A Modified Hybrid Particle Swarm Optimization Algorithm for Multidimensional Knapsack Problem

Said Labed
Computer Science Department,
MISC Laboratory,
Mentouri University,
Constantine-Algeria

Amira Gherboudj
Computer Science Department,
MISC Laboratory,
Mentouri University,
Constantine-Algeria

Salim Chikhi
Computer Science Department,
MISC Laboratory,
Mentouri University,
Constantine-Algeria

ABSTRACT
In this paper, a modified hybrid Particle Swarm Optimization (MHPSO) algorithm that combines some principles of Particle Swarm Optimization (PSO) and Crossover operation of the Genetic Algorithm (GA) is presented. Our contribution has a twofold aim: first, is to propose a new hybrid PSO algorithm. Second is to prove the effectiveness of the proposed algorithm in dealing with NP-hard and combinatorial optimization problems. In order to test and validate our algorithm, we have used it for solving the Multidimensional Knapsack Problem (MKP) which is a NP-hard combinatorial optimization problem. The experimental results based on some benchmarks from OR-Library, show a good and promise solution quality obtained by the proposed algorithm.

Keywords
Particle Swarm Optimization, Crossover Operation, Continuous/Discrete Optimization Problems, Multidimensional Knapsack Problem.

1. INTRODUCTION
Optimization problems can be classed into two main classes: continuous optimization problems and discrete optimization problems. In continuous optimization problems, the solution is represented by a set of real numbers. However, in discrete optimization problems, the solution is represented by a set of integer numbers. Furthermore, discrete binary optimization problems are a sub class of the discrete optimization problems class in which a solution is represented by a set of bits.

Several metaheuristics such as genetic algorithm (GA), simulated annealing (SA) and tabu search (TS)… have been proposed in the literature and utilized to solve various optimization problems (continuous or discrete problems). However, most of them are very greedy in terms of computation time required to reach the optimal solution.

Particle Swarm Optimization (PSO) is an evolutionary metaheuristic that mimics the collective behavior of animals living in groups such as bird flocking and fish schooling. The original PSO version operates in continuous search spaces and it has proved its simplicity of implementation, its effectiveness and its very fast convergence [1]. However, the selection and adaptation of the large number of PSO parameters such as: swarm size, inertia coefficient, acceleration coefficients ... play a crucial role for good and efficient operation of PSO. On the other hand, PSO may be easily trapped into local optima if the global best and local best positions are equal to the position of particle over a number of iterations [2].

The main purpose of this paper is to propose a modified hybrid Particle Swarm Optimization algorithm that we have called MHPSO, in which we combine some principles of Particle Swarm Optimization and the Crossover operation of the Genetic Algorithm. The aim of this work is twofold: first, is to benefit from PSO advantages (simplicity, efficiency and rapidity) and propose a new hybrid algorithm which can be used to solve different optimization problems in continuous or discrete areas. Second, is to prove that the proposed algorithm (MHPSO) is effective in dealing with NP-hard combinatorial optimization problems. The feature of our approach is that it requires few parameters to be set compared with PSO or GA. On the other hand, this hybridization has allowed a good balance between exploration and exploitation of the search space.

To validate and prove the performance and the effectiveness of our algorithm, we have tested it on some multidimensional knapsack problem instances. Moreover, we have used the particle repair algorithm that we have proposed and a check and repair operator [3] to transform infeasible solutions to feasible solutions with small and big size instances respectively. Experimental results show the effectiveness of the proposed algorithm and its ability to achieve good quality solutions.

The remainder of this paper is organized as follows. Section 2 presents PSO principle. An overview of the Binary Particle Swarm Optimization (BPSO) is presented in section 3. Section 4, presents the Multidimensional Knapsack Problem principle and formulation. In section 5, the proposed algorithm is described. Experimental results are discussed in section 6, and a conclusion is provided in the seven section of this paper.

2. PSO PRINCIPLE
Particle Swarm Optimization (PSO) is an evolutionary metaheuristic. It was created in 1995 by Kennedy and Eberhart [4] for solving optimization problems. It mimics the collective behavior of animals living in groups such as bird flocking and fish schooling. The PSO method involves a set of agents for solving a given problem. This set is called swarm, each swarm is composed of a set of members, they are called particles. Each particle is characterized by position $x_{id}= (x_{i1}, x_{i2},…, x_{id},…, x_{iD})$ and velocity $v_{id}= (v_{i1}, v_{i2},…, v_{id},…, v_{iD})$ in a search space of D-dimension. During the search procedure, the particle tends to move towards the best position (solution) found. At each iteration of the search procedure, the particle moves and updates its velocity and its position in the swarm based on experience and the results found by the particle itself, its neighbors and the swarm. It therefore combines three components: its own current velocity, its best position $p_{bestid}= (p_{besti1}, p_{besti2},…, p_{bestid},…,$

p_{bestiD}) and the best position obtained by its informants. Thus the equations for updating the velocity and position of particles are presented below [5]:

$$v_{id}(t)= \omega\, v_{id}(t-1) + c_1\, r_1\, (p_{bestid}(t-1) - x_{id}(t-1)) + c_2\, r_2\, (g_{bestd}(t-1) - x_{id}(t-1)) \quad (1)$$

$$x_{id}(t)= x_{id}(t-1) + v_{id}(t) \quad (2)$$

ω is an inertia coefficient. (x_{id} (t), x_{id} (t-1)), (v_{id} (t), v_{id} (t-1)): Position and Velocity of particle i in dimension d at times t and t-1, respectively. p_{bestid} (t-1), g_{bestd}(t-1) : the best position obtained by the particle i and the best position obtained by the swarm in dimension d at time t-1, respectively. c_1, c_2: two constants representing the acceleration coefficients. r_1, r_2: random numbers drawn from the interval [0,1[. v_{id} (t-1), c_1 r_1 (p_{bestid} (t-1) - x_{id} (t-1)), c_2 r_2 (g_{bestd}(t-1) - x_{id} (t-1)): the three components mentioned above, respectively.

The PSO algorithm begins by initializing the size of the swarm and the various parameters. Assign randomly to each particle an initial position and velocity. Initialize p_{bestid}, then calculate the fitness f(x_{id}) of particles in order to calculate the best position found by the swarm (g_{bestd}). At each iteration, particles are moved using equations (1) and (2). Their objective functions are calculated and p_{bestid}, g_{bestd} are updated. The process is repeated until the satisfaction of stopping criterion. A pseudo PSO algorithm is presented in our previous work in [1].

3. BINARY PARTICLE SWARM OPTIMIZATION (BPSO) ALGORITHM

The first version of the Binary Particle Swarm Optimization (BPSO) algorithm (The Standard BPSO algorithm) was proposed in 1997 by Kennedy and Eberhart [6]. In the BPSO algorithm, the position of particle i is represented by a set of bit. The velocity v_{id} of the particle i is calculated from equation (1). v_{id} is a set of real numbers that must be transformed into a set of probabilities, using the sigmoid function as follows:

$$sig(v_{id}) = \frac{1}{1+\exp(-v_{id})} \quad (3)$$

Where sig (v_{id}) represents the probability of bit x_{id} takes the value 1.

To avoid the problem of the divergence of the swarm, the velocity v_{id} is generally limited by a maximum value V_{max} and a minimum value $-V_{max}$, i.e. $v_{id} \in [-V_{max}, V_{max}]$. The position x_{id} of particle i is updated as follows:

$$x_{id}= \begin{cases} 1 \text{ if } r < \text{Sig}(v_{id}) \\ 0 \quad \text{Otherwise} \end{cases} \quad (4)$$

Where r is a random number taken from the interval [0, 1[.

Two main parameter problems with BPSO are discussed in [7]. First, the effect of velocity clamping in the continuous PSO is opposite of that in the binary PSO (BPSO). In fact, in the continuous PSO the maximum velocity of the particle encourage the exploration, but it limits the exploration in the binary PSO [7]. The second problem is the difficulties with choosing proper values for inertia weight. In fact, w < 1 prevents convergence [7].

4. MULTIDIMENSIONAL KNAPSACK PROBLEM

The knapsack problem (KP) is one of the easier NP-hard problems [8]. It can be defined as follows: Assuming that we have a knapsack with maximum capacity C and a set of n objects. Each object i has a profit p_i and a weight w_i. The problem consists to select a subset of objects that maximize the knapsack profit without exceeding the maximum capacity of the knapsack. KP can be formulated as:

$$\text{Maximize} \sum_{i=1}^{n} p_i x_i \quad (5)$$

$$\text{Subject} \sum_{i=1}^{n} w_i x_i \le C \quad (6)$$
$$x_i \in \{0,1\}$$

Many variants of the knapsack problem were proposed in the literature including the Multidimensional Knapsack Problem (MKP). MKP is an important issue in the class of knapsack problem. It is a combinatorial optimization problem [9] and it is also a NP-hard problem [9, 10]. In the MKP, each item x_i has a profit p_i like in the simple knapsack problem. However, instead of having a single knapsack to fill, we have a number m of knapsack of capacity C_j (j = 1 ... m). Each x_i has a weight w_{ij} that depends of the knapsack j (example: an object can have a weight 3 in knapsack 1, 5 in knapsack 2, etc.). A selected object must be in all knapsacks. The objective in MKP is to find a subset of objects that maximize the total profit without exceeding the capacity of all dimensions of the knapsack. MKP can be stated as follows:

$$\text{Maximize} \sum_{i=1}^{n} p_i x_i \quad (7)$$

$$\text{Subject} \sum_{i=1}^{n} w_{ij} x_i \le C_j \quad j = 1...m \quad (8)$$
$$x_i \in \{0,1\}$$

The MKP can be used to formulate many industrial problems such as capital budgeting problem, allocating processors and databases in a distributed computer system, cutting stock, project selection and cargo loading problems [10]. Due to its importance and its NP-Hardness, MKP has received the attention of many researches. It was treated by several methods for examples, Chu and Beasley [10] proposed a genetic algorithm for the MKP, Alonso et al [11] suggested an evolutionary strategy for MKP based on genetic computation of surrogate multipliers, Drexl [12] proposed a simulated annealing approach for the MKP, Stefka Fidanova [13] applied the ant colony optimization to solve the MKP, Li et al [14] suggested a genetic algorithm based on the orthogonal design for MKP, Zhou et al [9] suggested a chaotic neural network combined heuristic strategy for MKP, Angelelli et al [15] proposed Kernel search: A general heuristic for MKP, Kong and Tian [16] proposed a particle swarm optimization to solve the MKP and so one.

5. THE PROPOSED ALGORITHM (MHPSO)

In this section, we present the proposed algorithm called MHPSO which combines some principles of Particle Swarm Optimization and Crossover operation of the Genetic Algorithm in the aim to get benefit from the good exploration of the search space offered by the crossover operation and the good exploitation of the PSO algorithm and its fast convergence. The proposed algorithm is explained in more detail in the follow:

5.1 Representation

Since the MKP is a 0-1 optimization problem and it requires a binary solution, it is an obvious choice to represent and initialize the population with a binary representation. In this aim, we have utilized binary vectors of size D to represent different particles. The representation of particle i is as follows:

$$x_{id} = [x_{i1}, x_{i2}, \ldots, x_{id}, \ldots, x_{iD}]$$

Where $x_{id} = \begin{cases} 1 & \text{If the object is selected} \\ 0 & \text{Otherwise} \end{cases}$

5.2 Particle Repair Algorithm (PRA)

In the MKP, the solution must verify the m constrained of the knapsack to be accepted as a feasible solution. If a solution x exceed the capacity of any dimension of the knapsack, it is considered as infeasible solution and it is not accepted. To repair a solution x, we have proposed Particle Repair algorithm (PRA) which allows conversion of an infeasible solution to feasible solution. A pseudo Particle Repair algorithm is presented below.

Particle Repair Algorithm (PRA)

Input : solution vector x
Output : repaired solution vector x

Calculate $R_j = \sum_{i=1}^{n} w_{ji}\ x_i$, j=1,…, m;
For (j=1,…, m) {
  While $R_j > C_j$ {
    Select randomly $i \in \{1, \ldots, n\}$
    If $x_i = 1$
    { $x_i = 0$;
      Calculate $R_j = \sum_{i=1}^{n} w_{ji}\ x_i$, j=1,…, m
    }
  }
}

5.3 Check and Repair Operator (CRO)

Although PRA guarantees feasible and good quality solution with small size instances, it does not work well with big size instances. Consequently, we have used a Check and Repair Operator (CRO) [3] as alternative to the PRA in order to deal with the big size instances. As described in [3 and 14], CRO is based on two phases that are defined in Algorithm 1 and Algorithm 2 respectively. These two phases used the profit density of every item in every knapsack which is calculated as follows:

$$\delta_{ij} = C_j . p_i / w_{ij} \quad (9)$$

In the first phase the infeasible solution is transformed into feasible solution by the Algorithm 1 [14]. Then the obtained feasible solution is improved in the second phase by the Algorithm 2 [14].

Algorithm 1

1. Calculate the profit density $\delta_{ij} = C_j . p_i / w_{ij}$ for every item in every knapsack.
2. Compute the lowest value of the profit density $\delta_i = \min\{C_j . p_j / w_{ij}\}$ for every item.
3. Sort and relabel items according to the ascending order of δ_i.
4. Remove the corresponding item with lowest values of δ_i from the item set. (i.e. change corresponding gene 1 into gene 0).
5. Repeat Step 4 until a feasible solution is achieved.

Algorithm 2

1. Calculate the profit density $\delta_{ij} = C_j . p_i / w_{ij}$ of every item out of the knapsack.
2. Compute the lowest value of the profit density $\delta_i = \min\{C_j . p_j / w_{ij}\}$ for every item.
3. Sort and relabel items according to the descending order of δ_i.
4. Add the corresponding item with highest values of δ_i into the item set (i.e. change corresponding gene 0 into gene 1).
5. If one of knapsack constraints is not satisfied, then stop, and output the resulting chromosome. Otherwise, return to Step 4.

5.4 Crossover Operation

Crossover operation is one of the Genetic Algorithm (GA) operations which has introduced by John Holland in 1960 [17]. The main role of the crossover operation is to produce a new population (individual). It consists in combining the characteristics of two individuals (parents) to produce one or two new individuals (childs). In the proposed algorithm and in the aim to produce a new population, we have used the crossover operation between the best position p_{bestid} of particle i and its current position x_{id} and between the best position obtained by the swarm g_{bestd} and the current position x_{id} of the particle i. In all cases, if we assume that we have 2 particles x_1 and x_2 and we want to cross them, we begin by initializing the step p, and 2 random positions c_1 and c_2, then we follow steps presented in Algorithm 2. Where Algorithm 2 represents a pseudo Crossover Algorithm. The proposed Crossover Algorithm, gives birth to two new child. To choose which one will represent the new particle, we calculate the fitness $f(x_i)$ of each child and we select the best one.

Crossover Algorithm

Input : Tow particles x_1 and x_2
Output : One particle x_i

1. Choose a step p
2. Choose two random positions c_1 and c_2 from x_1 and x_2 : $c_1, c_2 \in \{1, \ldots, D\text{-}p\}$
3. Swap elements of x_1 from c_1 to c_1+p with those of x_2 from c_2 to c_2+p
4. Swap elements of x_1 from c_2 to c_2+p with those of x_2 from c_1 to c_1+p
5. Calculate the fitness of new particles and select the best one.

5.5 Outlines of the proposed algorithm

Now, we explain how the proposed algorithm MHPSO can found a solution of an optimization problem (continuous or discrete problem). As any algorithm, the first step in the MHPSO algorithm is to initialize some necessary parameters for good and efficient operation of the algorithm. The main characteristic of the MHPSO algorithm is its simplicity. In fact, comparing with other population metaheuristic such as PSO and GA, there are few parameters to be set. Steps of the MHPSO algorithm are presented below.

Step 1: Initialize a swarm size S and random position of each particle. For each particle, let $p_{bestid} = x_{id}$

Step 2: Apply **PRA** on each infeasible solution and evaluate the fitness of particles
Step 3: Calculate the g_{bestd}
Step 4: Calculate the new x_{id} of each particle using the following equation:

$$x_{id} = \text{Max}\ [(p_{bestid} \otimes x_{id}), (g_{bestd} \otimes x_{id})] \quad (10)$$

Where the «⊗» operator is the crossover operation of the Genetic Algorithm. A pseudo code of the proposed crossover operation is presented in Crossover Algorithm

Step 5: Apply **PRA** or **CRO** on each infeasible solution and evaluate the fitness of particles
Step 6: update p_{bestid} and g_{bestd} as follows:

If $(f(x_{id}) > f(p_{bestid}))$ $p_{bestid} = x_{id}$;
If $(f(p_{bestid}) > f(g_{bestd}))$ $g_{bestd} = p_{bestid}$;

Step 7: Stop iterations if stop condition is verified. Return to Step 4, otherwise.

The solution of the problem is the last g_{bestd}. Fig 1 shows the flowchart of the proposed algorithm.

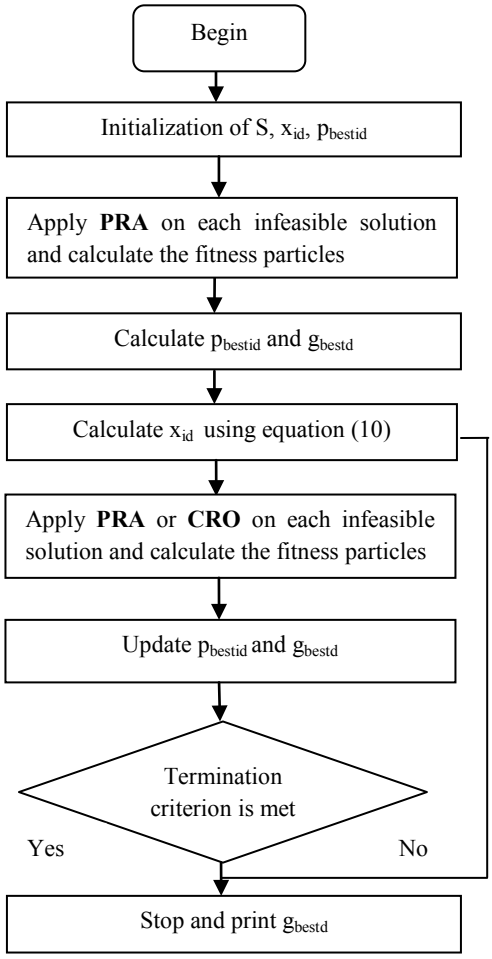

Fig1: Flowchart of the proposed algorithm (MHPSO)

6. EXPERIMENTAL RESULTS

The proposed MHPSO algorithm was implemented in Matlab 7. To assess the efficiency and performance of our MHPSO algorithm, we have tested it on some instances from OR-Library [18]. Two parts of experiments were performed. In the first part of experiments, we have tested and compared our algorithm on some small MKP instances. In this part of experiments, we have first tested the MHPSO algorithm on some MKP instances from the literature, the used instances are named: HP, SENTO, WEING and WEISH instances. On the other hand, we have compared the MHPSO algorithm with the best known solution (the exact solution) and the obtained solution by the standard PSO with penalty function technique (denoted as PSO-P) [16] [6] on MKP instances taken from 7 benchmarks named mknap1. In the second part of experiments, we have tested the MHPSO algorithm on some big size MKP instances taken from benchmarks named mknapcb1 and mknapcb4. We have used 5 tests of the benchmarks mknapcb1 (5.100) which have 5 constraints and 100 items, and we have used 5 tests of the benchmarks mknapcb4 (10.100) which have 10 constraints and 100 items.

Table1 shows the experimental results of our MHPSO algorithm with some instances taken from the literature. The first column indicates the instance name, the second and third columns indicate the problem size i.e. number of objects and number of knapsack dimensions respectively. The fourth column indicates the best known solution from OR-Library. Column 5 record the best results obtained by the MHPSO. Table1 shows that the proposed algorithm is able to find the best known result of all instances.

Table2 shows experimental results of our MHPSO algorithm and the PSO-P algorithm with 7 benchmarks taken from mknap1 instances. The first column indicates the problem index, the second and third columns indicate the problem size i.e. number of objects and number of knapsack dimensions respectively. The fourth column indicates the best known solution from OR-Library. Column 5 and 6 record the best and average (AVG) results obtained by the MHPSO and PSO-P during 30 independent runs for each instance.

Table2 shows that our algorithm is able to found the best solution of all the mknap1 instances. Compared with PSO-P algorithm which utilized penalty function technique to deal with the constrained problems, the MHPSO algorithm gives better averages (AVG) in most cases. The found results are very encouraging. They prove the efficiency of the proposed algorithm.

Table 1. Experimental results with some instances from the literature.

Instance	n	m	best known	MHPSO
HP1	28	4	3418	3418
HP2	35	4	3186	3186
PB5	20	10	2139	2139
PB6	40	30	776	776
PB7	37	30	1035	1035
SENTO1	60	30	7772	7772
SENTO2	60	30	8722	8722
WEING1	28	2	141278	141278
WEING2	28	2	130883	130883
WEING3	28	2	95677	95677
WEING4	28	2	119337	119337
WEING7	105	2	1095445	1095445
WEISH01	30	5	4554	4554
WEISH06	40	5	5557	5557
WEISH10	50	5	6339	6339
WEISH15	60	5	7486	7486
WEISH18	70	5	9580	9580
WEISH22	80	5	8947	8947

Finally, Table 3 shows the experimental results of our MHPSO with some hard instances of mknapcb1 and mknapcb4. Column 1 shows the benchmark name. Column 2 indicates the problem size and columns 3 and 4 indicate the best known and the MHPSO solutions respectively. Obtained results in table 4 allow saying that MHPSO algorithm gives good results. However, the performances of MHPSO can be increased by the introduction of other specified knapsack heuristic operators utilizing problem-specific knowledge.

Table 2. Results and Comparison of MHPSO with PSO-P for mknap1 instances.

N°	n	M	best known	MHPSO		PSO-P	
				Best	AVG	Best	AVG
1	6	10	3800	3800	3800	3800	3800
2	10	10	8706,1	8706,1	8706,1	8706,1	8570.7
3	15	10	4015	4015	4015	4015	4014.7
4	20	10	6120	6120	6120	6120	6118
5	28	10	12400	12400	12386	12400	12394
6	39	5	10618	10618	10566	10618	10572
7	50	5	16537	16537	16460	16537	16389

Table 3. Experimental Results of MKP with mknapcb1 and mknapcb4 instances

Benchmark Name	Problem size	Best known	MHPSO
mknapcb1	5.100.00	24381	24329
	5.100.01	24274	24149
	5.100.02	23551	23494
	5.100.03	23534	23370
	5.100.04	23991	23889
mknapcb4	10.100.00	23064	22983
	10.100.01	22801	22657
	10.100.02	22131	21853
	10.100.03	22772	22511
	10.100.04	22751	22614

7. CONCLUSION

In this paper, we have proposed a new hybrid particle swarm optimization algorithm that we have called MHPSO. In the MHPSO, we have combined some principle of the particle swarm optimization and the crossover operation of the genetic algorithm. In opposite to the original PSO algorithm that is designed for solving continuous optimization problems, the main feature of the proposed algorithm is that is can be applied to solve all type of optimization problems (continuous, discrete and discrete binary optimization problems) by the appropriate choose of the population type. In the aim to verify and prove the performance of our new algorithm, we have tested it on some MKP benchmarks taken from OR-Library. Experimental results show a good and encouraging solution quality obtained by our proposed algorithm. Based on this promising result, our fundamental perspective is to use the proposed algorithm to solve other NP-hard and combinatorial optimization problems.

8. REFERENCES

[1] A. Gherboudj, S. Chikhi. BPSO Algorithms for Knapsack Problem. A. Özcan, J. Zizka, and D. Nagamalai (Eds.): WiMo/CoNeCo 2011, CCIS 162, pp. 217–227, 2011. Springer (2011)

[2] J. Olamaei, T. Niknam, G. Gharehpetian. Application of particle swarm optimization for distribution feeder reconfiguration considering distributed generators. *Appl. Math. Comput.*, 201(1-2):575-586. (2008)

[3] Carlos Cotta, Jose Ma Troya: A Hybrid Genetic Algorithm for the 0-1 Multiple Knapsack Problem. Artificial Neural Nets and Genetic Algorithms 3, New York (1998) 250-254

[4] J. Kennedy, R.C. Eberhart. Particle Swarm Optimization. In: Proc. IEEE Int. Conf. On Neural Networks, WA, Australia, pp. 1942–1948 (1995)

[5] Shi. Y, R. Eberhart. Parameter Selection in Particle Swarm Optimization. Proceedings of the 7th Annual Conference on Evolutionary Programming, pp. 591-600, LNCS 1447, Springer (1998)

[6] J. Kennedy, R.C. Eberhart. A discrete binary version of the particle swarm algorithm. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, pp. 4104-4109, NJ: Piscatawary (1997)

[7] Khanesar. M-A, Teshnehlab. M and Shoorehdeli. M-A. A Novel Binary Particle Swarm Optimization. In proceedings of the 15th Mediterranean Conference on Control & Automation, July 27 – 29, 2007, Athens – Greece.

[8] Pisinger, D.: Where are the hard knapsack problems? Computers and Operations Research, Vol.32, N°. 9, pp. 2271-2284, 2005.

[9] Y. Zhou, Z. Kuang, J. Wang. A Chaotic Neural Network Combined Heuristic Strategy for Multidimensional Knapsack Problem. In: Proc. L. Kang et al. (Eds.): ISICA 2008, LNCS 5370, pp. 715–722, 2008. Springer (2008)

[10] P.C. Chu, J.E. Beasley. A Genetic Algorithm for the Multidimensional Knapsack Problem. Journal of Heuristics, 4: 63–86 (1998).

[11] C-L. Alonso,F. Caro, J-L. Montana. An Evolutionary Strategy for the Multidimensional 0-1 Knapsack Problem Based on Genetic Computation of Surrogate Multipliers. In: Proc. J. Mira and J.R. Alvarez (Eds.): IWINAC 2005, LNCS 3562, pp. 63–73, 2005.Springer (2005)

[12] Drexl A. A simulated annealing approach to the multiconstraint zero–one knapsack problem. Computing 1988; 40:1–8.

[13] Stefka Fidanova. Ant Colony Optimization for Multiple Knapsack Problem and Model Bias Z. Li et al. (Eds.): NAA 2004, LNCS 3401, pp. 280–287, Springer (2005).

[14] H. Li, Y-C.Jiao, L. Zhang, Z-W. Gu. Genetic Algorithm Based on the Orthogonal Design for Multidimensional Knapsack Problems. In: Proc. L. Jiao et al. (Eds.): ICNC 2006, Part I, LNCS 4221, pp. 696–705, 2006.Springer (2006)

[15] E. Angelelli, R. Mansini, M.G. Speranza. Kernel search: A general heuristic for the multi-dimensional knapsack problem. Computers & Operations Research 37 (2010) 2017–2026. Elsevier (2010).

[16] M. Kong, P. Tian. Apply the Particle Swarm Optimization to the Multidimensional Knapsack Problem. In: Proc. L. Rutkowski et al. (Eds.): ICAISC 2006, LNAI 4029, pp. 1140–1149, 2006. Springer (2006)

[17] J H. Holland. Adaptation in natural and artificial system. Ann Arbor, The university of Michigan Press, (1975).

[18] OR-Library, J.E. Beasley, http: // www. people.brunel.ac.uk/mastjjb/jeb/orlib/mknapinfo.html