamma * \max(0, \sum_{i=1}^{n} x_i w_i - C)$
return *Fitness*

3.2.2. *Repair operator.* The repair operator includes two phases: DROP phase and ADD phase. When the total weight exceeds the knapsack, the DROP is used. The ADD phase works to improve the quality of the solution when the knapsack is not full. The details of this function can be found in Algorithm 2.

Algorithm 2: Repair operator

Input: Solution x
Output: Solution x
% ADD phase
$gap \leftarrow C - \sum_{i=1}^{n} x_i w_i$
while *(*$gap > 0$*)* **do**
  Select the feasible items to add to knapsack.
% DROP phase
$outweigh \leftarrow \sum_{i=1}^{n} x_i w_i - C$
while *(*$outweigh > 0$*)* **do**
  Randomly select items in the knapsack to drop.

The advance of repair operator when compared to penalty function is that the repair function not only repairs the violate solutions, but also helps improve the quality of potential solutions.

4. **Simulation Results.** In our experiments, we test effectiveness of two handle techniques (penalty function and repair function) when combined with PSO, GA and BSSA. The GA is coded as describing in [18]. We named GA1, and GA2 for genetic algorithm with penalty function, and repair function, respectively. The population size is $popsize = 10$, the crossover probability is set as $P_c = 0.8$, and the mutation probability is set as $P_m = 0.1$.

For the particle swarm optimization, BPSO1, and BPSO2 are PSO combined with penalty function, and repair function, respectively. The parameters for BPSO1 and BPSO2 are set as: inertia weight $w = 2$, local weight $c_1 = 2$, global weight $c_2 = 2$.

We named BSSA1 and BSSA2 for binary social spider algorithm with penalty function, and repair function, respectively. For the BSSA1 and BSSA2, the parameters are turning by trial and error. The parameters are set as: $r_a = 1$, $p_c = 0.7$, $p_m = 0.1$, and $popsize = 10$.

TABLE 1. The dimension and parameters of five test problems

Instance	Dimension	Parameters (q, C, p)
f_1	4	q = (6, 5, 9, 7), C = 20, p = (9, 11, 13,15)
f_2	10	q = (30, 25, 20, 18, 17, 11, 5, 2, 1, 1), C = 60, p = (20, 18, 17, 15, 15, 10, 5, 3, 1, 1)
f_3	7	q = (31, 10, 20, 19, 4, 3, 6), C = 50, p = (70, 20, 39, 37, 7, 5, 10)
f_4	5	q = (15, 20, 17, 8, 31), C = 80, p = (33, 24, 36, 37, 12)
f_5	20	q = (84, 83, 43, 4, 44, 6, 82, 92, 25, 83, 56, 18, 58, 14, 48, 70, 96, 32, 68, 92), C = 879, p = (91, 72, 90, 46, 55, 8, 35, 75, 61, 15, 77, 40, 63, 75, 29, 75, 17, 78, 40, 44)

TABLE 2. The detailed information of the optimal solutions

Instance	Opt. solution x^*	Opt. value $f(x^*)$	The gap of knapsack
f_1	$(1, 1, 0, 1)$	35	2
f_2	$(0, 0, 1, 0, 1, 1, 1, 1, 1, 1)$	52	3
f_3	$(1, 0, 0, 1, 0, 0, 0)$	107	0
f_4	$(1, 1, 1, 1, 0)$	130	20
f_5	$(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1)$	1025	8

The performances of the BSSA1 and BSSA2 algorithms are extensively investigated by a large number of experimental studies. Nine 0-1 knapsack instances are considered to testify the validity of the BSSA.

All the algorithms are implemented in Matlab 2014a. The test environment is set up on a Desktop with Core i5 3340 CPU at 3.1 GHz, 4G RAM, running on Windows 10 (64 bit).

4.1. **The performance of three algorithms on solving 0-1 knapsack problems with small dimension sizes.** In this section, five test functions collected from [9] are used. In Table 1, five test functions with dimension are 4, 10, 7, 5, and 20, respectively. Table 2 describes the optimal solutions of each function.

The experiment for these five test functions is run 30 independent times. For a fare comparison, we adopt the same termination criterion, and the function evaluation limit is set to 100000. Table 3 shows the experience results of six algorithms. In this test, the algorithms BSSA2, BPSO1, and BPSO2 outperform the other algorithms.

To extensively study the performance of BSSA2, four strong correlated instances with large dimension are also used.

4.2. **The performance of three algorithms for solving 0-1 knapsack problems with large dimension sizes.** To test the performance of BSSA on KP01 with large dimension, it is compared with both BPSO and GA on the 0-1 knapsack problem. In

TABLE 3. Experimental results: the test functions f_1-f_5, the maximum number evaluation 100000, the number of runs 30

Instance	Algorithm	Best	Worst	Mean	Std
f_1	BSSA1	35	35	35.00	0.00
	BSSA2	35	35	35.00	0.00
	BPSO1	35	35	35.00	0.00
	BPSO2	35	35	35.00	0.00
	GA1	35	33	34.60	0.81
	GA2	35	35	35.00	0.00
f_2	BSSA1	52	52	52.00	0.00
	BSSA2	52	52	52.00	0.00
	BPSO1	52	52	52.00	0.00
	BPSO2	52	52	52.00	0.00
	GA1	52	45	51.13	1.63
	GA2	52	50	51.93	0.37
f_3	BSSA1	107	105	106.87	0.51
	BSSA2	107	107	107.00	0.00
	BPSO1	107	107	107.00	0.00
	BPSO2	107	107	107.00	0.00
	GA1	107	81	102.43	5.64
	GA2	107	96	105.67	2.20
f_4	BSSA1	130	130	130.00	0.00
	BSSA2	130	130	130.00	0.00
	BPSO1	130	130	130.00	0.00
	BPSO2	130	130	130.00	0.00
	GA1	130	109	127.30	5.70
	GA2	130	130	130.00	0.00
f_5	BSSA1	1025	1019	1024.80	1.10
	BSSA2	1025	1025	1025.00	0.00
	BPSO1	1025	1025	1025.00	0.00
	BPSO2	1025	1025	1025.00	0.00
	GA1	1025	821	978.67	41.61
	GA2	1025	996	1019.17	7.47

these test cases, strongly correlated sets of data are considered. The test instances are generated as described by Truong et al. in [19].

We do experiment on four test instances with 50, 100, 500 and 1000 items. Figures 2, 3, 4 and 5 show the convergence curves of the best profits of BSSA1, BSSA2, BPSO1, BPSO2, GA1, and GA2 in the four instances. The BSSA2 shows better diversification and intensification when it is fast convergence and finds out the better profit value compared with other ones.

It indicates the global search ability and the convergence ability of BSSA2. BSSA2 outperformed other algorithms in terms of convergence rate and profit amount.

As shown in Figures 2, 3, 4 and 5, the BSSA2 displays no premature convergence in average profits throughout the iterations. The GA1 shows premature convergence compared with BSSA2 in three test instances. The BSSA2 shows better diversification and intensification when it is fast convergence and finds out the better profit value compared with other algorithms.

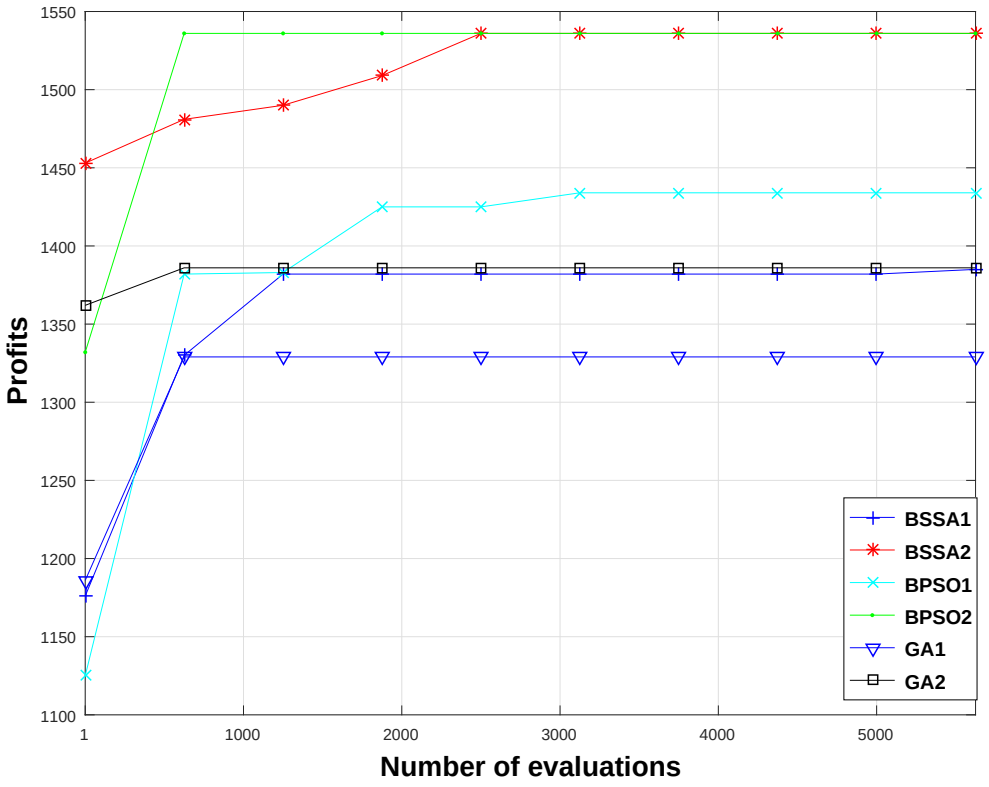

FIGURE 2. The convergence curve test function f_6

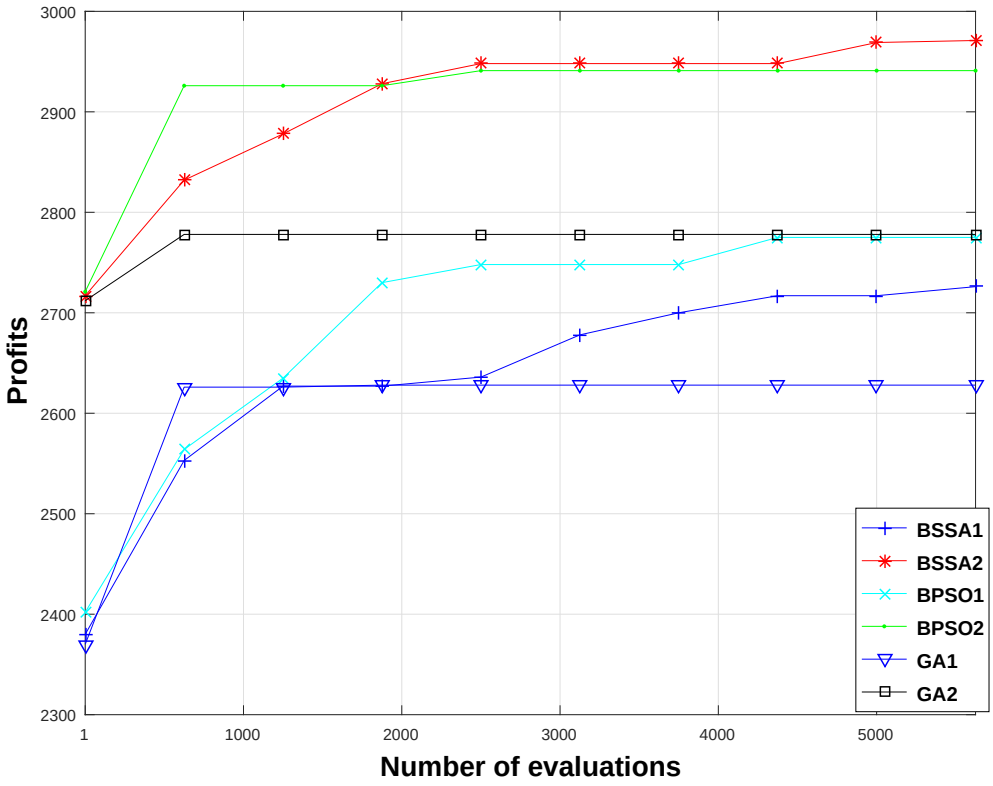

FIGURE 3. The convergence curve test function f_7

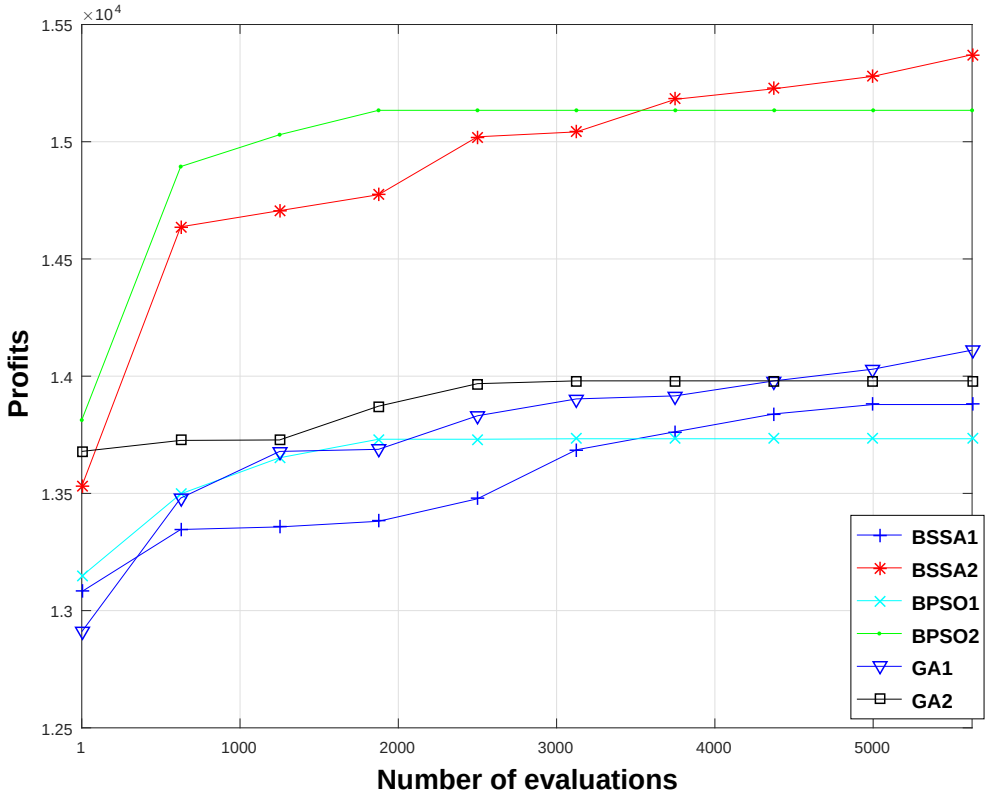

FIGURE 4. The convergence curve test function f_8

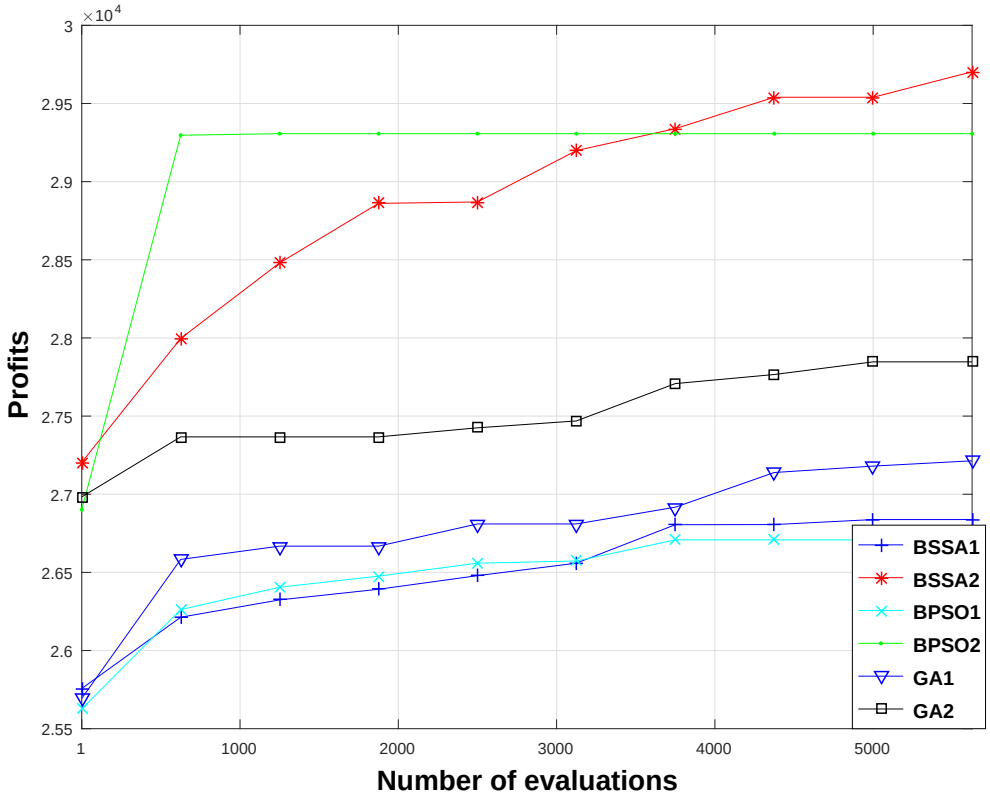

FIGURE 5. The convergence curve test function f_9

TABLE 4. Experimental results: the test functions f_6-f_9, the maximum number evaluation 100000, the number of runs 30

Instance	Algorithm	Best	Worst	Mean	Std
f_6	BSSA1	1536	1536	1536.00	0.00
	BSSA2	**1536**	**1536**	**1536.00**	0.00
	BPSO1	1536	1486	1531.47	9.45
	BPSO2	1536	1536	1536.00	0.00
	GA1	1435	1230	1320.73	42.09
	GA2	1486	1364	1432.20	38.62
f_7	BSSA1	2978	2928	2975.30	10.57
	BSSA2	**2978**	**2977**	**2977.97**	0.18
	BPSO1	2886	2852	2874.10	6.55
	BPSO2	2978	2978	2978.00	0.00
	GA1	2728	2428	2576.33	66.28
	GA2	2828	2628	2744.67	53.07
f_8	BSSA1	15781	15190	15742.47	107.85
	BSSA2	**15781**	**15631**	**15758.10**	37.54
	BPSO1	14341	14109	14199.67	54.81
	BPSO2	15531	15369	15432.43	33.66
	GA1	15131	14631	14863.63	112.15
	GA2	15129	14631	14791.30	124.35
f_9	BSSA1	31419	29527	31239.27	358.36
	BSSA2	**31419**	**30926**	**31310.63**	130.61
	BPSO1	27660	27335	27453.03	81.29
	BPSO2	30304	29954	30137.27	69.33
	GA1	29245	28569	28919.00	177.83
	GA2	29162	28567	28875.13	162.86

Table 4 shows the experimental results of the instances. We adopt the same termination criterion, and the function evaluation limit is set to 100000, for all the tests. For all the instances, the BSSA2 yields superior results compared with the other ones. The series of experimental results demonstrate the superiority and effectiveness of BSSA2. The experimental results show that BSSA2 outperforms the other algorithms in solution quality. The reason for this superior performance of BSSA2 is that our proposed algorithm has a good search ability and a greedy repair operator.

5. **Conclusion.** In this paper, a new algorithm has been proposed based on the binary social spider algorithm with a greedy to solve 0-1 knapsack problem efficiently. Two constraint techniques based on penalty factor and greedy strategy are proposed to improve the efficiency of the proposed algorithm. The simulation results on five state-of-the-art benchmark instances and strong correlated data sets demonstrate that the proposed algorithm has superior performance compared with previous algorithms.

Acknowledgement. This work was partly supported by National Natural Science Foundations of China (No. 61301148 and No. 61272061), the Fundamental Research Funds for the Central Universities of China (No. 531107040263, 345531107040276), the Research Funds for the Doctoral Program of Higher Education of China (No. 20120161110002 and No. 20130161120019), Hunan Natural Science Foundation of China (No. 14JJ7023).

REFERENCES

[1] S. Martello, *Knapsack Problem: Algorithms and Computer Implementations*, John Wiley and Sons, New York, 1990.

[2] B. Chor and R. Rivest, A knapsack-type public key cryptosystem based on arithmetic in finite fields, *IEEE Trans. Information Theory*, vol.34, no.5, pp.901-909, 1988.

[3] C.-S. Laih, J.-Y. Lee, L. Harn and Y.-K. Su, Linearly shift knapsack public-key cryptosystem, *IEEE Journal on Selected Areas in Communications*, vol.7, no.4, pp.534-539, 1989.

[4] R. Mansi, C. Alves, J. Valério de Carvalho and S. Hanafi, An exact algorithm for bilevel 0-1 knapsack problems, *Mathematical Problems in Engineering*, 2012.

[5] A. Rong, J. R. Figueira and K. Klamroth, Dynamic programming based algorithms for the discounted {0-1} knapsack problem, *Applied Mathematics and Computation*, vol.218, no.12, pp.6921-6933, 2012.

[6] L. He and Y. Huang, Research of ant colony algorithm and the application of 0-1 knapsack, *The 6th International Conference on Computer Science Education*, pp.464-467, 2011.

[7] K.-H. Han and J.-H. Kim, Quantum-inspired evolutionary algorithm for a class of combinatorial optimization, *IEEE Trans. Evolutionary Computation*, vol.6, no.6, pp.580-593, 2002.

[8] Y. Liu and C. Liu, A schema-guiding evolutionary algorithm for 0-1 knapsack problem, *International Association of Computer Science and Information Technology – Spring Conference*, pp.160-164, 2009.

[9] D. Zou, L. Gao, S. Li and J. Wu, Solving 0-1 knapsack problem by a novel global harmony search algorithm, *Applied Soft Computing*, vol.11, no.2, pp.1556-1564, 2011.

[10] T. K. Truong, K. Li, Y. Xu, A. Ouyang and T. T. Nguyen, Solving 0-1 knapsack problem by artificial chemical reaction optimization algorithm with a greedy strategy, *Journal of Intelligent and Fuzzy Systems*, vol.8, no.5, pp.2179-2186, 2015.

[11] Y. Zhou, L. Li and M. Ma, A complex-valued encoding bat algorithm for solving 0-1 knapsack problem, *Neural Processing Letters*, vol.44, no.2, pp.407-430, 2016.

[12] Y. Feng, G.-G. Wang, S. Deb, M. Lu and X.-J. Zhao, Solving 0-1 knapsack problem by a novel binary monarch butterfly optimization, *Neural Computing and Applications*, vol.28, no.7, pp.1619-1634, 2017.

[13] X. Zhang, S. Huang, Y. Hu, Y. Zhang, S. Mahadevan and Y. Deng, Solving 0-1 knapsack problems based on amoeboid organism algorithm, *Applied Mathematics and Computation*, vol.219, no.19, pp.9959-9970, 2013.

[14] A. J. Kulkarni and H. Shabir, Solving 0-1 knapsack problem using cohort intelligence algorithm, *International Journal of Machine Learning and Cybernetics*, vol.7, no.3, pp.427-441, 2016.

[15] Y. Zhou, X. Chen and G. Zhou, An improved monkey algorithm for a 0-1 knapsack problem, *Applied Soft Computing*, vol.38, pp.817-830, 2016.

[16] Y. Zhou, Z. Bao, Q. Luo and S. Zhang, A complex-valued encoding wind driven optimization for the 0-1 knapsack problem, *Applied Intelligence*, vol.46, no.3, pp.684-702, 2017.

[17] J. J. Yu and V. O. Li, A social spider algorithm for global optimization, *Applied Soft Computing*, vol.30, pp.614-627, 2015.

[18] R. Singh, Solving 0-1 knapsack problem using genetic algorithms, *IEEE the 3rd International Conference on Communication Software and Networks (ICCSN)*, pp.591-595, 2011.

[19] T. K. Truong, K. Li and Y. Xu, Chemical reaction optimization with greedy strategy for the 0-1 knapsack problem, *Applied Soft Computing*, vol.13, no.4, pp.1774-1780, 2013.