Solving Integer Programming Problems by Using Artificial Bee Colony Algorithm

Bahriye Akay and Dervis Karaboga

Erciyes University, The Dept. of Computer Engineering,
38039, Melikgazi, Kayseri, Turkiye
bahriye@erciyes.edu.tr, karaboga@erciyes.edu.tr

Abstract. This paper presents a study that applies the Artificial Bee Colony algorithm to integer programming problems and compares its performance with those of Particle Swarm Optimization algorithm variants and Branch and Bound technique presented to the literature. In order to cope with integer programming problems, in neighbour solution production unit, solutions are truncated to the nearest integer values. The experimental results show that Artificial Bee Colony algorithm can handle integer programming problems efficiently and Artificial Bee Colony algorithm can be considered to be very robust by the statistics calculated such as mean, median, standard deviation.

1 Introduction

Integer programming is a form of linear programming in which the variables are required to have integer values. Capital budgeting, portfolio analysis, network and VLSI circuit design as well, as automated production systems are some applications in which integer programming problems are met [1, 2]. An integer programming problem in which all variables are integer is called a pure integer programming problem as stated in (1).

$$min\ f(x),\ x \in \mathcal{S} \subseteq Z^n \tag{1}$$

where Z is the set of integers, and $\mathcal{S}$ is the feasible region. If some variables are integer and some are not then the problem is a mixed integer programming problem. The case where the integer variables are 0 or 1 is called pure (mixed) 0-1 programming problems or pure (mixed) binary integer programming problems. The integer programming techniques can be classified into three broad categories: approximate, exact, and heuristic [3]. In approximate techniques, variables to be optimized take on non-integer values and they are rounded to integers. Exact techniques such as Branch and Bound, dynamic programming divide the feasible region into smaller sub-regions or problem into sub-problems. However, they have high computational cost since they explore a search tree containing hundreds or more nodes on large-scale real-life problems [3, 4]. Branch and bound algorithms are also subjected to linear constraints with an objective function that need not be linear [3]. Main drawback of the Branch and Bound technique is that it

R. Serra and R. Cucchiara (Eds.): AI*IA 2009, LNAI 5883, pp. 355–364, 2009.

requires partitioning the feasible space accurately. Heuristic techniques give near optimal solutions in a reasonable time.

Heuristic optimization methods can also handle with integer programming problems. Most of them can cope with integer variables efficiently by truncating or rounding the real valued solutions. Particle Swarm Optimization (PSO) algorithm, introduced by Eberhart and Kennedy in 1995 [5], simulating the social behaviour of fish schools and bird flocks; and Artificial Bee Colony (ABC) algorithm, introduced by Karaboga [6], simulating the foraging behaviour of honey bee swarm are both heuristic optimization methods that can handle with integer programming problems. In this paper, results of the ABC algorithm was compared to those of PSO variants, Quantum-Behaved PSO (QPSO); and Branch and Bound tecniques on integer problems.

Rest of the paper is organized as follows: In the second section, Artificial Bee Colony Algorithm, and in the third section, other algorithms regarded in this study which are Particle Swarm Optimization, Quantum-behaved PSO, Branch and Bound technique, are summarized. In Section 4, experimental results are presented and finally it is concluded.

2 Artificial Bee Colony Algorithm

The minimal model of forage selection that leads to the emergence of collective intelligence of honey bee swarms consists of three essential components: food sources, employed foragers, and unemployed foragers, and defines two leading modes of the behaviour: recruitment to a nectar source and abandonment of a source [7]. The quality of a food source depends on many factors such as its proximity to the nest, richness or concentration of energy, and the ease of extracting this energy [7]. Employed foragers are associated with a particular food source which they are currently exploiting or are employed at. They carry information about this particular source, its distance and direction from the nest, and the profitability of the source and share this information with a certain probability by dancing on the dance area. Unemployed foragers look for a food source to exploit. There are two types of unemployed foragers: scouts searching the environment surrounding the nest for new food sources and onlookers waiting in the nest and finding a food source through the information shared by employed foragers.

Artificial Bee Colony algorithm is a recently proposed optimization algorithm that simulates the foraging behaviour of a bee colony [6]. The main steps of the algorithm are as below:

1: Initialize the population of solutions $x_i, i = 1 \ldots SN$
2: Evaluate the population
3: cycle=1
4: **repeat**
5: {**Employed Bees' Phase**}

6: Produce new solutions υ_i for the employed bees by using (2) and evaluate them

$$\upsilon_{ij} = \begin{cases} x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \text{ , } if \quad R_j < MR \\ x_{ij} \qquad\qquad\qquad\qquad \text{, otherwise} \end{cases} \tag{2}$$

where $k \in \{1, 2,..., SN\}$ and $j \in \{1, 2,..., D\}$ are randomly chosen indexes. Although k is determined randomly, it has to be different from i. $\phi_{i,j}$ is a random number between [-1, 1]. It controls the production of neighbour food sources around $x_{i,j}$ and represents the comparison of two food positions visually by a bee. MR is a control parameter of ABC algorithm in the range of [0,1] which controls the number of parameters to be modified.

7: Apply the greedy selection process between υ_i and x_i

8: {**Onlooker Bees' Phase**}

9: Calculate the probability values p_i for the solutions x_i by (3)

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \tag{3}$$

where fit_i is the fitness value of the solution i which is proportional to the nectar amount of the food source in the position i and SN is the number of food sources which is equal to the number of employed bees or onlooker bees.

10: Produce the new solutions υ_i for the onlookers from the solutions x_i selected depending on p_i and evaluate them

11: Apply the greedy selection process between υ_i and x_i

12: {**Scout Bees' Phase**}

13: Determine the abandoned solution for the scout, if exists, and replace it with a new randomly produced solution $x_i$by (4)

$$x_i^j = x_{\min}^j + \text{rand}[0, 1](x_{\max}^j - x_{\min}^j) \tag{4}$$

14: Memorize the best solution achieved so far

15: cycle=cycle+1

16: **until** cycle=MCN

In ABC algorithm employed for integer programming problems, the position of a food source represents a possible solution consisting of integer numbers to the optimization problem and the nectar amount of a food source corresponds to the quality (fitness) of the associated solution. The number of the employed bees or the onlooker bees is equal to the number of solutions in the population. At the first step, the ABC generates a randomly distributed initial population $P(Cycle = 0)$ of SN solutions (food source positions), where SN denotes the size of employed bees or onlooker bees. Each solution x_i $(i = 1, 2, ..., SN)$ is a D-dimensional vector. Here, D is the number of optimization parameters. After initialization, the population of the positions (solutions) is subject to repeated cycles, $Cycle = 1, 2, ..., MCN$, of the search processes of the employed bees, the onlooker bees and scout bees. An employed bee produces a modification on the

position (solution) in her memory depending on the local information (visual information) and tests the nectar amount (fitness value) of the new source (new solution). Provided that the nectar amount of the new one is higher than that of the previous one, the bee memorizes the new position and forgets the old one. Otherwise she keeps the position of the previous one in her memory. After all employed bees complete the search process, they share the nectar information of the food sources and their position information with the onlooker bees. An onlooker bee evaluates the nectar information taken from all employed bees and chooses a food source with a probability related to its nectar amount. As in the case of the employed bee, she produces a modification on the position in her memory and checks the nectar amount of the candidate source. Providing that its nectar is higher than that of the previous one, the bee memorizes the new position and forgets the old one. These steps are repeated through a predetermined number of cycles called Maximum Number of Cycle (MCN) or until a termination criteria is satisfied. It is clear from the explanation that there are four control parameters in the ABC: The number of food sources which is equal to the number of employed or onlooker bees (SN), the modification rate (MR), the value of *limit* and the maximum cycle number (MCN).

3 Other Algorithms Considered in the Experiments

3.1 Particle Swarm Optimization Algorithm

In Particle Swarm Optimization algorithm [5], a population of particles starts to move in search space by following the current optimum particles and changes the positions in order to find out the optima. Main steps of the procedure are:

1: Initialize Population
2: **repeat**
3: Calculate fitness values of particles
4: Choose the best particle
5: Calculate the velocities of particles by (5)

$$\boldsymbol{v}(t+1) = \begin{array}{l} \chi(\omega \boldsymbol{v}(t)+ \\ \phi_1 \text{rand}(0,1)(\boldsymbol{p}(t) - \boldsymbol{x}(t))+ \\ \phi_2 \text{rand}(0,1)(\boldsymbol{g}(t) - \boldsymbol{x}(t))) \end{array} \tag{5}$$

6: Update the particle positions by (6)

$$\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \boldsymbol{v}(t+1) \tag{6}$$

The velocity update is performed as indicated in Equation 6:
7: **until** requirements are met

The parameter ω is called the inertia weight and controls the magnitude of the old velocity $\boldsymbol{v}(t)$ in the calculation of the new velocity, whereas ϕ_1 and ϕ_2 determine the significance of $\boldsymbol{p}(t)$ and $\boldsymbol{g}(t)$, respectively. χ parameter is constriction

factor that insures the convergence of particle swarm optimization algorithm. Furthermore, υ_i at any time step of the algorithm is constrained by the parameter υ_{max}. The swarm in PSO is initialized by assigning each particle to a uniformly and randomly chosen position in the search space. Velocities are initialized randomly in the range $[\upsilon_{min},\upsilon_{max}]$.

3.2 Quantum-Behaved Particle Swarm Optimization (QPSO)

In Quantum-Behaved Particle Swarm Optimization (QPSO), proposed by Sun et al. [8], quantum theory, following the principle of state superposition and uncertainty, was integrated with PSO. In QPSO, next position is calculated as:

$$x(t+1) = p \pm \beta * |mbest - x(t)| * \ln(1/u) \tag{7}$$

mbest, center-of-gravity position of all the particles, is defined by (8):

$$\begin{aligned} mbest &= \sum_{i=1}^{M} p_i/M \\ mbest &= \left(\sum_{i=1}^{M} p_{i1}/M, \sum_{i=1}^{M} p_{i2}/M, \ldots, \sum_{i=1}^{M} p_{id}/M \right) \end{aligned} \tag{8}$$

where M is the population size and p_i is the pbest of particle i.

Parameter β is called Creativity Coefficient that affects the individual particles convergence speed and performance of the algorithm.

3.3 Branch and Bound Technique

The branch and bound method is based on tree structures of integer solutions. Keeping the tree as much as possible is the main purpose of the technique. Most promising nodes are mostly allowed to grow and these nodes are branched. When sub-nodes are determined to be on lower and upper bounds, these nodes are bounded.

4 Experimental Results and Discussion

In the experiments, nine problems widely used in the literature are employed in order to investigate the performance of the ABC algorithm on integer programming problems. These problems are listed below:

Test Problem 1 [9]:
$F_1(x) = \|x\|_1 = |x_1| + \ldots + |x_D|$.
with $x = (x_1, \ldots, x_D) \in [-100, 100]^D$ where D is the dimension. The solution is $x_i^* = 0$, $i = 1, \ldots, D$ and $F_1(x^*) = 0$.

Test Problem 2 [9]:

$$F_2(x) = x^\top x = (x_1 \quad \ldots \quad x_D) \begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix}.$$

with $x = (x_1, \ldots, x_D)^\top \in [-100, 100]^D$ where D is the dimension. The solution is $x_i^* = 0$, $i = 1, \ldots, D$ and $F_2(x^*) = 0$.

Test Problem 3 [10]:
$F_3(x) = -(\,15\ 27\ 36\ 18\ 12\,)x +$
$$x^\top \begin{pmatrix} 35 & -20 & -10 & 32 & -10 \\ -20 & 40 & -6 & -31 & 32 \\ -10 & -6 & 11 & -6 & -10 \\ 32 & -31 & -6 & 38 & -10 \\ -10 & 32 & -10 & -20 & 31 \end{pmatrix} x.$$
The best known solutions are $x^* = (0, 11, 22, 16, 6)^\top$ and $x^* = (0, 12, 23, 17, 6)^\top$ and $F_3(x) = -737$.

Test Problem 4 [10]:
$F_4(x) = (9x_1^2 + 2x_2^2 - 11)^2 + (3x_1 + 4x_2^2 - 7)^2$. The solution is $x^* = (1, 1)^\top$ and $F_4(x^*) = 0$.

Test Problem 5 [10]:
$F_5(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$. The solution is $x^* = (0, 0, 0, 0)^\top$ and $F_5(x^*) = 0$.

Test Problem 6 [11]:
$F_6(x) = 2x_1^2 + 3x_2^2 + 4x_1x_2 - 6x_1 - 3x_2$. The solution is $x^* = (2, -1)^\top$ and $F_6(x^*) = -6$.

Test Problem 7 [10]:
$F_7(x) = -3803.84 - 138.08x_1 - 232.92x_2 + 123.08x_1^2 + 203.64x_2^2 + 182.25x_1x_2$. The solution is $x^* = (0, 1)^\top$ and $F_7(x^*) = -3833.12$.

Test Problem 8 [10]:
$F_8(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$. The solution is $x^* = (3, 2)^\top$ and $F_8(x^*) = 0$.

Test Problem 9 [10]:
$F_9(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$. The solution is $x^* = (1, 1)^\top$ and $F_9(x^*) = 0$.

In order to cope with integer programming problems, Particle Swarm Optimization and Artificial Bee Colony algorithms truncate the parameter values to the closest integer after producing new solutions.

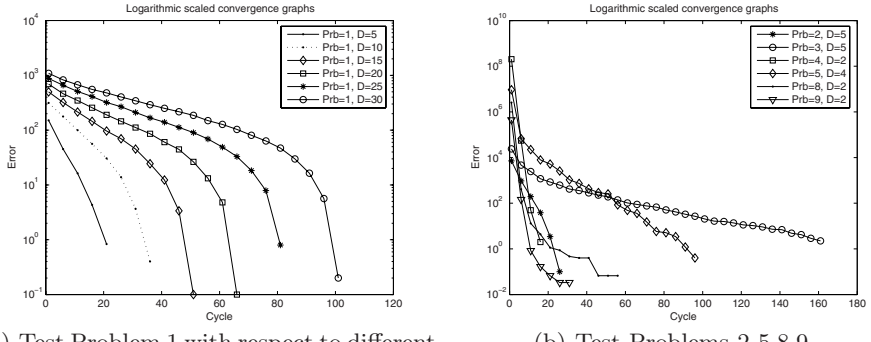
In the experiments, the results of the ABC algorithm were compared to those of Quantum-Behaved Particle Swarm Optimization (QPSO) algorithm taken from [12] and those of Particle Swarm Optimization variants and Branch and Bound algorithm taken from [1]. In Table 1, the comparison of ABC and QPSO is presented. Results in Table 1 are success rates of 50 independent runs with different random seeds and mean number of iterations required for obtaining desired accuracy. In [12], desired accuracy is not given, we assumed that to be 10^{-6} which is precise enough. PSO-In indicates a variant of PSO algorithm with inertia weight and without constriction factor. Inertia weight, ω, is gradually decreased from 1.0 to 0.4. β parameter of QPSO algorithm is changed from 1.2 to

Table 1. Success Rates, Mean Iteration numbers, Swarm Size and Maximum number of Iterations of PSO-In, QPSO [12] and ABC algorithms for of test problems $F_1 - F_5$, F_8, F_9. SS:Swarm Size, MNI:Maximum Number of Iterations, D:Dimension of the problem, SR:Success Rate, MI: Mean of iteration numbers, Bold face indicates the best result.

				PSO-In		QPSO		ABC	
SS	MNI	F	D	SR	MI	SR	MI	SR	MI
20	1000	F_1	5	100	72.8	100	27.2	100	**20.03**
50	1000	F_1	10	100	90.26	100	48.26	100	**34.63**
100	1000	F_1	15	100	94.36	100	64.46	100	**48.93**
200	1000	F_1	20	100	96.3	100	72.24	100	**64.43**
250	1500	F_1	25	100	99.44	100	83.86	100	**81.33**
300	2000	F_1	30	100	103.14	100	**92.56**	100	99.76
20	1000	F_2	5	100	77.82	100	**21.2**	100	21.93
150	1000	F_3	5	100	**125.03**	100	166.9	100	165.43
20	1000	F_4	2	100	32.9	100	**8.95**	100	11.5
40	1000	F_5	4	100	79.6	100	65.7	100	**60.1**
20	1000	F_8	2	100	81.24	100	**19.35**	100	19.4
50	1000	F_9	2	100	41.4	100	14.9	100	**11.93**

Table 2. Dimension of the functions and Swarm Size used in the experiments for each function. D: Dimension of the problem, SS: Swarm Size.

F	F1	F1	F1	F1	F1	F1	F2	F3	F4	F5	F6	F7
D	5	10	15	20	25	30	5	5	2	4	2	2
SS	20	20	50	50	100	100	10	70	20	20	10	20



(a) Test Problem 1 with respect to different dimensions

(b) Test Problems 2-5,8,9

Fig. 1. Logarithmic scaled convergence graphs of the ABC algorithm

0.4 [12]. While obtaining the results in Table 1, the values of control parameters of ABC algorithm were $SN * D * 5$ and 0.8 for $limit$ and MR, respectively. Maximum number of iterations and swarm sizes for each problem are also reported on Table 1. From the results in Table 1, for all functions, PSO, QPSO and ABC algorithms were able to find the solutions with desired accuracy in allowed maximum number of function evaluations. In terms of mean number of iterations to reach the desired accuracy, algorithms produce similar results for lower dimensional functions ($F_2, F_3, F_4, F_5, F_8, F_9$), while ABC algorithm is much better than

Table 3. Success Rates, Mean, Standard Deviation and Medians of evaluation numbers of PSO variants, BB Technique [1] and ABC Algorithm for Test Problems F_1-F_7, SR:Success Rate, StD: Standard Deviation, Bold face indicates the best result

Function	Method	SR	Mean	StD	Median	Function	Method	SR	Mean	StD	Median
F_1, $D = 5$	PSO-In	30/30	1646.0	661.5	1420	F_2 D=5	PSO-In	30/30	1655.6	618.4	1650
	PSO-Co	30/30	744.0	89.8	730		PSO-Co	30/30	428.0	57.9	430
	PSO-Bo	30/30	692.6	97.2	680		PSO-Bo	30/30	418.3	83.9	395
	BB	30/30	1167.83	659.8	1166		BB	30/30	**139.7**	102.6	93
	ABC	30/30	**376**	64.6	380		ABC	30/30	449.3	56.7	440
F_1, $D = 10$	PSO-In	30/30	4652.0	483.2	4610	F_3 D=5	PSO-In	30/30	4111.3	1186.7	3850
	PSO-Co	30/30	1362.6	254.7	1360		PSO-Co	30/30	**2972.6**	536.4	2940
	PSO-Bo	30/30	1208.6	162.7	1230		PSO-Bo	30/30	3171.0	493.6	3080
	BB	30/30	5495.8	1676.3	5154		BB	30/30	4185.5	32.8	4191
	ABC	30/30	**727.3**	64.4	740		ABC	24/30	13850	6711.3	11550
F_1, $D = 15$	PSO-In	30/30	7916.6	624.1	7950	F_4 D=2	PSO-In	30/30	304.0	101.6	320
	PSO-Co	30/30	3538.3	526.6	3500		PSO-Co	30/30	297.3	50.8	290
	PSO-Bo	30/30	2860.0	220.2	2850		PSO-Bo	30/30	302.0	80.5	320
	BB	30/30	10177.1	2393.4	10011		BB	30/30	316.9	125.4	386
	ABC	30/30	**974**	60.5	960		ABC	30/30	**240.7**	79.4	240
F_1, $D = 20$	PSO-In	30/30	8991.6	673.3	9050	F_5 D=4	PSO-In	30/30	1728.6	518.9	1760
	PSO-Co	30/30	4871.6	743.3	4700		PSO-Co	30/30	1100.6	229.2	1090
	PSO-Bo	29/30	4408.3	3919.4	3650		PSO-Bo	30/30	1082.0	295.6	1090
	BB	30/30	16291.3	3797.9	14550		BB	30/30	2754.0	1030.1	2714
	ABC	30/30	**1275.3**	97.7	1260		ABC	30/30	**193.3**	53.5	180
F_1, $D = 25$	PSO-In	30/30	11886.6	543.7	11900	F_6 D=2	PSO-In	30/30	**178.0**	41.9	180
	PSO-Co	30/30	9686.6	960.1	9450		PSO-Co	30/30	198.6	59.2	195
	PSO-Bo	25/30	9553.3	7098.6	6500		PSO-Bo	30/30	191.0	65.9	190
	BB	20/30	23689.7	2574.2	25043		BB	30/30	211.1	15.0	209
	ABC	30/30	**1554.7**	108.6	1540		ABC	30/30	258.7	113.6	240
F_1, $D = 30$	PSO-In	30/30	13186.6	667.8	13050	F_7 D=2	PSO-In	30/30	334.6	95.5	340
	PSO-Co	30/30	12586.6	1734.9	12500		PSO-Co	30/30	324.0	78.5	320
	PSO-Bo	19/30	13660.0	8863.9	7500		PSO-Bo	30/30	306.6	96.7	300
	BB	14/30	25908.6	755.5	26078		BB	30/30	358.6	14.7	355
	ABC	30/30	**1906**	129.9	1900		ABC	30/30	**106.7**	44.8	100

other two algorithms for higher dimensions of F1 function. From these results, it can be said that ABC algorithm shows a similar or better performance compared to QPSO and much better performance respect to PSO-In. Convergence graphs of the ABC algorithm are presented on Figure 1(a) and 1(b). Figure 1(a) shows the convergence rate of the ABC algorithm for Test Problem 1 with respect to different dimensions. For all cases, ABC algorithm converges the optima around 100 cycles. Figure 1(b) demonstrates the convergence of the ABC algorithm for Test Problems 2, 3, 4, 5, 8, 9.

In the second part of the experiments, ABC algorithm was compared to Branch and Bound (BB) technique and PSO variants which are PSO-In, with inertia weight and without constriction factor, PSO-Co, with constriction factor and without inertia weight, and PSO-Bo, with both constriction factor and inertia weight. For these comparisons, dimension and swarm sizes for all test problems are listed on Table 2, maximum number of evaluations are set to 25000 and algorithms are terminated when the accuracy is reached 10^{-6}. In these comparisons, algorithms were repeated through 30 independent runs with the control parameter values as follows: the constriction factor χ was set equal to 0.729; the inertia weight ω was gradually decreased from 1 towards 0.1; $\phi_1 = \phi_2 = 2$; and $\upsilon_{max} = 4$ [1]. Parameters of the ABC algorithm were set to the values given in previous experiments. Success Rates, Mean, Standard Deviation and Medians of evaluation numbers of PSO variants, BB Technique and ABC Algorithm for Test Problems F_1-F_7 are given in Table 3. Results in Table 3 indicate that ABC algorithm shows superior performance respect to other algorithms on all dimensions of the function F_1 (on all of six cases) in terms of mean, standard deviation and median values. It is also clear that BB technique produces the

best result on one function, F_2. PSO-Co is the winner on one function, F_3, while ABC algorithm produces the worst result in terms of success rate in allowed maximum number of function evaluations on this function. On three functions, F_4, F_5 and F_7, ABC has the best performance. For F_6 function, PSO-In shows superior performance.

In overall evaluation, ABC algorithm can produce comparable or better performance respect to PSO variants and BB technique for integer programming problems considered in this study. Exact techniques such as BB produce solutions in computationally expensive period. ABC algorithm produces high quality solutions more quickly and is not subjected to linear constraints on the objective function that need not be linear. There is no limitation on the applicability of ABC algorithm. Control parameters certainly have effect on the performance of the algorithms. Lower values of MR has negative effect on the convergence rate of the algorithm. From the experiences gained from the previous studies, 0.8 is an appropriate value for MR. Lower values of *limit* increases the exploration capability while decreasing the exploitation capability. Value of the control parameter limit should balance the exploration and exploitation processes.

5 Conclusion

Since most of combinatory problems can be converted to Integer Programming problems, integer programming problems are important in the subject of optimization field. In this paper, performance of the Artificial Bee Colony Algorithm was investigated on integer problems and compared to the results of Particle Swarm Optimization algorithm variants, and Branch and Bound method. From the results, it can be concluded that Artificial Bee Colony algorithm can cope with integer problems efficiently in a robust manner. This work is supported by Erciyes University, the Department of Research Projects under contract FBA–06–22.

References

[1] Laskari, E.C., Parsopoulos, K.E., Vrahatis, M.N.: Particle swarm optimization for integer programming. In: CEC 2002: Proceedings of the Evolutionary Computation on 2002. CEC 2002. Proceedings of the 2002 Congress, pp. 1582–1587. IEEE Computer Society, Los Alamitos (2002)

[2] Nemhauser, G.L., Wolsey, L.A.: Integer Progamming. In: Handbooks in Operations Research and Management Science, vol. 1. Elsevier Science and Technology, Amsterdam (1989)

[3] Misra, K.B., Sharma, U.: An efficient algorithm to solve integer-programming problems arising in system-reliability design. IEEE Transactions On Reliability 40(1), 81–91 (1991)

[4] Rouillon, S., Desaulniers, G., Soumis, F.: An extended branch-and-bound method for locomotive assignment. Transportation Research Part B: Methodological 40(5), 404–423 (2006)

[5] Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: IEEE International Conference on Neural Networks 1995, vol. 4, pp. 1942–1948 (1995)
[6] Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department (2005)
[7] Tereshko, V.: Reaction–diffusion model of a honeybee colony's foraging behaviour. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) PPSN 2000. LNCS, vol. 1917. Springer, Heidelberg (2000)
[8] Sun, J., Feng, B., Xu, W.: Particle swarm optimization with particles having quantum behavior. In: Evolutionary Computation, CEC 2004, vol. 1, June 2004, pp. 325–331 (2004)
[9] Rudolph, G.: An evolutionary algorithm for integer programming. In: PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature, pp. 139–148. Springer, Heidelberg (1994)
[10] Glankwahmdee, A., Liebman, J.S., Hogg, G.L.: Unconstrained discrete nonlinear programming. Engineering Optimization 4, 95–107 (1979)
[11] Rao, S.S.: Engineering Optimization- Theory and Practice. Wiley Eastern, New Delhi (1996)
[12] Liu, J., Sun, J., Xu, W.: Quantum-Behaved Particle Swarm Optimization for Integer Programming. In: King, I., Wang, J., Chan, L.-W., Wang, D. (eds.) ICONIP 2006. LNCS, vol. 4233, pp. 1042–1050. Springer, Heidelberg (2006)