A HEURISTIC FOR THE MULTIDIMENSIONAL 0-1 KNAPSACK PROBLEM

Andre Renato Sales Amaral (UTL)
andre.r.s.amaral@gmail.com
Jose Rui Figueira (INPL)
jose.figueira@mines.inpl-nancy.fr

This paper deals with the concepts of core and core problems for multidimensional 0-1 knapsack (MKP) problems. Core problems were introduced by Balas and Zemel (1980) and have been widely studied in the literature. The core refers to the set of variables for which it is hard to decide what their value will be in an optimal solution; and the core problem is a reduction of the original problem in which only the core is considered, i.e. the variables with high (low) efficiency measures are fixed to 1(0). However, it is not obvious how to determine the efficiency function that yields efficiency measures for the MKP. In a study by Puchinger et al. (2010) an approximate core is generated by using the dual efficiency measures as an approximation of the efficiency function; and the corresponding approximate core problems are solved in order to obtain near optimal MKP solutions. In the same line of research we propose in this paper a different way of exploiting the concept of core. Instead of using dual efficiency measures, our exploitation tries to obtain a statistical approximation of the true efficiency function. Computational results on a large set of instances are presented.

Palavras-chaves: Integer Programming, Heuristics, Knapsack Problem
1. Introduction

This paper deals with the concepts of core and core problems for multidimensional 0-1 knapsack (MKP) problems.

The MKP is an extension of the classical 0-1 knapsack (KP) problem to several constraints. The latter can be defined as the selection of a subset of projects (portfolio), which maximize the overall profit, without the possibility of surpassing a given budget that imposes a restriction on the summation of the weights (costs) related to the chosen projects. The MKP, instead of dealing with a single resource, deals with several resources and constraints, which can be stated as follows.

MKP:
maximize
\[ f(x_1, x_2, \ldots, x_n) = \sum_{j \in J} p_j x_j \]
Subject to:
\[ \sum_{j \in J} w_{ij} x_j \leq W_i \quad (i \in I) \]
\[ x_j \in \{0,1\} \quad (j \in J) \]

where \( J = \{1, \ldots, n\} \) is the set of all the individual projects and \( I = \{1, \ldots, m\} \) is the set of indices related with the whole set of constraints; \( p_j > 0 \) represents the profit of project \( j \in J \); \( w_{ij} > 0 \) is the weight associated to each project \( j \in J \) with respect to the consumption of resource \( i \in I \); \( W_i > 0 \) is the availability of resource \( i \in I \); and, \( x_j \) is a Boolean variable indicating whether the project \( j \) is selected \((x_j = 1)\) or not \((x_j = 0)\). The objective is to select the best portfolio (set of projects), that is, the ones leading to the maximum overall profit, so as to not exceed any knapsack capacities or resources availability (e.g. Kellerer et al., 2004). The KP is known to be NP-Hard and, therefore, the MKP is NP-Hard too.
2. Some Recent Developments

Chu and Beasley (1998) proposed a genetic algorithm to deal with the MKP and presented a set of benchmark instances, which can be found in the well-known OR-Library. For $n \in \{100, 250, 500\}$, $m \in \{5, 10, 30\}$ and tightness ratio $\alpha \in \{0.25, 0.5, 0.75\}$, Chu and Beasley (1998) generated ten instances, which totals 270 instances. Vasquez and Vimont (2005) proposed a cooperative approach (linear programming and tabu search) for solving the MKP and applied it to the Chu and Beasley's instances. As a result they obtained better sub-optimal solutions, but the CPU time was rather huge. Wilbaut and Hanafi (2009) proposed an enumerative method, which improved some sub-optimal solutions of the previous work. Boussier et al. (2010) proposed an exact method that obtains optimal solutions for some of the large size instances in the OR-Library. Their approach uses Resolution Search to guide the search toward promising parts of the search space and uses the branch-and-bound and the Depth First Search to solve small subproblems. They pointed out that their algorithm is not suitable for the Chu and Beasley's instances with $(m, n) = (10, 500)$ and instances with $(m, n) = (30, 250)$ since their computational times are huge for most of these.

3. Core Problems

The concept of core and the resolution of core problems have their inception in the literature for single constraint 0-1 knapsack problems in the early eighties (Balas and Zemel, 1980). The core refers to the set of variables for which it is hard to decide what their value will be in an optimal solution; and the core problem is a reduction of the original problem in which only the core is considered, i.e. the variables with high (low) efficiency measures are fixed to 1(0). The core concept is based on the notion of efficiency and since the efficiency function that assigns efficiency measures to variables is not known, several efficiency definitions have been used as approximations of the efficiency function, for example, *simple efficiency*, *scaled efficiency*, *general efficiency*, and *dual efficiency* (See, Kellerer et al., 2004). Puchinger et al.
(2010) extended the core concept from the KP to the MKP. They generated an approximate core based on the dual efficiency measures and solved the corresponding approximate core problems to obtain near optimal MKP solutions.

4. The Proposed method

In this paper, we propose a different way of exploiting the concept of core. Instead of using dual efficiency measures, our exploitation tries to obtain a statistical approximation of the true efficiency function. For each instance, we sample $T$ efficiency functions at random. After these $T$ trials, we expect that one tentative efficiency measure might closely match the measure provided by the true efficiency function for MKP. The basic algorithm is presented next.

Let $MKP^{LP}$ be the linear relaxation of MKP.

Solve $MKP^{LP}$ and obtain its optimal solution vector $x^{LP}$.

For $k = 1, \ldots, T$

Change $MKP^{LP}$ into into $ILP_k$ as follows:

- Look at variables $j$ with $x_j^{LP} = 1$ and randomly fix $p$ of these to one;
- Look at variables $j$ with $x_j^{LP} = 0$ and randomly fix $q$ of these to zero;
- Any variable not fixed is declared as binary 0-1;

Solve the integer program $ILP_k$ and print its solution $x^{(k)}$;

Unfix all variables in $ILP_k$;

End For

Note: the solution $x^{(k)}$ obtaining the maximum global profit is taken as output of the algorithm.

Algorithm 1
In Algorithm 1, the solution of $\text{MKP}^{LP}$ is very quick in practice and fixing/unfixing variables can be done in $O(n)$ time. If the solution of $\text{ILP}_k$ takes time $H(n)$, then Algorithm 1 has a worst case time complexity of $O(T \times H(n))$.

5. Computational Results

The proposed core algorithm was implemented using CPLEX 12.1 Callable Library with Microsoft Visual C++ 2008. The experiments were run on a 1.73GHz Computer with an Intel Core Duo Processor and 1 GB of RAM. The algorithm was tested on all of the Chu and Beasley's instances. For all tests we adopted $T=10$ and $p=q=50\%$.

All tables present results (averaged over 10 instances) for the computational time (Time), the solution value for Algorithm 1 (Value), the maximum global profit (Optimal Value) and the percentage difference (Diff) between Value and Optimal Value ($100 \times (\text{Value} – \text{Optimal Value}) / \text{Optimal Value}$). The best-known solution value is used when the optimal value is not known. Thus, the percentage difference (Diff) represents the distance relative to the known optimal value or best-known value of a solution.

5.1. Instances with $n=100$ and with $(m, n)=(5, 250)$

Table 1 presents results for Chu and Beasley's instances with $(m, n)=(5,100), (10, 100), (30, 100), and (5, 250). Algorithm 1 produces solutions with very small percentage differences relatively to the known optimal solution values. For the instances in Table 1, the largest percentage difference observed is 0.17% and the largest computational time is 230 sec.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>$\alpha$</th>
<th>Time (s)</th>
<th>Value</th>
<th>Optimal value</th>
<th>Diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100</td>
<td>0.25</td>
<td>4.33</td>
<td>24194.3</td>
<td>24197.2</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>0.50</td>
<td>3.85</td>
<td>43224.4</td>
<td>43252.9</td>
<td>0.07</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>0.75</td>
<td>2.63</td>
<td>60455.2</td>
<td>60471</td>
<td>0.03</td>
</tr>
</tbody>
</table>
5.2. The larger Chu and Beasley's instances

In this subsection we are testing instances with \( n \in \{250, 500\} \), then \( T << n \), and the solution of \( ILP_k \) dominates the computational time. Therefore, in what follows, the integer program \( ILP_k \) is solved with an imposed time limit of 60 sec.

The instances in Table 2 have known optimal solutions and also for these instances Algorithm 1 produces solutions with very small percentage differences. It can be seen that the largest percentage difference attained is 0.08\% and the largest computational time reaches 394 sec. The heuristics of Wilbaut and Hanafi (2009) applied to instances with \((m, n) = (5, 500)\), require computational times, on average, of about 2600 and 3000 seconds on a 3.4 GHz Pentium IV. The heuristic of Vasquez and Vimont (2005) requires an average computational time of nearly 50 hours for each instance on a Pentium IV 2 GHz. For instances with \((m, n) = (10, 500)\) Wilbaut and Hanafi (2009) mention computational times of about 730, 4500 and 4800 seconds for their different heuristics, while Vasquez and Vimont (2005) reported 70 hours.

### Table 1 – Results for each different benchmark instance structures (averaged over 10 instances)

<table>
<thead>
<tr>
<th>(m)</th>
<th>(n)</th>
<th>(\alpha)</th>
<th>Time (s)</th>
<th>Value</th>
<th>Optimal value</th>
<th>Diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>250</td>
<td>0.25</td>
<td>234.67</td>
<td>58974.5</td>
<td>59021.6</td>
<td>0.08</td>
</tr>
<tr>
<td>10</td>
<td>250</td>
<td>0.50</td>
<td>225.24</td>
<td>108711.1</td>
<td>108729.3</td>
<td>0.02</td>
</tr>
<tr>
<td>10</td>
<td>250</td>
<td>0.75</td>
<td>181.93</td>
<td>151327.6</td>
<td>151346.2</td>
<td>0.01</td>
</tr>
</tbody>
</table>
The most difficult Chu and Beasley's instances are those with \((m, n) = (30, 250)\) and with \((m, n) = (30, 500)\). Results for these instances are presented in Table 3. Algorithm 1 produces solutions with very small percentage differences relatively to best known solution values. It can be seen that the largest computational time reaches 653 sec. and the largest percentage difference observed is 0.09%. The results obtained by Wilbaut and Hanafi (2009) for instances with 500 variables and 30 constraints mention computational times of about 3600, 4600 and 5200 seconds, for their different heuristics and the heuristic of Vasquez and Vimont (2005) required 100 hours.

<table>
<thead>
<tr>
<th>(m)</th>
<th>(n)</th>
<th>(\alpha)</th>
<th>Time (s)</th>
<th>Value</th>
<th>Best-Known value</th>
<th>Diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>250</td>
<td>0.25</td>
<td>358.65</td>
<td>56907.3</td>
<td>56933.7</td>
<td>0.05</td>
</tr>
<tr>
<td>30</td>
<td>250</td>
<td>0.50</td>
<td>367.01</td>
<td>106687.2</td>
<td>106719.0</td>
<td>0.03</td>
</tr>
<tr>
<td>30</td>
<td>250</td>
<td>0.75</td>
<td>332.58</td>
<td>150446.2</td>
<td>150485.0</td>
<td>0.03</td>
</tr>
<tr>
<td>30</td>
<td>500</td>
<td>0.25</td>
<td>653.46</td>
<td>115520.5</td>
<td>115623.7</td>
<td>0.09</td>
</tr>
<tr>
<td>30</td>
<td>500</td>
<td>0.50</td>
<td>630.14</td>
<td>216197.4</td>
<td>216274.7</td>
<td>0.04</td>
</tr>
<tr>
<td>30</td>
<td>500</td>
<td>0.75</td>
<td>623.91</td>
<td>302395.4</td>
<td>302446.5</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Table 3 – Results for each different benchmark instance structures (averaged over 10 instances)
6. Conclusions

The computational tests showed that, on average, the method investigated here can quickly produce solutions with values very close to the optimal values or, when optimal values are not known, the values produced are very close to best known values. For the hardest instances in a classical benchmark, the method required, on average, about 600 sec on a 1.73GHz PC to produce high quality solutions.

Further studies may address the theoretical characterization of the method's behaviour.
References


