A Fast Particle Swarm Optimization Algorithm for Large Scale Multidimensional Knapsack Problem*

Kunpeng KANG*

School of Computer and Information Technology, Shangqiu Normal University, Shangqiu 476000, China

Abstract

Multidimensional knapsack problem (MKP) is very important and NP-hard. A strategy of mimetic profit/weight ratio (P/W-Rt) is proposed and accordingly, a fast particle swarm optimization algorithm is proposed. The most important merit of this strategy is that it can effectively solve MKP with arbitrary number of constraints under similar computing costs as one dimensional KP. Experiments illustrate the robust performance, efficiency, and computing costs independence with instance scale.

Keywords: Multidimensional Knapsack Problem; Particle Swarm Optimization; Profit/Weight Ratio; Metaheuristics

1 Introduction

The knapsack problem (KP) is one of the most studied combinatorial optimization problems. The well known multidimensional knapsack problem (MKP) is a generalization of KP with multiple knapsacks and NP-hard. A subset of $n$ given objects are chosen in such a way that the total profit of the selected objects is maximized while a set of knapsack constraints are satisfied, which is formulated as follows [1, 2]:

\[
\begin{align*}
\max & \sum_{j=1}^{n} p_j x_j \\
\text{s.t.} & \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \ i \in M = \{1, \ldots, m\}, \\
& x_j \in \{0, 1\}, \ j \in N = \{1, \ldots, n\},
\end{align*}
\]  

where $p_j$, $w_{ij}$, $c_i$ are positive integers, $\forall i \in M$, $j \in N$. It is often used as a platform to evaluate new metaheuristics [3]. Furthermore, its difficulty increases with the number of constraints.

The KP/MKP has many practical applications such as cargo loading, cutting stock, capital budgeting and project selection applications [2] among others. KP/MKP model have also been used

*Project supported by the Basic and Frontier Research Programs of Henan Science and Technology Committee of China (Grant No.102300410257)

Email address: zxy_kkp@126.com (Kunpeng KANG).
the to generate covering inequalities, whereas it is used in the area of cryptography. Hartvigsen
[4] showed that the vote-trading problem is polynomially equivalent to an integer program and
more precisely to a KP.

A canonical particle swarm optimizer (PSO) [9, 10] maintains a swarm of particles and each
individual is composed of three $n$-dimensional vectors, which are the current position $x_i$, the
previous best position $p_i$, and the velocity $v_i$, where $n$ is the dimension of the search space. The
current position $x_i = (x_{i1}, \cdots, x_{in})$ can be considered as a set of coordinates describing a point
in space. The best solution found so far is stored in $p_i = (p_{i1}, \cdots, p_{in})$. New points are chosen by
adding $v_i = (v_{i1}, \cdots, v_{in})$ coordinates to $x_i$, and the algorithm operates by adjusting $v_i$, which
can be seen as a step size. The updating equations are

$$v_{ij}^{k+1} = w \times v_{ij}^k + c_1 \times r_1^k \times (p_{ij}^k - x_{ij}^k) + c_2 \times r_2^k \times (p_{gj}^k - x_{ij}^k) \tag{2}$$

$$x_{ij}^{k+1} = x_{ij}^k + v_{ij}^{k+1} \tag{3}$$

where $v_{ij}^k$ and $x_{ij}^k$ are the $j$-th dimensional velocity/position of particle $i$ in cycle $k$; $p_{ij}^k$ is the $j$-th
dimension of personal best (pbest) positions of particle $i$; $p_{gj}^k$ is the $j$-th dimension of the gbest; $w$ is inertia weight; $c_1$ and $c_2$ are accelerating coefficients; $r_1, r_2$ are random numbers in $(0, 1)$.

A mimetic P/W-Rt idea is presented to transform MKP into KP with single knapsack, for
which there are many existed solving methods. The transformed MKP is solved with the greedy
idea of the P/W-Rt of the objects as a KP. Some repair strategies for the overweight solutions and
a fast PSO are proposed to solve the MKP. Of course, the main effects of the proposed algorithm
are to solve the small scale MKP effectively and to give an upper bound for the large scale MKP
efficiently. It is obvious that there are usually a gap between the optimal results that algorithm
found and its true optima.

The paper is organized as follows. The fast PSO (fPSO) algorithm for MKP is proposed in
Section 2, including the strategy of how to transform MKP into KP, mimetic P/W-Rt idea for
the transformed KP, repair strategy for the overweight solution. The extensive numerical experi-
ments are conducted in Section 3, including performance comparison with PSO and independence
validation between CPU time and the number of constraints. Section 4 concludes this paper.

## 2 Related Works

Depending on the nature of the solution, the algorithms for MKP are divided into two groups,
namely exact algorithms [1], which strive for exact solutions, and heuristic algorithms [5], where
we are satisfied in near-optimal solutions. Several exact algorithms gave different upper bounds
for the optimal solutions [6]. However, computing the optimal surrogate multipliers becomes
no applicable for large values of $m$ and $n$ and other strategies or heuristic approaches must be
introduced: greedy-like assignment, LP-based search and surrogate duality information [3, 7] are
some examples. Some papers using evolutionary strategies have emerged which have shown to
be well suited for solving large MKP instances. Chu and Beasley [8] have developed a genetic
algorithm that searches only into the feasible search space. They use a repair operator based on
the surrogate multipliers of some suitable surrogate problem.

Moraga et al. [11] have recently applied another metaheuristic called Metaheuristic for Ran-
domized Priority Search (Meta-RaPs) on the MKP. It combines the use of randomness with
greedy heuristics to construct and improve feasible solutions. The final quality of their solutions
on a set of available instances is near to the one obtained by Chu and Beasley [8] while using a reasonable computational effort. The MKP is an NP-hard problem and its difficulty increases with the number of constraints. As a guide, instances with 500 variables and 30 constraints cannot be solved optimally within a reasonable amount of computing time and memory requirement. From an implementation point of view when evaluating new methods for the MKP, the majority of the instances used have a small number of constraints.

3 Proposed Strategies and Fast PSO

3.1 Solution representation and construction

A binary string \( x = \{x_1, \ldots, x_n\} \in \{0, 1\}^n \) represents a potential solution. \( x_j = 0 \) means the object \( j \) is not selected and vice versa. An infeasible solution is one for which at least one of the knapsack constraints is violated, i.e., \( \sum_{j=1}^{n} w_{ij} x_j > c_i \) for some \( i \in M \). Velocity is a float number, which is truncated into binary variable with the following \( \text{sig} \) function [12]

\[
x_{k+1}^{id} = \begin{cases} 
1, & \text{if } r \leq \text{sig}(v_{ij}^k) \\
0, & \text{if } r > \text{sig}(v_{ij}^k)
\end{cases}
\]

(4)

where \( r \) is a uniform random number in \((0, 1)\) and \( \text{sig} \) function \( \text{sig}(v) = \frac{1}{1+e^{-v}} \).

3.2 Transforming MKP into KP

**Definition 1 [One Norm]:** One norm of an \( n \) dimensional vector \( \alpha \) is \( \| \alpha \|_1 = \sum_{i=1}^{n} |a_i| \).

The strategy of transforming MKP into KP is, in brief, like to compute the surrogate constraint, and with surrogate multipliers being constant “1”[13]. Concretely, the \( m \times n \) weight matrix \( w \) being summed along the columns, i.e., the one norm of the column vector \( w_j \) in matrix \( w^{m \times n} \), is looked on as the coefficient of the decision variable \( x_j \). Then the constrained inequalities groups \( \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \; i \in \{1, \ldots, m\} \) is transformed into the inequality of \( \sum_{j=1}^{n} (\sum_{i=1}^{m} w_{ij}) x_j \leq \sum_{i=1}^{m} c_i \). Then the MKP model (1) is transformed into the following single dimensional knapsack problem.

\[
\begin{align*}
\text{max} \quad & p \cdot x \\
\text{s.t.} \quad & \sum_{j=1}^{n} (\sum_{i=1}^{m} w_{ij}) x_j \leq \sum_{i=1}^{m} c_i, \\
& x_j \in \{0, 1\}
\end{align*}
\]

(5)

It is evident that the surrogate problem (5) is much easier than the initial (1) to obtain the optimal solution. Furthermore, it is easy to see that any optimal solution for the surrogate problem (5) generates an upper bound for the optimal solution of the initial MKP (1), i.e.

**Proposition 2:** Any feasible solution of model (1) is also an feasible solution of model (5).

3.3 Mimetic profit/weight ratio construction for MKP

The most important heuristics for KP is to use the profit/weight utility criterion for selecting the objects to be added into a solution [13]. It is convenient to compute the P/W-Rt for the single
knapsack case (KP). For the general MKP, one effective computing method for the mimetic P/W-Rt is detailed presented.

We assume that MKP problem has $m$ knapsacks and $n$ objects indicated as Equ.(1), i.e.,
$$
\sum_{j=1}^{n} w_{ij} x_j \leq c_i, \ i = 1, ..., m.
$$

The mimetic P/W-Rt for MKP is computed as follows:

1) Compute P/W-Rt for $i$th knapsack, $c_1/w_{i1}, ..., c_j/w_{ij}, ..., c_n/w_{im} \ i = 1, ..., m$;

2) Compute mimetic P/W-Rt for the $j$th item via
$$
m_{PW}r_j := \sum_{i=1}^{m} \lambda_i \cdot (c_i/w_{ij}) = \lambda \cdot (c./w_j)
$$
denotes element-by-element division and $a \cdot b$ denotes the dot production of vectors $a$ and $b$.

3) Construct the solution according to the criterion of the mimetic P/W-Rt.

The weight coefficient $m_{PW}r_j = \sum_{i=1}^{m} \lambda_i \cdot (c_i/w_{ij})$ is a convex combination of the respective profit/weight ratio of the $j$th object for all the constraints. After computing the mimetic profit/weight ratio of multidimensional knapsack problem, we can construct new solution or repair the infeasible solution on the same basis of P/W-Rt ($m_{PW}r_j$) as the case of single knapsack problem. This heuristic/greedy knowledge is now available to design the hybrid algorithm.

### 3.4 Repair strategy based on mimetic P/W-Rt

Feasible solutions are not absolute requirement for the PSO evolution (velocity and position updating), which is a difference with the well-known CHUGA [8]. Feasible or infeasible solutions require similar, however, discrepant evaluating method. The mimetic P/W-Rt based repair strategy examines each bit of the solution string in increasing order of mimetic P/W-Rt of MKP and changes the bit from one to zero if feasibility is violated. Then the fitness of the decision variable, including the crude and the repaired solutions, can be computed by $f(x) = \sum_{j=1}^{n} p_j x_j$. At the same time, such strategy bypasses the sensitive penalty parameters choice.

### 3.5 Amend strategy for the possible final infeasible solution

As we know, the obtained (near) optimal solution from KP model (5) is unnecessary optimal, even not infeasible, for the initial MKP model (1). Hence the final solution of model (5) must be verified whether it is feasible. If it is feasible, it is sure to be an upper bound the model (1). Otherwise, the repair strategy is utilized as to take out the enclosed and the worst objects according to P/W-Rts, until to satisfy all the knapsack constraints.

### 4 Numerical Experiments for fPSO on MKP

The fPSO is compared with PSO for performance evaluation with benchmarks with various constraints for independence validation between computing costs and the number of constraints based on the randomly generated instances [14]. These experiments are executed on a LENOVO T notebook with an Intel Core Duo CPU P8600, 2GB RAM and MatLab 2009a.
4.1 Performance evaluation and comparison

4.1.1 Parameters for algorithms

The goal of these experiments is to validate the necessity and efficacy of our proposed strategies compared with PSO on randomly generated MKP instances. This paper mainly focuses on the problem of transforming MKP into KP and the mimetic P/W-Rt, so the parameters fine-tuning is omitted and they are constants here. The inertia weight $w = 0.8$, cognition weight $c_1 = 2.8$, social weight $c_2 = 1.3$ and the population size $POP = 30$. The maximal iteration numbers vary with the problem size and the (instance object, corresponding maximal iteration number) pairs are as follows: (100, 500), (200, 800), (300, 800), (500, 1000), (700, 1000), (1000, 1000).

4.1.2 Penalty function for canonical PSO

A solution $x$ maybe include some good gene blocks even it is infeasible, i.e. it maybe a useful linkage between a good feasible solution and another [15]. Although the feasible-limited approach reduces the search space it makes the space rough and interrupted as the expense. An adaptive penalty function is adopted for including the promising infeasible solutions for PSO.

(1) Count the violated constraints $nVIO$ of $x$; (2) Sum the weights $ENwSum$ of the enclosed objects, and sum the knapsacks capacity $sumCap$ for all the violated constraints; (3) fitness is

$$f(x) = p \cdot x - (ENwSum - sumCap) \times \alpha(t)$$

(6)

where $\alpha(t) = \begin{cases} t \cdot (nVIO + 1)^2, & \text{if } t \leq 0.5 \times \text{ITERATION} \\ [t \cdot (nVIO + 1)]^2, & \text{if } t > 0.5 \times \text{ITERATION} \end{cases}$

(7)

and $t$/ITERATION is the current/maximal iteration number.

The penalty will increase with the violation measurement, the number of violated constraints and the current iteration number. However, a solution is sure to participate the population evolution when it has a promising fitness and a little violation, especially for the first half executing time of algorithm. Finally, the infeasible solution is more and more difficult to stay in the population with process of algorithm.

4.1.3 Experimental results and analysis

Experiments are given in Table-1 and the data are obtained from 30 independent runs. Items “Best”, “Mean”, “STD”, “Time” and “fIter” are the best, average, standard deviation, average CPU time and the average iteration number when the optimal results are found.

Observed from Table-1, it can be seen that fPSO greatly outperforms PSO on all the “Best” and “Mean” items and it has significantly and steadily better performance than PSO. It also indicates that fPSO has even better convergent ability and even more robust performance as “Best” and “Mean” items illustrate. Of course, good properties are reached at the expense of a little computational costs as illustrated in “Time” columns. The “Time” items of fPSO in Table-1 and 2 display that its computational time is not strictly increasing with the expanding of the instance size, which proves again the excellent independent behaviors of algorithm to some extent. The “STD” and “fIter” items are comparative each other for two algorithms which maybe belong to the adaptive penalty fitness function Equ.(6).
4.2 Online performance comparison for two algorithms

The best fitness found so far at every iteration of all 30 runs is averaged to further investigated the online performance of algorithms.

Observed from Figure-1 we can see that fPSO has great superiority over PSO for all these instances which illustrates that fPSO has consistent and distinct merits. Due to the selection strategy based on the proposed mimetic profit/weight ratio, fPSO outperforms PSO even from the initial stage. In a word fPSO has quickly convergent ability and robust performance with the aid of mimetic P/W-Rt idea.

4.3 Experiments on various knapsacks for large benchmarks

As illustrated above the fPSO is nearly independent with the dimensions of the MKP. Various numbers of constraints are used to validate this promising strategy with 2 to 500 knapsacks and 1000 objects. All the parameters are the same as before except for the maximal iteration numbers and algorithm running times 20 because of the statistic balance between performance and CPU time. The constraints number (Cn), the maximal iteration number (Iter), the average iteration number (AVit) when the optimal results found and the average CPU time (Time) for all 20 independent runs are presented in Table-2.

The second to fifth rows in Table-2 give the computational results of fPSO for various instances. The average iteration numbers when the optimal results found are nearly independent of the “Cn” and “Iter”, which is observed from the “AVit” and “AVit/ITER” rows. The fPSO found the optimal results at about four fifths of the algorithm for various scale constraint benchmarks from

<table>
<thead>
<tr>
<th>(m, n)</th>
<th>PSO</th>
<th>fPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Mean</td>
</tr>
<tr>
<td>(2,100)</td>
<td>28848</td>
<td>27638.4</td>
</tr>
<tr>
<td>(2,200)</td>
<td>56347</td>
<td>53736.6</td>
</tr>
<tr>
<td>(2,300)</td>
<td>44999</td>
<td>43940.7</td>
</tr>
<tr>
<td>(2,500)</td>
<td>163886</td>
<td>161887.3</td>
</tr>
<tr>
<td>(2,700)</td>
<td>178976</td>
<td>173988.1</td>
</tr>
<tr>
<td>(2,1000)</td>
<td>264135</td>
<td>258327.3</td>
</tr>
<tr>
<td>(5,100)</td>
<td>29943</td>
<td>28854.4</td>
</tr>
<tr>
<td>(5,200)</td>
<td>61214</td>
<td>59272.1</td>
</tr>
<tr>
<td>(5,300)</td>
<td>40143</td>
<td>39047.4</td>
</tr>
<tr>
<td>(5,500)</td>
<td>130737</td>
<td>128001.7</td>
</tr>
<tr>
<td>(5,700)</td>
<td>202610</td>
<td>199929.2</td>
</tr>
<tr>
<td>(5,1000)</td>
<td>277079</td>
<td>273474.5</td>
</tr>
</tbody>
</table>
Fig. 1: Average best fitness versus iteration number of PSO and fPSO.

Table 2: Cn, Iter, AVit and Time for instances with various constraint numbers

<table>
<thead>
<tr>
<th>Items</th>
<th>Items values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cn</td>
<td>2  5  10  20  30  50  70  100  500</td>
</tr>
<tr>
<td>AVit</td>
<td>739.65  780.65  1616  1628.8  1546.1  1502.5  2185.1  2481  3401.75</td>
</tr>
<tr>
<td>Time</td>
<td>55.42  14.75  44.06  25.16  32.84  29.25  42.59  41.85  212.97</td>
</tr>
<tr>
<td>AVit/Iter</td>
<td>0.7397  0.7806  0.8080  0.8144  0.7730  0.7512  0.7284  0.8270  0.6804</td>
</tr>
<tr>
<td>Time/AVit</td>
<td>0.0749  0.0189  0.0273  0.0154  0.0212  0.0195  0.0195  0.0169  0.0626</td>
</tr>
<tr>
<td>T/Ai/Cn</td>
<td>0.0375  0.0038  0.0027  0.0008  0.0007  0.0004  0.0003  0.0002  0.0001</td>
</tr>
</tbody>
</table>

2 to 500 as showed in “AVit/Iter” row. The average CPU time per iteration (Time/AVit) ranges from 0.0154 to 0.0749(s) for the different constraint instances. It is very interesting that the average CPU time costed by every iteration and every knapsack (T/Ai/KP) is strictly decreasing with the constraints increasing as illustrated at the last row in Table-2. This result tells us that the efficiency of PSO becomes higher and higher with the constraints increasing. All the computational results and the data analysis clearly indicates that the CPU time is independent of the constraint numbers for MKP. That is to say, the proposed strategy and algorithm are extraordinarily suited to solve the multidimensional knapsack problem.

5 Conclusions

A fast PSO for large scale MKP is proposed in this paper based on the novel mimetic profit/weight ratio idea. The most important feature of this idea is that it can transform MKP into KP model,
and then the greedy idea according to the P/W-Rt of the objects can therefore be utilized, therefore the abundant existent approaches for KP can be used for this question. It is obvious that any optimum of the surrogate MKP efficiently generates an upper bound for the optimal solution of the MKP model. Experiments on performance evaluation and independence validation indicate the robust performance and efficiency for large scale benchmarks, and most importantly, its highly adaptable to the size of MKP instance.

References