Chapter 5

Binary MOPSO for 0/1 Knapsack Problem

The 0/1 knapsack problem is a widely studied problem due to its practical importance. In the previous years, the generalization of this problem has been very much studied. Many papers regarding multi-objective knapsack problem and the corresponding algorithms proposed for solving them can be found in the literature [7, 26, 57, 58, 76, 99, 119, 121, 143, 154, 177, 204, 205]. In this work, we have developed and implemented a binary MOPSO algorithm for solving multi-objective 0/1 knapsack problem and also performed a comparison with the NSGA-II algorithm proposed by Deb et al. [41].

This chapter describes the multi-objective 0/1 knapsack problem in Section (5.1). In Section (5.2), we propose a binary multi-objective binary MOPSO algorithm to solve this problem. Section (5.3) shows and discusses the experimental results.
5.1 Multi-objective 0/1 Knapsack Problem

5.1.1 Problem Statement

Generally, a 0/1 knapsack problem consists of a set of items, weight and profit associated with each item, and an upper bound for the capacity of the knapsack. The task is to find a subset of items which maximizes the total of the profits in the subset, yet all the selected items fit into the knapsack, i.e. the total weight does not exceed the given capacity [121]. This single objective problem can be extended directly to a multi-objective case by allowing an arbitrary number of the knapsacks. Formally, the multi-objective 0/1 knapsack problem is defined through 5.1 and 5.2.

Given a set of $m$ items and a set of $n$ knapsacks with

\[ p_{i,j} = \text{profit of item } j \text{ according to knapsack } i, \]

\[ w_{i,j} = \text{weight of item } j \text{ according to knapsack } i, \]

\[ c_i = \text{capacity of knapsack } i, \] (5.1)

find a vector \( x = (x_1, x_2, \ldots, x_m) \in \{0, 1\}^m \) such that

\[ \forall i \in \{1, 2, \ldots, n\} : \sum_{j=1}^{m} w_{i,j} \cdot x_j \leq c_i \]

and for which \( f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \) is maximum, where

\[ f_i(x) = \sum_{j=1}^{m} p_{i,j} \cdot x_j \] (5.2)

and \( x_j = 1 \) iff item \( j \) is selected.

In order to obtain reliable and sound result, three different test problems are investigated where both the number of knapsacks (i.e., number of objectives) and the number of items are varied. Two knapsacks (i.e., two objectives) are taken under consideration in combination with 250, 500, and 750 items. Following suggestions
in [121], profits and weights are chosen, where \( p_{i,j} \) and \( w_{i,j} \) are random integers in the interval \([10, 100]\). Also as reported in [121], about half of the items are expected to be in the optimal solutions when this type of knapsack capacity is used. Thus, the knapsack capacities are normally set to half the total weight according to the corresponding knapsack as indicated in 5.3:

\[
c_i = 0.5 \cdot \sum_{j=1}^{m} w_{i,j}
\]

(5.3)

During fitness evaluation, the order in which items are deleted is determined by the maximum profit/weight ratio per item; for item \( j \) the maximum profit/weight ratio \( q_j \) is given by (5.4).

\[
q_j = \max_{i=1}^{n} \{ \frac{p_{i,j}}{w_{i,j}} \}
\]

(5.4)

### 5.2 Binary MOPSO Algorithm

To change the real valued MOPSO algorithm to binary MOPSO, we need to change the concept of velocity as a probability of change of position from rate of change of position. Here we replace the real valued velocity and position update equations with the discrete velocity and position update equations proposed in [93]. The velocity and position update equations for the binary MOPSO is given in equations (5.5) and (5.6).

\[
v_{id}(t + 1) = wv_{id}(t) + c_1 r_1(p_{id}(t) - x_{id}(t)) + c_2 r_2(p_{gd}(t) - x_{id}(t))
\]

(5.5)

where \( r_1() \) and \( r_2() \) are two separately generated uniformly distributed random values in the range \([0, 1]\), \( w \) is inertia weight (or inertia factor) which is employed to
control the impact of the previous history of velocities on the current velocity of a
given particle, $c_1$ and $c_2$ are constants known as acceleration coefficients (or learning
factors).

$$S(v_{id}) = 1/(1 + exp(-v_{id}))$$

$$if \ (S(v_{id}) > \ rand())$$

$$x_{id} = 1$$

$$else \ x_{id} = 0$$

(5.6)

where $S(v_{id})$ is the sigmoid function which is used to bound the velocity $v_{id}$ in $[0, 1]$, \( rand() \) is a quasirandom number selected from a uniform distribution in $[0.0, 1.0]$. Here the position of the particles in $d$-dimension are represented as string of bits denoted by $X_i = (x_{i1}, x_{i2}, ..., x_{id})^T$. The velocity of the particle is represented as $V_i = (v_{i1}, v_{i2}, ..., v_{id})^T$. The value of $v_{id}$ for each dimension determines the probability that a bit $x_{id}$ will take one or the zero value. The actual position of particle in $d$ dimensional space is ephemeral.

The guide for a particle, to direct the search, is selected by applying the k-mediod clustering technique on the external archive. One of the non-dominated particles from the less dense cluster is selected as a guide for the each particle. The clustering technique is also used as a secondary strategy for maintaining the repository. The algorithm for the proposed binary PSO is shown in Algorithm 3.
Algorithm 3 Binary Multi-Objective Particle Swarm Optimization

Initialize the swarm, SWARM

for $i=1$ to $max$  
  /*$max =$Maximum size of the swarm*/ 
  Initialize SWARM[$i$];
end for

Evaluate fitness of each of the particles in SWARM

Find the non-dominated particle from the SWARM and store them in external archive called EX-ARCHIVE.

Initialize the memory for each particle. (This memory serves as a guide for the particle to travel through the search space).

for $i=1$ to $max$ do
  $pbest[i]=$SWARM[$i$];
  $I=0$
  /*$I =$Iteration Count*/
end for

while ($I < I_{Max}$) do
  /*$I_{Max}$ is the maximum number of iterations*/
  for each particle do
    Update the velocity and position of particle using (5.5) and (5.6)
    Evaluate the fitness of the updated particle in the SWARM.
    Update the $pbest[i]$.
  end for

  Update the content of EX-ARCHIVE
  /*Here we need to determine whether the new solution should be added to the external archive or not. This is determined by performing a non-domination test on the external archive. As a secondary strategy we can apply clustering when the archive size exceeds pre-specified value.*/
  $I = I + 1$
end while

Report the results in the EX-ARCHIVE
Table 5.1: An Instance of a 0/1 Knapsack Problem: Items Associated with Profits and Weights

<table>
<thead>
<tr>
<th>Item Sl. No</th>
<th>Profit</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>63</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
<td>31</td>
</tr>
<tr>
<td>10</td>
<td>32</td>
<td>16</td>
</tr>
</tbody>
</table>

5.2.1 Particle Representation

Each particle in the swarm is represented as binary string of length $m$ (number of items) where each bit corresponds to presence or absence of the item in the knapsacks. The particle representation for the multi-objective 0/1 knapsack problem by considering for instance two knapsacks (i.e., two objectives) and for total of 10 distinct items is illustrated in Figure 5.1. Let us assume, for instance that the respective weights and profits associated with these 10 items are as per Table 5.1.

![Figure 5.1: Representation of a Particle in Multi-objective 0/1 Knapsack Problem](image)

As per the problem requirements, once one item is put in a knapsack, i.e. NS1, then that item cannot be put in other knapsack i.e., NS2. Accordingly, as per Figure
5.1, five items i.e., item1, item2, item4, item8, and item9 have been placed in NS1 and item3, item5, item6, item7, and item10 have been placed in NS2.

The evaluation of the fitness function for this particle is illustrated in Figure 5.2:

![Fitness Evaluation Illustration](image)

After evaluating fitness for each particle in the above manner, the further steps are followed as per binary MOPSO algorithm (Algorithm 3).

### 5.2.2 Guide Selection Strategy

Guide plays an important role in directing the search direction to the particle in the search space. The literature provides various guide selection strategies for guiding the search. In our algorithm, we adopt the k-medoid clustering technique for selection of the guide. The archive particles are clustered in objective space and a particle
from the less dense cluster is selected as the guide for \(i^{th}\) particles. The clustering technique is repeated for every particle of the swarm. More than one particle of the swarm can be assigned a single common guide. The clustering strategy is adopted if there are more than two particles in the archive. However, if there are less than or equal to two particles in the archive, we randomly assign one particle from the archive as the guide to the \(i^{th}\) particle.

5.3 Experimental Study

In experimental study, we describe the experimental environment and discuss the results of the experiments in Subsections (5.3.1) and (5.3.2) respectively.

5.3.1 Experimental Setup

We have applied our proposed MOPSO algorithm by considering 100 iterations and variable population size. The experiment is performed in Core2Duo processor, 1GB RAM and Matlab 2009b. The parameter settings for the experiments is shown in Table (5.2). This table specifies the inertia factor \(w\), the personal cognition factor \(c_1\), and the social interaction factor \(c_2\) used in the algorithm.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>No. of Item</th>
<th>PopSize*</th>
<th>(w)</th>
<th>(c_1)</th>
<th>(c_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary MOPSO</td>
<td>250</td>
<td>150</td>
<td>1.0</td>
<td>1.0</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>200</td>
<td>1.0</td>
<td>1.0</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>750</td>
<td>250</td>
<td>1.0</td>
<td>1.0</td>
<td>2.5</td>
</tr>
</tbody>
</table>

*PopSize represents the population size
### Table 5.3: Set Coverage Metric

<table>
<thead>
<tr>
<th>No. of Elements</th>
<th>Cases</th>
<th>$C(A, B)$</th>
<th>$C(B, A)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>Best</td>
<td>1.000</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>0.802</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>0.996</td>
<td>0.001</td>
</tr>
<tr>
<td>500</td>
<td>Best</td>
<td>1.000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>750</td>
<td>Best</td>
<td>1.000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>1.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

### 5.3.2 Results and Discussion

This section illustrates the results of implementation of multi-objective 0/1 knapsack problem. The Figure 5.3 shows that our binary MOPSO algorithm shows better results than the NSGA-II algorithm for some range of objective values. We can conclude from the Figure 5.3 that the our binary MOPSO algorithm has better density and has better diversity. To analyze the performance more effectively, we use the quantitative approach called set coverage metric. The results are tabulated in Table (5.3). While calculating the set coverage metric, $A$ represents the binary MOPSO algorithm and $B$ represents the NSGA-II algorithm. We perform a weak domination test on $A$ and $B$ to find out how much $A$ covers $B$ and vice versa. The results clearly illustrate the better performance of binary MOPSO over NSGA-II.
Figure 5.3: Pareto Fronts of Multi-objective 0/1 Knapsack Problem