Ant algorithm for the multidimensional knapsack problem
Ines Alaya, Christine Solnon, Khaled Ghedira

To cite this version:

HAL Id: hal-01541529
https://hal.science/hal-01541529
Submitted on 24 Mar 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
ANT ALGORITHM FOR THE MULTIDIMENSIONAL KNAPSACK PROBLEM

Inès Alaya  
SOIE, Institut Supérieur de Gestion de Tunis  
41 Rue de la Liberté- Cité Bouchoucha-2000 Le Bardo-Tunis  
ines.alaya@isg.rnu.tn

Christine Solnon  
LIRIS, CNRS FRE 2672, University Lyon 1  
43 bd du 11 novembre, 69622 Villeurbanne cedex  
christine.solnon@liris.cnrs.fr

Khaled Ghédira  
SOIE, Institut Supérieur de Gestion de Tunis  
41 Rue de la Liberté- Cité Bouchoucha-2000 Le Bardo-Tunis  
khaled.ghedira@isg.rnu.tn

Abstract  We propose a new algorithm based on the Ant Colony Optimization (ACO) meta-heuristic for the Multidimensional Knapsack Problem, the goal of which is to find a subset of objects that maximizes a given objective function while satisfying some resource constraints. We show that our new algorithm obtains better results than two other ACO algorithms on most instances.

Keywords:  Ant Colony Optimization, Multidimensional Knapsack Problem

1. Introduction  

The Multidimensional Knapsack Problem (MKP) is a NP-hard problem which has many practical applications, such as processor allocation in distributed systems, cargo loading, or capital budgeting. The goal of the MKP is to find a subset of objects that maximizes the total profit while satisfying some resource constraints. More formally, a MKP is
stated as follows:

\[
\begin{align*}
\text{maximize} & \quad \sum_{j=1}^{n} p_j \cdot x_j \\
\text{subject to} & \quad \sum_{j=1}^{n} r_{ij} \cdot x_j \leq b_i, \forall i \in 1..m \\
& \quad x_j \in \{0, 1\}, \forall j \in 1..n
\end{align*}
\]

where \( r_{ij} \) is the consumption of resource \( i \) for object \( j \), \( b_i \) is the available quantity of resource \( i \), \( p_j \) is the profit associated with object \( j \), and \( x_j \) is the decision variable associated with object \( j \) and is set to 1 (resp. 0) if \( j \) is selected (resp. not selected).

In this paper, we describe a new algorithm for solving MKPs. This algorithm is based on Ant Colony Optimization (ACO) [4], a stochastic metaheuristic that has been applied to solve many combinatorial optimization problems such as traveling salesman problems [3], quadratic assignment problems [6], or vehicle routing problems [1]. The basic idea of ACO is to model the problem to solve as the search for a minimum cost path in a graph, and to use artificial ants to search for good paths. The behavior of artificial ants is inspired from real ants: they lay pheromone trails on components of the graph and they choose their path with respect to probabilities that depend on pheromone trails that have been previously laid; these pheromone trails progressively decrease by evaporation. Intuitively, this indirect stigmergetic communication means giving information about the quality of path components in order to attract ants, in the following iterations, towards the corresponding areas of the search space.

To solve MKPs with ACO, the key point is to decide which components of the constructed solutions should be rewarded, and how to exploit these rewards when constructing new solutions. A solution of a MKP is a set of selected objects \( S = \{o_1, \ldots, o_k\} \) (we shall say that an object \( o_i \) is selected if the corresponding decision variable \( x_{o_i} \) has been set to 1). Given such a solution \( S = \{o_1, \ldots, o_k\} \), one can consider three different ways of laying pheromone trails:

- A first possibility is to lay pheromone trails on each object selected in \( S \). In this case, the idea is to increase the desirability of each object of \( S \) so that, when constructing a new solution, these objects will be more likely to be selected;

- A second possibility is to lay pheromone trails on each couple \( (o_i, o_{i+1}) \) of successively selected objects of \( S \). In this case, the idea is to increase the desirability of choosing object \( o_{i+1} \) when the last selected object is \( o_i \).

- A third possibility is to lay pheromone on all pairs \( (o_i, o_j) \) of different objects of \( S \). In this case, the idea is to increase the desirability
Algorithm Ant-knapsack:

Initialize pheromone trails to \( \tau_{\text{max}} \)

repeat the following cycle:

for each ant \( k \) in 1..\( \text{nbAnts} \), construct a solution \( S_k \) as follows:

Randomly choose a first object \( o_1 \in 1..n \)

\[ S_k \leftarrow \{ o_1 \} \]

Candidates \( \leftarrow \{ o_i \in 1..n / o_i \text{ can be selected without violating resource constraints} \} \)

while Candidates \( \neq \emptyset \) do

Choose an object \( o_i \in \text{Candidates} \) with probability \( p_{S_k}(o_i) \)

\[ S_k \leftarrow S_k \cup \{ o_i \} \]

remove from Candidates every object that violates some resource constraints

end while

end for

Update pheromone trails w.r.t. \( \{ S_1, \ldots, S_{\text{nbAnts}} \} \)

if a pheromone trail is lower than \( \tau_{\text{min}} \) then set it to \( \tau_{\text{min}} \)

if a pheromone trail is greater than \( \tau_{\text{max}} \) then set it to \( \tau_{\text{max}} \)

until maximum number of cycles reached or optimal solution found

Figure 1. ACO algorithm for solving MKPs

of choosing together two objects of \( S \) so that, when constructing a new solution \( S' \), the objects of \( S \) will be more likely to be selected if \( S' \) already contains some objects of \( S \). More precisely, the more \( S' \) will contain objects of \( S \), the more the other objects of \( S \) will be attractive.

To solve MKP with ACO, Leguizamon and Michalewicz [7] have proposed an algorithm based on the first possibility, whereas Fidanova [5] has proposed another algorithm based on the second possibility. In this paper, we propose a new ACO algorithm for solving MKPs that is based on the third possibility. Our intuition is that this strategy should attract ants in a more precise way as the desirability of an object depends on the objects that already belong to the partial solution under construction.

2. Ant-knapsack description

We define the construction graph, on which ants lay pheromone trails, as a complete graph that associates a node to each object of the MKP. The quantity of pheromone laying on an edge \((o_i, o_j)\) is denoted by \( \tau(o_i, o_j) \). Intuitively, this quantity represents the learnt desirability of selecting together objects \( o_i \) and \( o_j \).
The proposed ACO algorithm for solving MKPs is described in figure 1 and more particularly follows the MAX - MIN Ant System [8]: we explicitly impose lower and upper bounds $\tau_{min}$ and $\tau_{max}$ on pheromone trails (with $0 < \tau_{min} < \tau_{max}$), and pheromone trails are set to $\tau_{max}$ at the beginning of the search.

At each cycle of this algorithm, every ant constructs a solution. It first randomly chooses an initial object, and then iteratively adds objects that are chosen within a set Candidates that contains all the objects that can be selected without violating resource constraints. Once each ant has constructed a solution, pheromone trails are updated. The algorithm stops either when an ant has found an optimal solution (when the optimal bound is known), or when a maximum number of cycles has been performed.

2.1 Definition of transition probabilities

At each step of the construction of a solution, an ant $k$ randomly selects the next object $o_i$ within the set Candidates with respect to a probability $p_{S_k}(o_i)$. This probability is defined proportionally to a pheromone factor and a heuristic factor, i.e.,

$$p_{S_k}(o_i) = \frac{[\tau_{S_k}(o_i)]^\alpha [\eta_{S_k}(o_i)]^\beta}{\sum_{o_j \in \text{Candidates}} [\tau_{S_k}(o_j)]^\alpha [\eta_{S_k}(o_j)]^\beta}$$

where $\tau_{S_k}(o_i)$ is the pheromone factor of $o_i$, $\eta_{S_k}(o_i)$ is its heuristic factor, and $\alpha$ and $\beta$ are two parameters that determine the relative importance of these two factors.

The pheromone factor $\tau_{S_k}(o_i)$ depends on the quantity of pheromone laid on edges connecting the objects that already are in the partial solution $S_k$ and the candidate node $o_i$, i.e.,

$$\tau_{S_k}(o_i) = \sum_{o_j \in S_k} \tau(o_i, o_j)$$

Note that this pheromone factor can be computed in an incremental way: once the first object $o_i$ has been randomly chosen, for each candidate object $o_j$, the pheromone factor $\tau_{S_k}(o_j)$ is initialized to $\tau(o_i, o_j)$; then, each time a new object $o_l$ is added to the solution $S_k$, for each candidate object $o_j$, the pheromone factor $\tau_{S_k}(o_j)$ is incremented by $\tau(o_l, o_j)$.

The heuristic factor $\eta_{S_k}(o_i)$ also depends on the whole set $S_k$ of selected objects. Let $c_{S_k}(i) = \sum_{g \in S_k} r_{ig}$ be the consumed quantity of the resource $i$ when the ant $k$ has selected the set of objects $S_k$. And let $d_{S_k}(i) = b_i - c_{S_k}(i)$ be the remaining capacity of the resource $i$. We
define the following ratio:

\[ h_{S_k}(j) = \sum_{i=1}^{m} \frac{r_{ij}}{d_{S_k}(i)} \]

which represents the tightness of the object \( j \) on the constraints \( i \) relatively to the constructed solution \( S_k \). Thus, the lower this ratio is, the more the object is profitable.

We integrate the profit of the object in this ratio to obtain a pseudo-utility factor. We can now define the heuristic factor formula as follows:

\[ \eta_{S_k}(j) = \frac{p_j}{h_{S_k}(j)} \]

### 2.2 Pheromone updating

Once each ant has constructed a solution, pheromone trails laying on the construction graph edges are updated according to the ACO meta-heuristic. First, all amounts are decreased in order to simulate evaporation. This is done by multiplying the quantity of pheromone laying on each edge of the construction graph by a pheromone persistence rate \( 1 - \rho \) such that \( 0 \leq \rho \leq 1 \).

Then, the best ant of the cycle deposits pheromone. More precisely, let \( S_k \in \{ S_1, \ldots, S_{nbAnts} \} \) be the best solution (with maximal profit) constructed during the cycle, and \( S_{best} \) be the best solution built since the beginning of the run. The quantity of pheromone laid by ant \( k \) is inversely proportional to the gap of profit between \( S_k \) and \( S_{best} \), i.e., it is equal to \( 1/(1 + \text{profit}(S_{best}) - \text{profit}(S_k)) \). This quantity of pheromone is added on each edge connecting two different vertices of \( S_k \).

### 3. Parameters setting

When solving a combinatorial optimization problem with a heuristic approach such as evolutionary computation or ACO, one usually has to find a compromise between two dual goals. On one hand, one has to intensify the search around the most “promising” areas, that are usually close to the best solutions found so far. On the other hand, one has to diversify the search and favor exploration in order to discover new, and hopefully more successful, areas of the search space. The behavior of ants with respect to this intensification/diversification duality can be influenced by modifying parameter values. In particular, diversification can be emphasized either by decreasing the value of the pheromone factor weight \( \alpha \) — so that ants become less sensitive to pheromone trails — or by decreasing the value of the pheromone evaporation rate \( \rho \) — so that pheromone evaporates more slowly. When increasing the exploratory
Figure 2. Influence of $\alpha$ and $\rho$ on solution quality: each curve plots the evolution of the profit of the best solution when the number of cycles increases, for a given setting of $\alpha$ and $\rho$. The other parameters have been set to $\beta = 5$, nbAnts = 30, $\tau_{\text{min}} = 0.01$, and $\tau_{\text{max}} = 6$.

ability of ants in this way, one usually finds better solutions, but as a counterpart it takes longer time to find them.

This is illustrated in figure 2 on a MKP instance with 100 objects and 5 resource constraints. When emphasizing pheromone guidance, by choosing values such as $\alpha = 2$ and $\rho = 0.02$, Ant-knapsack quickly finds good solutions but it may fail in finding the optimal (or the best) solution. On the contrary, when choosing values for $\alpha$ and $\rho$ that emphasize exploration, such as $\alpha = 1$ and $\rho = 0.01$, ants find better solutions, but they need more cycles to converge towards these solutions. A good compromise between solution quality and computation time is reached when $\alpha$ is set to 1 and $\rho$ to 0.01.

For all experiments reported below, we have set $\alpha$ to 1, $\beta$ to 5, $\rho$ to 0.01, the number of ants nbAnts to 30, and the pheromone bounds $\tau_{\text{min}}$ and $\tau_{\text{max}}$ to 0.01 and 6. Finally, we limited the number of cycles to 2000.
4. Experiments and results

The Ant-knapsack has been tested on benchmarks of MKP from OR-Library. We compare the results of Ant-knapsack with the two ACO algorithms of Leguizamon and Michalewicz [7] and Fidanova [5], and the genetic algorithm of Chu and Beasley [2].

Table 1. Results on 5,100 instances. For each instance, the table reports the best solutions found by Chu and Beasley as reported in [2](C. & B.), the best and average solutions found by Leguizamon and Michalewicz as reported in [7](L. & M.), and the best solutions found by Fidanova as reported in [5]. It then reports results obtained by Ant-knapsack: best and average solutions over 50 runs, followed by standard deviation in brackets, and the average number of cycles needed to find the best solution (C*).
Table 2. Results on 10.100 instances. For each instance, the table reports the best solutions found by Chu and Beasley as reported in [2](C. & B), and by Leguizamon and Michalewicz as reported in [7](L. & M). It then reports results obtained by Ant-knapsack: best and average solutions over 50 runs, followed by standard deviation in brackets, and the average number of cycles needed to find the best solution (C*).

<table>
<thead>
<tr>
<th>N°</th>
<th>C. &amp; B Best</th>
<th>L. &amp; M. Best</th>
<th>Ant-knapsack Best</th>
<th>Ant-knapsack Avg (sdv)</th>
<th>C*</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>23064</td>
<td>23057</td>
<td>23064</td>
<td>23016 (42.2)</td>
<td>538</td>
</tr>
<tr>
<td>01</td>
<td>22801</td>
<td>22801</td>
<td>22801</td>
<td>22714 (67.2)</td>
<td>575</td>
</tr>
<tr>
<td>02</td>
<td>22131</td>
<td>22131</td>
<td>22131</td>
<td>22034 (66.9)</td>
<td>598</td>
</tr>
<tr>
<td>03</td>
<td>22772</td>
<td>22772</td>
<td>22772</td>
<td>22634 (60.6)</td>
<td>700</td>
</tr>
<tr>
<td>04</td>
<td>22751</td>
<td>22654</td>
<td>22654</td>
<td>22547 (66.3)</td>
<td>640</td>
</tr>
<tr>
<td>05</td>
<td>22777</td>
<td>22652</td>
<td>22652</td>
<td>22602 (63.3)</td>
<td>645</td>
</tr>
<tr>
<td>06</td>
<td>21875</td>
<td>21875</td>
<td>21875</td>
<td>21777 (44.9)</td>
<td>552</td>
</tr>
<tr>
<td>07</td>
<td>22635</td>
<td>22551</td>
<td>22551</td>
<td>22453 (89.2)</td>
<td>586</td>
</tr>
<tr>
<td>08</td>
<td>22511</td>
<td>22418</td>
<td>22418</td>
<td>22351 (69.4)</td>
<td>534</td>
</tr>
<tr>
<td>09</td>
<td>22702</td>
<td>22702</td>
<td>22702</td>
<td>22591 (88.5)</td>
<td>588</td>
</tr>
<tr>
<td>10</td>
<td>41395</td>
<td>41395</td>
<td>41395</td>
<td>41329 (48.5)</td>
<td>501</td>
</tr>
<tr>
<td>11</td>
<td>42344</td>
<td>42344</td>
<td>42344</td>
<td>42214 (49.5)</td>
<td>559</td>
</tr>
<tr>
<td>12</td>
<td>42401</td>
<td>42401</td>
<td>42401</td>
<td>42300 (58.1)</td>
<td>584</td>
</tr>
<tr>
<td>13</td>
<td>45624</td>
<td>45624</td>
<td>45624</td>
<td>45461 (73.6)</td>
<td>562</td>
</tr>
<tr>
<td>14</td>
<td>41884</td>
<td>41884</td>
<td>41884</td>
<td>41739 (57.3)</td>
<td>536</td>
</tr>
<tr>
<td>15</td>
<td>42995</td>
<td>42995</td>
<td>42995</td>
<td>42909 (76.3)</td>
<td>525</td>
</tr>
<tr>
<td>16</td>
<td>43559</td>
<td>43559</td>
<td>43559</td>
<td>43464 (71.7)</td>
<td>597</td>
</tr>
<tr>
<td>17</td>
<td>42970</td>
<td>42970</td>
<td>42970</td>
<td>42903 (47.7)</td>
<td>439</td>
</tr>
<tr>
<td>18</td>
<td>42212</td>
<td>42212</td>
<td>42212</td>
<td>42146 (48.0)</td>
<td>598</td>
</tr>
<tr>
<td>19</td>
<td>41207</td>
<td>41207</td>
<td>41207</td>
<td>41067 (89.7)</td>
<td>548</td>
</tr>
<tr>
<td>20</td>
<td>57375</td>
<td>57375</td>
<td>57375</td>
<td>57318 (59.5)</td>
<td>330</td>
</tr>
<tr>
<td>21</td>
<td>58978</td>
<td>58978</td>
<td>58978</td>
<td>58889 (40.2)</td>
<td>504</td>
</tr>
<tr>
<td>22</td>
<td>58391</td>
<td>58391</td>
<td>58391</td>
<td>58129 (29.5)</td>
<td>513</td>
</tr>
<tr>
<td>23</td>
<td>61966</td>
<td>61966</td>
<td>61966</td>
<td>61885 (42.4)</td>
<td>427</td>
</tr>
<tr>
<td>24</td>
<td>60803</td>
<td>60803</td>
<td>60803</td>
<td>60798 (5.0)</td>
<td>316</td>
</tr>
<tr>
<td>25</td>
<td>61437</td>
<td>61437</td>
<td>61437</td>
<td>61293 (52.7)</td>
<td>502</td>
</tr>
<tr>
<td>26</td>
<td>56377</td>
<td>56377</td>
<td>56377</td>
<td>56324 (35.7)</td>
<td>453</td>
</tr>
<tr>
<td>27</td>
<td>59391</td>
<td>59391</td>
<td>59391</td>
<td>59339 (53.3)</td>
<td>445</td>
</tr>
<tr>
<td>28</td>
<td>60205</td>
<td>60205</td>
<td>60205</td>
<td>60146 (62.6)</td>
<td>360</td>
</tr>
<tr>
<td>29</td>
<td>60633</td>
<td>60633</td>
<td>60633</td>
<td>60605 (36.1)</td>
<td>360</td>
</tr>
</tbody>
</table>
Ant algorithm for the multidimensional knapsack problem

Table 3. Results on 5,500 instances. For each instance, the table reports the best solutions found by Vasquez and Hao as reported in [9](V. & H). It then reports results obtained by Ant-knapsack: best and average solutions over 50 runs, followed by standard deviation in brackets, and the average number of cycles needed to find the best solution (C*).

<table>
<thead>
<tr>
<th>N°</th>
<th>V. &amp; H.</th>
<th>Ant-knapsack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Best</td>
</tr>
<tr>
<td>00</td>
<td></td>
<td>120134</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>117864</td>
</tr>
<tr>
<td>02</td>
<td></td>
<td>121112</td>
</tr>
<tr>
<td>03</td>
<td></td>
<td>120804</td>
</tr>
<tr>
<td>04</td>
<td></td>
<td>122319</td>
</tr>
</tbody>
</table>

Table 1 displays the results for 30 instances with 100 objects and 5 constraints (n=100 and m=5). On these instances, Ant-knapsack clearly outperforms Fidanova’s algorithm. It also obtains better results than the algorithm of Leguizamon and Michalewicz: the best solutions found are always larger or equal, and the average solutions found are larger for 7 instances, and smaller for 3 instances. Ant-knapsack finds the best known results of Chu and Beasley for 26 instances over the 30 tested instances.

Table 2 displays the results for 30 instances with 100 objects and 10 constraints (n=100 and m=10). On these instances, Ant-knapsack also obtains better results than the algorithm of Leguizamon and Michalewicz: the best solutions found are larger or equal for 9 instances over 10, and the average solutions found are larger for 8 instances, and smaller for 2 instances. Ant-knapsack finds for this set also the best known results of Chu and Beasley for 25 instances over 30.

We also tested Ant-knapsack on larger MKP instances with 500 objects and 5 constraints (table 3). The best known results for this set are obtained by Vasquez and Hao [9]. They proposed an hybrid algorithm that combines tabu search and linear programming. On these difficult instances, we find worse results than those of Vasquez and Hao.

5. Conclusion

In this paper, we propose an ACO algorithm for the multidimensional knapsack problem. This algorithm differs from many ACO algorithms in the fact that pheromone trails are laid not only on the edges of the visited paths, but on all edges connecting any pair of nodes belonging to the solution. In addition, when adding a node to the solution under construction, the probability of choosing a node not only depends on the
pheromone trail between the last visited node and the candidate node but on the trails laying on all edges connecting the candidate node and all visited nodes in the solution. The proposed algorithm finds most of the best known results for the tested MKP benchmarks. This algorithm improves also many results found by other ACO algorithms.

Notes

1. available at http://mscmga.ms.ic.ac.uk/.

References


