Apply the Particle Swarm Optimization to the Multidimensional Knapsack Problem

Min Kong and Peng Tian

Shanghai Jiaotong University, Shanghai 200052, China
kongmin@sjtu.edu.cn, ptian@sjtu.edu.cn

Abstract. This paper proposes a new heuristic approach based on the Particle Swarm Optimization (PSO) for the Multidimensional Knapsack Problem (MKP). Instead of the penalty function technique usually used to deal with the constrained problem, a heuristic repair operator utilizing problem-specific knowledge is incorporated into the modified algorithm. Computational results show that the new PSO based algorithm is capable of quickly obtaining high-quality solutions for problems of various characteristics.

1 Introduction

Particle Swarm Optimization (PSO) is a recently developed meta-heuristic for NP-hard optimization problems. Based on the simulation of both the movement of individual of bird flocks or fish schools and their collective behavior as a swarm, Kennedy and Eberhart[5] introduced the method of PSO in 1995. Applications to various nonlinear optimization problems have shown the success of PSO[7]. To solve constrained problems, PSO usually makes use of penalty function technique in order to reduce the constrained problem to an unconstrained problem by penalizing the objective function despite ill-conditioning[4,9].

This paper deals with the application of PSO in the field of combinatorial optimization (CO) problems, which is a quite rare field tackled by PSO. The constrained problem discussed in this paper is the well-known NP-hard CO problem, the multidimensional knapsack problem (MKP), which can be formulated as:

$$\text{maximize} \quad f = \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} r_{ij} x_j \leq b_i, i = 1, ..., m \tag{2}$$

$$x_j \in \{0, 1\}, j = 1, ..., n \tag{3}$$

Equation (1) describes the objective function for the MKP. Each of the m constraints described in condition (2) is called a knapsack constraint, so the MKP is also called the m-dimensional knapsack problem. Let $I = \{1, 2, ..., m\}$ and $J = \{1, 2, ..., n\}$, with $b_i > 0$ for all $i \in I$ and $r_{ij} \geq 0$ for all $i \in I, j \in J$, a well-stated MKP assumes that $p_j > 0$ and $r_{ij} \leq b_i < \sum_{j=1}^{n} r_{ij}$ for all $i \in I, j \in J$.

L. Rutkowski et al. (Eds.): ICAISC 2006, LNAI 4029, pp. 1140–1149, 2006.

MKP can be regarded as a resource allocation problem of m resources and n objects. Each resource $i \in I$ has a burget b_i, each object $j \in J$ has a profit p_j and consumes r_{ij} of resource i. The problem is to maximize the profit within a limited budget.

MKP is one of the most intensively studied discrete programming problems, mainly because its simple structure which, on the one hand allows exploitation of a number of combinatorial properties and, on the other, more complex optimization problems to be solved through a series of knapsack-type subproblems. Meanwhile, many practical problems can be formulated as a MKP, such as the capital budgeting problem, allocating processors and databases in a distributed computer system, project selection and cargo loading, and cutting stock problems.

This paper utilizes the structure of the binary PSO[6] and combines this method with a problem-specific repair operator instead of the penalty function technique to avoid the violations to problem constraints. Experimental results show that the modified PSO is good at dealing with the specific CO problem.

This paper is organized as follows, the binary PSO algorithm to MKP is briefly introduced in Section 2. In section 3, the modified PSO algorithm applied to MKP is proposed, experimental results are shown in the following Section 4, and a short discussion is presented in Section 5. We end with some conclusions in Section 6.

2 The Binary PSO Model

2.1 Solution Representation and Fitness Function

In the binary PSO model[6], a potential solution to a problem is represented as a particle having binary coordinates $x = \{x_1, \ldots, x_n\}$, $x_j \in \{0, 1\}$ in a n-dimensional space as illustrated in Fig.1.

j	1	2	3	4	5	...	$n-1$	n
x_j	0	1	0	0	1	...	0	1

Fig. 1. Solution Struction of the Binary PSO

For the application to MKP, $x_j = 0$ means that object j is not selected, while $x_j = 1$ means that the object is selected. By this solution representation, we can see that such a solution might not be feasible for MKP. An infeasible solution is one for which at least one of the knapsack constraints is violated, i.e. $\sum_{j=1}^{n} r_{ij} x_j > b_i$ for some $i \in I$.

A penalty function technique is normally incorporated to solve the constrained problem in PSO. For the MKP problem, the fitness function is modified as:

$$f = \sum_{j=1}^{n} p_j x_j - \sum_{i=1}^{m} poslin\left(M_i\left(\sum_{j=1}^{n} r_{ij} x_j - b_i\right)\right) \tag{4}$$

where M_i are some big penalty parameters and *poslin* is a positive linear transform function, which is defined as:

$$poslin(s) = \begin{cases} s & s > 0 \\ 0 & s \leq 0 \end{cases} \tag{5}$$

2.2 The Standard PSO with Penalty Function Technique

The standard PSO with penalty function technique (denoted as PSO-P) does not take care of the feasibility of the solutions generated during the iterations. The knapsack constraints are totally manifested by the penalty function. By automatically moving to the coordinates with bigger objective function value during the iterations, PSO-P is able to find good solutions observing the knapsack constraints.

In PSO-P, a number of particles move stochastically among the binary solution space by flipping various numbers of bits. The position of each particle forms a solution to MKP, which can be represented as a n-dimensional binary string: $x_i = \{x_{i1}, \ldots, x_{in}\}$. The velocity of the movement of each particle is defined as the changes of probabilities that a bit will be in one state or the other, which is represented as $v_i = \{v_{i1}, \ldots, v_{in}\}$, where v_{id} represents the probability for particle i to select 1 at bit d. The velocity of each particle is determined by three kinds of information. One is its velocity value at last iteration, the second is the record of the position of its previous best performance, denoted as $p_i = \{p_{i1}, \ldots, p_{in}\}$, which represents the experience of the particle during the search, the last is the record of the position of the best performance among its topological neighborhood, denoted as $g_i = \{g_{i1}, \ldots, g_{in}\}$, which represents the social experiences of the particles during the search. In conclusion, the velocity for particle i at bit d can be summarized as:

$$v_{id}^{n+1} = v_{id}^{n} + \varphi_1 r_1 (p_{id}^{n} - x_{id}^{n}) + \varphi_2 r_2 (g_{id}^{n} - x_{id}^{n}) \tag{6}$$

where the superscript represents the number of iterations, φ_1 and φ_2 are two positive parameters, r_1 and r_2 are two randomly generated numbers uniformly distributed in $[0, 1]$. Normally, a bound limit V_{max} is incorporated to guarantee the value of the velocity be forced into a boundary $[-V_{max}, V_{max}]$ for the purpose of divergence avoidance.

To represent the velocity as the probability for selection of 1, a sigmoid transform function is incorporated to transform the velocity to the range of $(0, 1)$:

$$S(v_{id}) = \frac{1}{1 + \exp(-v_{id})} \tag{7}$$

The resulting change in position of a particle then is defined by the following rule:

$$\begin{aligned} &\text{if } rand() < S(v_{id}) \text{ then } x_{id} = 1 \\ &\qquad\qquad \text{else } x_{id} = 0 \end{aligned} \tag{8}$$

The algorithm skeleton of PSO-P is described in Fig.2. An iteration of PSO-P comprises evaluation of each particle using the modified fitness function of (4),

and calculations of the p_i and g_i for every particle, then the velocity is derived from (6) and particles move to their new positions according to (7) and (8). The iteration repeated until some termination condition is met, such as a maximum amount of cycles performed or a satisfied solution is found.

```
Procedure PSO-P
/*Initialization*/
    Input data
    Randomly generate initial particles positions and their velocities
    Parameter setting
/*Main Iteration Loop*/
While (end condition not met) do
    Solution evaluation according to (4)
    Calculate p_id and g_id for every particle
    Calculate velocities for every particle according to (6)
    Generate new particle positions according to (7) and (8)
End
```

Fig. 2. Algorithm Skeleton of PSO-P

3 The Modified PSO with Repair Operator

Although penalty function technique works well for most of the applications of PSO to the constrained problems, it contains some parameter setting problem. If the penalty parameter values are too high, the optimization algorithms usually get trapped in local minima. On the other hand, if penalty values are too low, they can hardly detect feasible optimal solutions. Furthermore, since the penalty function technique does not use the problem specific information, the final results are often not satisfied in dealing with CO problems.

This paper proposes a modified PSO with repair operator specially designed for MKP. The modified algorithm, denoted as PSO-R, is based on the structure of the binary PSO model described in the previous section, in combination with a problem-specific repair operator to guarantee feasible solutions. Fig.3 describes the pseudo code of PSO-R.

Instead of using the penalty function technique, PSO-R incorporates a repair operator to repair the solutions found by the particles. This idea comes from Chu and Beasley[1]. The general idea behind this method is described very briefly as follows.

The repair operator utilizes the notion of the pseudo-utility ratios derived from the surrogate duality approach. The surrogate relaxation problem of the MKP can be defined as:

$$\text{maximize} \quad f = \sum_{j=1}^{n} p_j x_j \tag{9}$$

```
Procedure PSO-R
/*Initialization*/
    Input data
    Calculate surrogate multipliers and pseudo-utility
    Sort and renumber data according to decreasing order of pseudo-utility
    Generate initial particles positions and their velocities
    Parameter setting
/*Main Iteration Loop*/
While (end condition not met) do
    Solution repair
    Solution evaluation
    Calculate p_id and g_id for every particle
    Calculate velocities for every particle
    Generate new particle positions
End
```

Fig. 3. Algorithm Skeleton of PSO-R

$$\text{subject to } \sum_{j=1}^{n}(\sum_{i=1}^{m}\omega_i r_{ij})x_j \leq \sum_{i=1}^{m}\omega_i b_i \tag{10}$$

$$x_j \in \{0,1\}, j = 1,2,...,n \tag{11}$$

where $\omega = \{\omega_1, \ldots, \omega_m\}$ is a set of surrogate multipliers (or weights) of some positive real numbers. One of the simplest methods to obtain reasonably good surrogate weights is to solve the LP relaxation of the original MKP and to use the values of the dual variables as the weights. In other words, ω_i is set equal to the shadow price of the ith constraint in the LP relaxation of the MKP.

After calculating the surrogate weights, the pseudo-utility is then defined as:

$$u_j = \frac{p_j}{\sum_{i=1}^{m}\omega_i r_{ij}} \tag{12}$$

The repair operator consists of two phases that is based on the value of u_j. The first phase, which is called DROP phase, examines each bit of the solution in increasing order of u_j and changes the value of the bit from one to zero if feasibility is violated. The second phase, which is called ADD phase, reverses the process by examining each bit in decreasing order of u_j and changes the value of the bit from zero to one as long as feasibility is not violated. To achieve an efficient implementation of the repair operator, at the initialization step, we sort and renumber variables of the original MKP problem according to the decreasing order of their u_j's. The pseudo-code for the repair operator is given in fig.4.

Although the repair operator takes some extra time at each iteration, from the description of the procedure of PSO-R, we can see that the computational complexity of the repair operator, as well as each iteration of PSO-R, is $\mathcal{O}(mn)$,

Repair Operator for PSO-R
Let: R_i = the accumulated resources of constraint i in S
Initialize $R_i = \sum_{j=1}^{n} r_{ij} S[j], \forall i \in I$
$j = n$
While ($R_i > b_i$, for any $i \in I$) do /* DROP phase */
if $S[j] = 1$ then
$S[j] = 0; R_i = R_i - r_{ij}, \forall i \in I$
endif
$j = j - 1$;
endwhile
for $j = 1$ to n do /* ADD phase */
if $S[j] = 0$ and $R_i + r_{ij} < b_i, \forall i \in I$ then
$S[j] = 1; R_i = R_i + r_{ij}, \forall i \in I$;
endif
end for

Fig. 4. Pseudo Code of the Repair Operator

which is the same as that of PSO-P. So, PSO-R takes just a little computational time over PSO-P.

4 Experimental Results

Since we have not found any literature concerning the PSO algorithm applied to the MKP problems, we select some benchmarks of MKP from OR-Library to test PSO-R, and we compare the results of PSO-R with that of PSO-P.

To make a fair comparison, we set the same parameter values for both the PSO-R and PSO-P: $\varphi_1 = \varphi_2 = 2$, $v_{max} = 2$, the number of particles is set equal to the number of objects of the problem, and we use the ring topology as the neighborhood structure with number of neighbors set to 2. These parameter settings are regarded as optimal to the standard PSO algorithms [7].

Fig.5 describes the typical performance of PSO-P and PSO-R on a MKP instance with 50 objects and 5 resource constraints. The x-axis describes the number of executed cycles, while the y-axis describes the best fitness value that is averaged over 30 runs. From this diagram, we can see clearly that PSO-R outperforms PSO-P with quick convergence to satisfied solution, and with better solution quality.

Tab.1 shows the experimental results of PSO-R and PSO-P over 7 benchmarks named mknap1 in OR-Library. All the tests are ran with 500 executed cycles. The first column indicates the problem index, the next two columns describe the problem dimension, where n is the number of objects and m is the number of constraints. The next column is the best-known solutions from OR-Library. The final 4 columns report the best and average solutions over 30 runs of PSO-P and PSO-R respectively. For all the 7 instances of mknap1 that we tested, both PSO-R and PSO-P are able to find good solutions, but PSO-R finds better solutions than that of PSO-P as the size of the problem increases.

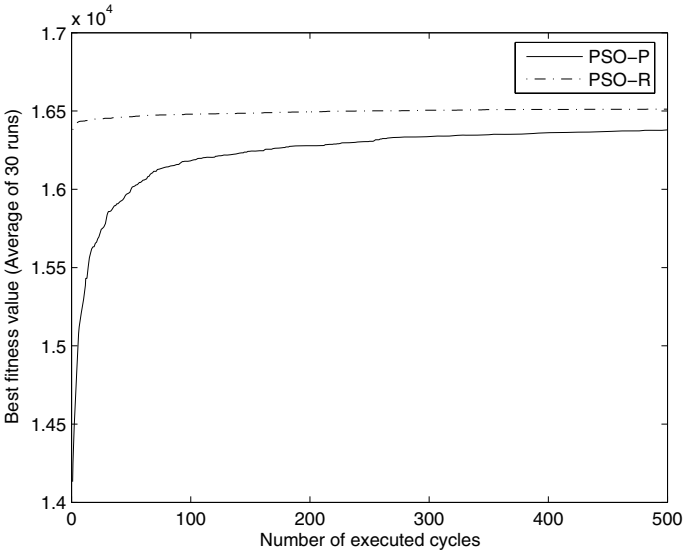

Fig. 5. Typical Performance of PSO-R and PSO-P

Table 1. Experimental Results of mknap1

N^o	n	m	Best known	PSO-P		PSO-R	
				Best	Avg.	Best	Avg.
1	6	10	3800	3800	3800	3800	3800
2	10	10	8706.1	8706.1	8570.7	8706.1	8706.1
3	15	10	4015	4015	4014.7	4015	4015
4	20	10	6120	6120	6118	6120	6119.3
5	28	10	12400	12400	12394	12400	12395
6	39	5	10618	10618	10572	10618	10592
7	50	5	16537	16491	16389	16537	16510

We also compare the PSO-R with PSO-P on some bigger size MKP instances in OR-Library, which are considered to be rather difficult for optimization approaches. The tested sets are 5.100 and 10.100, which has 5 constraints, 100 objects and 10 constraints, 100 objects respectively. We test first 5 instances of each set with maximum number of cycles of 2000, and Tab.2 reports the test results.

The first column of Tab.2 indicates the instance name, the second column is the best-known solutions from the OR-Library, and the next 4 columns record the best and average solutions over 30 runs of PSO-P and PSO-R respectively.

From Tab.2 we can see that PSO-R clearly outperforms PSO-P in all the tested instances. While PSO-P meets some difficulties in dealing with large size MKP problems, PSO-R is still able to find good solutions. Actually, PSO-R has found 5 best solutions out of 10 instances.

Table 2. Experimental Results of 5.100 and 10.100

Instance Name	Best known	PSO-P		PSO-R	
		Best	Avg.	Best	Avg.
5.100.00	24381	22525	22013	24381	24356
5.100.01	24274	22244	21719	24258	24036
5.100.02	23551	21822	21050	23551	23523
5.100.03	23534	22057	21413	23527	23481
5.100.04	23991	22167	21677	23966	23966
10.100.00	23064	20895	20458	23057	23050
10.100.01	22801	20663	20089	22781	22668
10.100.02	22131	20058	19582	22131	22029
10.100.03	22772	20908	20446	22772	22733
10.100.04	22751	20488	20025	22751	22632

5 Discussion

5.1 PSO Applied to CO

PSO has gained reputation in the field of function optimization problems, but few encouraging applications are recorded in the field of combinatorial optimization problems. The main reason is that the PSO is famous for its robustness regardless of the type of the fitness function, most of PSO rarely use the characteristic information of the problem instance, which is quite critical in tacking the combinatorial optimization problems.

Up to our knowledge, there has been no literature available concerning the application of PSO to MKP. The main purpose of this paper is to propose that the PSO technique is also effective in dealing with combinatorial optimization problems, rather than showing that PSO-R is the best algorithm overcoming MKP. The algorithm methodology, as well as the parameter setting in PSO-R, is quite normal method directly get from results of other PSO literature, however, the results presented in previous section are quite promising, indicating the potential of PSO in dealing with such kind of combinatorial optimization problems.

5.2 Role of the Repair Operator

The repair operator incorporated in PSO-R plays a critical role in quickly finding good solutions, this lies in two sides:

First, the repair operator itself improves the solution quality. Although the repair operator alone acts as a problem-specific greedy search method that can only find rather poor solutions, in cooperation with standard PSO, the repair operator acts as a local search to the solutions found by the standard PSO, which greatly improves the solution quality.

Secondly, the repair operator acts as a filter that makes all the solutions generated in the iteration being transferred to the feasible solution domain,

which makes the algorithm search around the quite promising area comparing to the normal penalty incorporated method.

The utility of a repair operator is important in applying the PSO to MKP. A good repair operator is critical in quick convergence. To find a good repair operator in other combinatorial optimization problems will be helpful for the PSO implementations.

6 Conclusions

This paper proposes a first implementation of Particle Swarm Optimization to the well-known multidimensional knapsack problem. Instead of the incorporation of the penalty function technique usually used for the constrained problems, we utilize a problem-specific repair operator to guarantee feasible solutions at each iteration cycle.

Computational results show that the modified PSO algorithm outperforms the standard PSO in MKP problems of various characteristics. Although the computational results of the modified algorithm are still not as good as the state-of-art algorithm proposed by Vasquez and Hao [11], the fact of its simplicity, quickness and that it is able to deal with large size MKP problems indicates its potential in dealing with such combinatorial optimization problems.

The repair operator technique plays a critical role in finding better solutions quickly. The procedure of our modified PSO algorithm indicates that this technique can be implemented in other combinatorial optimization problems. Further works will be on applying this technique in constrained integer programming with focus on how to apply the problem-specific information into some repair operators.

References

1. P.C. Chu and J.E.Beasley. A Genetic Algorithm for the Multidimensional Knapsack Problem. Journal of Heuristics, 4: 63-86 (1998)
2. Gavish, B. and H. Pirkuo. Efficient Algorithms for Solving Multiconstraint zero-One Knapsack Problems to Optimality, Mathematical Programming 31, 78-105. 1985
3. Gilmore, P.C. and R.E. Gomory. The Theory and Computation of Knapsack Functions, Operations Research 14, 1045-1075. 1966
4. X. Hu, R. Eberhart. Solving Constrained Nonlinear Optimization Problems with Particle Swarm Optimization. Proceedings of the Sixth World Multiconference on Systemics, Cybernetics and Informatics 2002 (SCI 2002), Orlando, USA. 2002 [3]
5. J. Kennedy, R.C. Eberhart. Particle Swarm Optimization. Proceedings of the International Conference on Neural Networks, Perth, Australiz, 1995, IEEE, Piscataway, 1995, pp, 1942-1948.
6. J. Kennedy, R.C. Eberhart. A Discrete Binary Version of the Particle Swarm Algorithm. In: Proc. 1997 Conf. On systems, Man, and Cybernetics. Piscataway, NJ: IEEE Service Center, 1997, 4104-4109.

7. J. Kennedy, R.C. Eberhart and Y. Shi. Swarm Intelligence. Morgan Kaufmann Publishers. 2001.
8. S. Martello, P. Toth. Knapsack Problems, Algorithms and Computer Implementations. John Wiley & Sons Ltd. 1990.
9. K.E. Parsopoulos, M.N. Vrahatis. Particle Swarm Optimization Method for Constrained Optimization ProblemsP. Sincak, J. Vascak, V. Kvasnicka, J. Pospichal (eds.), Intelligent Technologies - Theory and Applications: New Trends in Intelligent Technologies, pp. 214-220, IOS Press (Frontiers in Artificial Intelligence and Applications series, Vol. 76), 2002, ISBN: 1-58603-256-9.
10. Shih, W. A Branch and Bound method for the Multiconstraint Zero-One Knapsack Problem, Journal of the Operational Research Society 30, 369-378 1979
11. M. Vasquez and J.K Hao, A Hybrid Approach for the 0-1 Multidimensional Knapsack Problem. In Proceedings of the 17^{th} International Joint Conference on Artificial Intelligence, 2001, (pp.328-333). San Francisco, Morgan Kaufmann.