Applying the Greedy Algorithm for Reducing the Dimensionality of the Dynamic Programming Method in Solving the One-Dimensional Cutting Stock Problem

Rustam Abubakirovich Fayzrakhmanov, Rustam Talgatovich Murzakaev, Aleksey Sergeevich Mezentsev and Vadim Sergeevich Shilov

Perm National Research Polytechnic University, Perm, Russia

Abstract: The article proposes a modification of the greedy algorithm for reducing the dimensionality of the dynamic programming method in one-dimensional material cutting. This modification helps reduce the complexity of the dynamic programming method from exponential to pseudopolynomial, which substantially reduces the time needed for solving the dynamic programming problem for one-dimensional cutting. The algorithm proposed in this article helps find better solutions than the ones existing at the moment. The article furnishes examples substantiating these conclusions.

Key words: Algorithm, One-dimensional material, Cutting, Complexity, Dynamic programming, Greedy algorithms, NP-full problem

INTRODUCTION

Trends towards the reduction of technological production leftovers necessitate problems on efficient material cutting [1, 2, 3, 4]. Products that undergo cutting vary in dimensionality:

- One-dimensional – wires, rods and other similar materials;
- Two-dimensional – sheets of various thickness and other similar materials;
- Three-dimensional – spatial parts and other similar materials.

The trouble is that any cutting problem belongs to the class of NP-full problems, which means there are no algorithms capable of finding accurate solutions to the problem within polynomial time. Most algorithms search one of the acceptable solutions within finite (set beforehand) time. As a result, virtually all newer cutting algorithms strive to improve the quality of cutting compared to algorithms developed earlier.

Description of Earlier Algorithms: To demonstrate the differences between the algorithm proposed and other methods for seeking solutions to the one-dimensional cutting problem, we shall take a look at three heuristic algorithms [5, 6].

The Next Fit algorithm (NF): The items are packed in arbitrary order by the following rule:

1. The first item is placed in the first bin. A bin is an abstract representation of an object of an arbitrary nature, which under the statement of the problem is subject to cutting or into which items are packed.
2. At the k-th step, an attempt is made to place the k-th item in the current bin.
3. If the item fits in, it is placed inside and we move on to the next step – otherwise, the item is placed in a new bin.

This algorithm has complexity $O(n)$, where $n$ is the amount of parts.

The First Fit algorithm (FF): The items are packed in arbitrary order by the following rule:
The first item is placed inside the bin.

At the $k$-th step, we look for a bin with the lowest number, wherein the $k$-th item is placed.

If there is no such bin available, we use a new empty bin and put the item in it.

This algorithm has complexity $O(n')$.

**The Best Fit algorithm (BF):** The items are packed in arbitrary order by the following rule:

- The first item is placed in the bin.
- At the $k$-th step, the $k$-th item is placed by the rule: we search for partially filled bins, where there is enough room for the item and pick among them one filled the most. If there are no such bins available, we pick a new empty bin and place the $k$-th item in it.

This algorithm has complexity $O(n')$.

**Combined Use of the Algorithms:** The algorithms Next Fit (NF), First Fit (FF) and Best Fit can be used both independently and in combination with other algorithms – for instance, with metaheuristics [7].

The two most commonly used are two metaheuristics:

- Genetic algorithms [8]. These have different crossing-over and mutation procedures, algorithm operation parameters;
- The simulated annealing method [8].

Combining the metaheuristics with the algorithms (NF, FF, BF) helps enhance solutions obtained by the algorithms. However, this increases the overall time required for looking for a solution.

**Dynamic Programming + the Greedy Algorithm:** For solving the one-dimensional cutting problem, the authors propose a modified algorithm based on combined use of the greedy algorithm and dynamic programming.

The one-dimensional cutting stock problem resembles the problem about a rucksack with several rucksacks [9, 10]. The rucksack in one-dimensional cutting stock problem is analogous to the notion of a feedstock. In practice, it’s impossible to apply dynamic programming to this problem, since its dimensionality, which will equal the number of rucksacks plus 1, will be too large, which leads to the problem’s exponential complexity.

For reducing dimensionality in dynamic programming, the authors propose using the ideas of greedy algorithms.

Let us examine the problem in a simplified formulation. Let there be one feedstock; we need to cut it into parts in such a way as to minimize the waste; we do not need the feedstock to accommodate all the parts. We can also assume that all the parts have an integer length. If the starting length of some parts is not integer, we can multiply all the lengths by some constant for the length to be integer.

This problem is similar to the rucksack problem [10], which employs pseudopolynomial algorithms using dynamic programming with complexity $O(nW)$, where $n$ is the number of parts and $W$ is the length of the feedstock subject to cutting.

The selection of a group of objects is done using the following algorithm. Let the feedstock have length $L$ and the amount of parts not yet engaged in cutting be $m$. Our goal is to determine the maximum length of the feedstock we can use. Let the problem have the following parameters: $l$ – the current used length of the feedstock, $k$ – the amount of parts considered. Then the logic (Boolean) function $f(l, k)$ characterizes the attainability of the state, which can be determined from a recurrent ratio (1).

$$f(b, k) = \begin{cases} 
    \text{False}, & \text{if } h < 0, \\
    \text{False}, & \text{if } k \neq 0 \land k = 0, \\
    \text{True}, & \text{if } h = 0, \\
    f(h - (k), k - 1) \lor f(h - w[k], k - 1) \lor f(h, k - 1), & k > 0 \land h > 0.
\end{cases} \quad (1)$$

Thus, we apply the dynamic programming method to each feedstock.

Performing the above procedure until all parts are accommodated enables us to find a rational (acceptable) solution – thus, we implement the greedy algorithm. The greed of the algorithm lies in that at each step it looks for an optimum solution for that step but not the entire problem at large.

Using the greedy algorithm reduces the complexity of the overall problem in dynamic programming and thereby diminishes the increase in resources needed for solving the problem with increase in dimensionality.

It is apparent that although this approach will give us optimum solutions when it comes to sub-problems, at large the solution can turn out to be non-optimum. For improving the solution, we suggest, in restoring a set
of parts (reverse looping), taking parts of the largest length, above all. Although this procedure will enhance the solution found, we still cannot guarantee that it is going to be optimum.

Comparing the Algorithms’ Performance: The algorithms examined in the article were implemented in the high-level programming language C++. In the course of the algorithm’s operation, we compute the cutting and calculate the number of feedstocks needed for it.

In analyzing the algorithms, the following two characteristics were taken into account:

- The number of feedstocks that were used in cutting;
- Time spent on obtaining the solution.

The smaller any of these two characteristics, the better the cutting algorithm. Our priority here is the number of feedstocks used in cutting, whereas the time spent on getting the solution plays the limiting role and is not as important.

To compare the algorithms, we used five datasets differing in the lengths of parts and feedstocks, as well as their quantity.

Dataset 1 has the following characteristics:

- The size of the feedstocks: 1000 mm;
- The size of the parts: from 45 to 124 mm;
- The number of the parts: 100 pcs.

Table 1 contains the full list of all parts subject to cutting from Dataset 1. Table 2 lists the characteristics of solutions obtained for each algorithm.

For Dataset 1, we use commensurate times for the algorithms’ operation.

Dataset 2 has the following characteristics:

- The size of the feedstocks: 150 mm;
- The size of the parts: from 12 to 43 mm;
- The number of the parts: 44 PCs.

Table 3 contains the full list of all the parts subject to cutting from Dataset 2. Table 4 lists the characteristics of solutions obtained for each algorithm.

For Dataset 2, we use commensurate times for the algorithms’ operation.

Figure 1 visually demonstrates the comparison of solutions for Dataset 2 in the form of cutting maps obtained using dynamic programming with the greedy algorithm (Figure 1, a) and the Best Fit algorithm (Figure 1, b). These algorithms performed the best in relation to Dataset 2. The parts accommodated by the feedstocks are shown in green. The technological waste is shown in red.

We are not providing tables with data on the lengths and amount of parts for Datasets 3, 4 and 5 due to their large size.

Dataset 3 has the following characteristics:

- The size of the feedstocks: 1000 mm;
- The size of the parts: from 40 to 124 mm;
- The average length is 81.52;
- The dispersion of the parts’ lengths is 501.91;
- The mean-square deviation is 22.40;
- The number of the parts: 1005 pcs.

Table 5 lists the characteristics of solutions obtained for each algorithm.

Dataset 4 has the following characteristics:

- The size of the feedstocks: 1000 mm;
- The size of the parts: from 100 to 499 mm;
- The average length is 300.43;
- The dispersion of the parts’ lengths is 13152.51;
- The mean-square deviation is 1217.72;
- The number of the parts: 1005 pcs.

Table 6 lists the characteristics of solutions obtained for each algorithm. Figure 2 features a graph constructed based on Table 6, which characterizes the number of used feedstocks depending on the algorithm.

Dataset 5 has the following characteristics:

- The size of the feedstocks: 15000 mm;
- The size of the parts: from 604 to 1868 mm;
- The average length is 1217.72;
- The dispersion of the parts’ lengths is 156177.76;
- The mean-square deviation is 395.19;
- The number of the parts: 1003 PCs.

Table 7 lists the characteristics of solutions obtained for each algorithm.
Table 1: Parts in Dataset 1

<table>
<thead>
<tr>
<th>No</th>
<th>Length of parts</th>
<th>Quantity</th>
<th>#</th>
<th>Length of parts</th>
<th>Quantity</th>
<th>#</th>
<th>Length of parts</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45</td>
<td>9</td>
<td>7</td>
<td>82</td>
<td>6</td>
<td>13</td>
<td>108</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>6</td>
<td>8</td>
<td>91</td>
<td>8</td>
<td>14</td>
<td>109</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>56</td>
<td>2</td>
<td>9</td>
<td>97</td>
<td>3</td>
<td>15</td>
<td>121</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>62</td>
<td>12</td>
<td>10</td>
<td>103</td>
<td>2</td>
<td>16</td>
<td>122</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>73</td>
<td>9</td>
<td>11</td>
<td>105</td>
<td>5</td>
<td>17</td>
<td>124</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>74</td>
<td>8</td>
<td>12</td>
<td>106</td>
<td>2</td>
<td></td>
<td>Total amount of parts: 100</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Results of the algorithms’ operation on Dataset 1

<table>
<thead>
<tr>
<th>Algorithm + greedy algorithm</th>
<th>Dynamic programming</th>
<th>NF</th>
<th>FF</th>
<th>BF</th>
<th>NF + simulated annealing method</th>
<th>FF + simulated annealing method</th>
<th>BF + simulated annealing method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of feedstocks, pcs.</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 3: Parts in Dataset 2

<table>
<thead>
<tr>
<th>No</th>
<th>Length of parts</th>
<th>Quantity</th>
<th>#</th>
<th>Length of parts</th>
<th>Quantity</th>
<th>#</th>
<th>Length of parts</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>3</td>
<td>4</td>
<td>18</td>
<td>6</td>
<td>7</td>
<td>32</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>8</td>
<td>5</td>
<td>21</td>
<td>5</td>
<td>8</td>
<td>43</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>1</td>
<td>6</td>
<td>23</td>
<td>10</td>
<td></td>
<td>Total amount of parts: 44</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Results of the algorithms’ operation on Dataset 2

<table>
<thead>
<tr>
<th>Algorithm + greedy algorithm</th>
<th>Dynamic programming</th>
<th>NF</th>
<th>FF</th>
<th>BF</th>
<th>NF + simulated annealing method</th>
<th>FF + simulated annealing method</th>
<th>BF + simulated annealing method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of feedstocks, pcs.</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 5: Results of the algorithms’ operation on Dataset 3

<table>
<thead>
<tr>
<th>Algorithm + greedy algorithm</th>
<th>Dynamic programming</th>
<th>NF</th>
<th>FF</th>
<th>BF</th>
<th>NF + simulated annealing method</th>
<th>FF + simulated annealing method</th>
<th>BF + simulated annealing method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of feedstocks, pcs.</td>
<td>82</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>85</td>
<td>85</td>
</tr>
</tbody>
</table>

Table 6: Results of the algorithms’ operation on Dataset 4

<table>
<thead>
<tr>
<th>Algorithm + greedy algorithm</th>
<th>Dynamic programming</th>
<th>NF</th>
<th>FF</th>
<th>BF</th>
<th>NF + simulated annealing method</th>
<th>FF + simulated annealing method</th>
<th>BF + simulated annealing method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of feedstocks, pcs.</td>
<td>303</td>
<td>353</td>
<td>349</td>
<td>349</td>
<td>356</td>
<td>347</td>
<td>313</td>
</tr>
</tbody>
</table>

Table 7: Results of the algorithms’ operation on Dataset 5

<table>
<thead>
<tr>
<th>Algorithm + greedy algorithm</th>
<th>Dynamic programming</th>
<th>NF</th>
<th>FF</th>
<th>BF</th>
<th>NF + simulated annealing method</th>
<th>FF + simulated annealing method</th>
<th>BF + simulated annealing method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of feedstocks, pcs.</td>
<td>82</td>
<td>85</td>
<td>85</td>
<td>85</td>
<td>85</td>
<td>85</td>
<td>83</td>
</tr>
</tbody>
</table>

Fig. 1: Results of the algorithms’ operation on Dataset 2 (a – dynamic programming with the greedy algorithm; b – the Best Fit algorithm)
CONCLUSION

The modification examined enables us to lower the complexity of the dynamic programming method from exponential to pseudopolynomial, which substantially reduces the time needed for solving the dynamic programming problem for one-dimensional cutting, putting the one-dimensional cutting stock problem in the category of problems solved within acceptable time limits.

Inferences:

- The proposed algorithm is more efficient than the traditionally employed algorithms, Next Fit, First Fit and Best Fit, when it comes to datasets characterized by a large number of feedstocks subject to cutting. When the amount of parts and feedstocks is low, the proposed algorithm comes up with a solution not worse than the other algorithm.
- With increase in feedstocks’ lengths and the amount of parts subject to cutting, the time of the algorithm’s operation starts to increase in accordance with declared complexity $O(nW)$.

REFERENCES


