A SOFTWARE PROGRAM FOR OPTIMAL 1D CUTTING SUPPORT

Keywords: Linear programming, knapsack problem, optimization, cutting problem, column generation

1. INTRODUCTION

Optimization procedures aim at finding the best solution from a set of feasible solutions [1, 4, 14]. In the simplest case, they minimize the so-called objective function. In 1997, the tercentenary of the modern optimization took place, since in 1697 Johan Bernoulli announced his brachistochrone contest, which boiled down to the problem of finding the curve on a plane that connected A and B points, not lying on a vertical line, that were passed by a material point subject to the gravity force in the minimal time.

Nowadays, optimization problems are ubiquitous in many fields of knowledge, as in mechanics, engineering, economic sciences, automation or logistics. As an example, one can recall a traveling salesman problem or problem of placing a fence of a given length to embrace the maximum area [13].

Among the optimization problems one can find a practical and a very interesting optimal cutting task. Cutting problems were mathematically formulated by Leonid Kantorivich in 1930, a Soviet mathematician. This task was simply a linear programming problem with integer constraints imposed on the decision variables, since the demand on specific lengths of the cut material is always integer.

State-of-the-art solutions (see [2]) to this problem utilize the model of Kantorovich, Column Generation method of Gomory, or One-cut models of Stadler, whereas this paper extends these approaches with a hybrid reformulation of the objective function.

This paper presents the software program for supporting 1D cutting problems of planks that solves and visualises the results. It extends the results presented in [5, 9, 11], from which the description of basic algorithms is taken, and where the minimum cost solution is sought. In this paper, two additional objective functions are proposed and implemented in the program, as well as the graphical user interface for optimization problems is described, with all subfunctions presented in detail.

The basic algorithm presented in the paper is a modified primal simplex method [5] which
Matteusz Pacek and Dariusz Horla solve the following optimization problem:

$$\min_{\mathbf{x}} \quad \mathbf{c}^T \mathbf{x}$$

subject to:

$$\mathbf{A} \mathbf{x} \leq \mathbf{b},$$

$$\mathbf{x} \geq \mathbf{0}.$$

where $\mathbf{x} \in \mathbb{R}^n$ describes decision variables, $\mathbf{c} \in \mathbb{R}^n$ is the price vector, $\mathbf{b} \in \mathbb{R}^m$ denotes the resources vector, usually $m \leq n$, and $\mathbf{A}^{m \times n}$ defines the constraints.

During typical steps of a linear programming (LP) solver, the matrix $\mathbf{A}$ is known prior to performing optimization. In the approach presented in the paper, columns of this matrix are generated shortly after the algorithms start, using 1D auxiliary knapsack problem. To solve it, the dynamic programming solver is used [12, 7, 10].

Of course, freeware or commercial solvers to such problems are available (Astrokettle Algorithms 1D Stock Cutter, i-CUT Suite, Optimuncut software – the 1D Cutting Stock Problem, ÇelikProIV), though they are usually closed, in the sense that they are in the form of compiled libraries, whereas the paper presents the software program written in Matlab language, with all functions given as open text, i.e., not compiled, enabling one to alter the code or to use it for their purpose enabling optimization by Optimization Toolbox of Matlab. The source files of the considered program will be made available at the web site act.put.poznan.pl of the Applied Control Techniques Research Group.

2. CLASSIFICATION OF CUTTING PROBLEMS

In general, cutting problems can be divided into three main categories, namely: 1D problems, 2D problems and 3D problems [11, 3, 5].

A 1D cutting problem is about optimizing in a single dimension, and is one of the most common combinatorial optimization tasks, widely used in industry. Its purpose is to minimize the waste resulted from cutting process, e.g., in carpentry or paper industries. Its assumptions can be formulated in the following task [3, 7]:

$$\min_{\mathbf{x}} \quad \sum_{i=1}^{n} c_i x_i$$

subject to:

$$\sum_{i=1}^{n} a_{ij} x_i \geq b_j \quad (j = 1, \ldots, m),$$

where $a_{ij}$ denotes the number of occurrences of the $j$-th component in the $i$-th cutting pattern, $c_i$ is the cost of the $i$-th cutting pattern, $x_i$ is the number of occurrences of the $i$-th cutting pattern in the considered problem and $b_j$ is the demand for the planks of $j$-th length, and, finally, $m$ is the length of the demand list.

A standard cutting problem is an NP-hard problem, and in practice finding its solution is problematic [7].

A 2D cutting problem, such as cutting rectangular shapes from a larger rectangular block of material, is performed in a two-dimensional space, and is, again, NP-hard. Finally, a 3D cutting problem can be imagined as cutting smaller blocks from a larger block of stone. It is, obviously, NP-hard again.

© Authors and Poznańskie Towarzystwo Przyjaciół Nauk 2018, sait.cie.put.poznan.pl
3. APPROACHES TO SOLVING CUTTING PROBLEMS

3.1. INTRODUCTION

As has been already remarked, the 1D cutting problem is one of the most commonly encountered optimization problems, and it has multiple solution methods, what enables one to choose the most appropriate [3].

One of the solving methods is a linear programming algorithm, defined for problems with linear constraints and linear objective function. The set of admissible solutions is an \( n \)-dimensional convex polytope, where \( n \) denotes the number of variables in a cutting task. In order to find the optimal solution to the given task, a set of ordered iterations is performed and related to consecutive feasible solutions which lie on the vertices of the polytope. An example of such an algorithm is the primal matrix simplex algorithm [13, 5].

The alternative method that can be used to solve the cutting problem is the integer programming algorithm, used usually when the excess of the error caused by rounding partial solutions of an LP algorithm are not acceptable, i.e., when rounded solutions become infeasible. An example of such an algorithm is the branch and bound (BB) method [8, 1] that creates a pair of subproblems with separate sets of feasible solutions, repeating this procedure recursively, to find the final, optimal solution. The advantage of this approach is finding the optimal solution without rounding operations, but, on the contrary, as in the LP case, it requires all the cutting patterns to be defined prior to optimization stage.

In order to avoid the common drawback of LP and BB approaches, and especially for large-scale problems, one can use Column Generation method (CG) [3, 5]. It is based on the assumption that in a large-scale problem the majority of variables should lie in a non-basic set, i.e., should have zero values, thus only a part of these variables should be considered when solving the problem. To initialize this method, it is required to have the matrix defining constraints of size \( n_L \times n_L \) only, where \( n_L \) is the number of demanded lengths that are defined in the problem.

It is usually obtained by dividing the length of the basic plank by consecutive demanded lengths, what leads to obtaining cutting patterns. Consecutive cutting patterns are obtained by solving the auxiliary knapsack problem that is formulated in the following way [10, 3]:

\[
\begin{align*}
\max \quad & \mathbf{z}^T \mathbf{c}_B \mathbf{A}_B^{-1} \mathbf{z} \\
\text{s.t.} \quad & \mathbf{l}^T \mathbf{z} \leq L, \\
& \mathbf{z} \in \mathbb{Z}^n_+,
\end{align*}
\]

where \( \mathbf{c}_B \) is the basic prices vector, \( \mathbf{A}_B \) is the basic constraint matrix, \( \mathbf{z} \) is the vector of decision variables, \( \mathbf{l} \) is the vector of demanded lengths, and \( L \) is the length of the basic plank (i.e., of planks available for cutting).

In solving the knapsack problem, dynamic programming is useful. A discussion of computational complexity of this algorithm is included in [6].

3.2. COLUMN GENERATION METHOD

The CG method is used to generate columns of the constraint \( \mathbf{A} \) matrix, generated to improve the value of the objective function by proper cutting pattern selection. The generated
matrix can be used to solve the optimization problem. The advantage of this approach is a shortened computation time by choosing only these cutting patterns that improve the objective function, whereas the drawback is the necessity to solve the auxiliary knapsack problem.

The CG method can be summarized as below [5]:

1) start with the basic matrix $A_B \in \mathbb{R}^{m \times m}$ that contains a feasible cutting patterns (e.g., make this matrix diagonal),
2) solve the auxiliary problem (1), denote $x^*$ as its optimal solution, and $f^*$ as the optimal value of the objective function,
3) if the reduced prices vector is nonnegative for all basic variables, terminate the algorithm – the optimal cutting pattern has been found,
4) otherwise, the basic variable with the most negative reduced price should enter the basis, since it is the most profitable cutting pattern,
5) find the variable to leave the basis by the standard ratio test,
6) form the set of new basic variables, and proceed to Step 2.

Once the possible cutting patterns are found via CG algorithm, the main problem is ready to be solved.

4. HOW TO DEFINE THE OBJECTIVE FUNCTION

4.1. PRELIMINARIES

A basic element in formulation of the optimal cutting problem is to define a proper objective function [14]. It should be chosen to mimic the requirements imposed on the cutting problem. In general, one can distinguish two basic and one additional form of the objective function. The first approach is to minimize the waste, the other is to minimize the overall cost of the planks, whereas the third is a linear convex combination of the first two with some coefficient $\alpha$. It is to be stressed that all the methods can return the final matrix which is non-square. It is caused by choosing only non-slack variables to the final solution, thus whenever the algorithm puts the cutting pattern related to the slack variable in the basic optimal solution, this pattern will not be visible in the final matrix $A_F$.

4.2. MINIMUM WASTE

This approach is the most common one, and is based on specifying the waste for all cutting patterns. It can be put in the form [5]

$$W_1 = [s_1, s_2, \ldots, s_{n_L}],$$

$$s_i = L - b^T a_i,$$

where $s_i$ is the waste generated by the $i$-th cutting pattern, $L$ is the length of the basic plank that is cut into demanded lengths, $a_i$ is the $i$-th column of the $A_F$ matrix, $n_L$ is the number of cutting patterns considered in the problem, and $m$ is the number of constraints in the problem.

As an example, consider the following problem of optimal cutting:

$$l = [11, 13, 15]^T,$$

$$d = [100, 100, 100]^T,$$

$$L = 50.$$
where the above symbols refer to demanded lengths, demanded quantities and the length of the basic plank, respectively. The optimal solution becomes:

\[
A_F = \begin{bmatrix}
1 & 2 \\
3 & 1 \\
0 & 1
\end{bmatrix},
\]

\[x^* = [41, 101]^T,\]

\[f(x^*) = 0,\]

\[p = [249, 242, 101]^T,\]

with the final matrix \(A_F\) giving possible cutting patterns, \(x^*\) depicting the multiplicity of the use of each of the patterns, \(f(x^*)\) denoting total waste, and \(p\) presenting the number of the obtained planks.

As can be seen, when a pattern is related to zero waste, the simplex method may use it many times without increase of the objective function, what is a serious drawback, since excessive planks may be cut. The second drawback is that formulation of the prices vector for the optimization task is complicated and not as easy as in the second approach.

4.3. **MINIMUM COST**

This approach improves the optimization results in comparison to the results of the minimum waste approach. It is based on giving a unity value for all elements of the objective function, what can be thought as the increase of the objective function whenever any cutting pattern is used. This approach avoids generation of zero-waste solutions without increasing the objective function, but, on the contrary, the total waste is increased, since it is not taken here into consideration. The objective function can be put in the form

\[W_2 = [\mathbf{1}^T_n, \mathbf{0}^T_m]^T,\]

where \(n_L\) is the number of cutting patterns considered in the problem, and \(m\) is the number of constraints in the problem.

In this method, every cutting pattern is related to the increase of the objective function by 1. In this way, the number of used basic planks is minimized, and the solution to the optimal cutting problem considered above is:

\[
A_F = \begin{bmatrix}
0 & 1 & 2 \\
0 & 3 & 1 \\
3 & 0 & 1
\end{bmatrix},
\]

\[x^* = [21, 21, 40]^T,\]

\[f(x^*) = 105,\]

\[p = [101, 103, 103]^T.\]

As can be seen the number of excessive planks is reduced, what is a major advantage of this method. The other advantage is the simplicity of formulation of the objective function. The basic drawback is that the approach does not take waste into consideration.

© Authors and Poznański Towarzystwo Przyjaciół Nauk 2018, sait.cie.put.poznan.pl
4.4. **HYBRID APPROACH**

In the third approach, the convex linear combination of $W_1$ and $W_2$ is created for some $0 \leq \alpha \leq 1$, i.e.,

$$W_3 = \alpha W_1 + (1 - \alpha)W_2.$$  

By the proper choice of $\alpha$ it is possible to define the impact of the first two approaches on the final solution to the problem.

For the problem of optimal cutting from the first approach, with $\alpha = 0.5$, the optimal solution is:

$$A_F = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix},$$

$$x^* = \begin{bmatrix} 101 \end{bmatrix},$$

$$f(x^*) = 0,$$


And as can be seen the number of excessive planks is reduced again, keeping the minimal waste.

5. **PROGRAM DESCRIPTION**

5.1. **INTRODUCTION**

Three major aspects need to be considered when describing the program. The first two are connected with the implementation of the basic functions, and the latter is connected to GUI designing. The first function is responsible for cutting problem formulation and formulation of the auxiliary knapsack problem. It also calls the dynamic programming solver to solve a knapsack problem, launches LP solver and presents the optimization results. The second function is responsible for solving the auxiliary knapsack problem and passing the results to the master function. The graphical interface is intended to supply a comfortable mean to communicate the optimization results to the user.

5.2. **CuttingStockSolver FUNCTION**

The first three arguments of the function are mandatory and must be introduced in the given order, whereas the remaining arguments are optional, and their values should be given in key-value pairs.

- **L** length of the basic plank (scalar value),
- **D** vector of demanded cutting lengths, of length $n_L$,
- **d_det** accuracy level of the information printed in the console (scalar):
  - 0 – silent mode,
  - 1 – optimal solution printed in the console,
2 – optimal and intermediate solutions printed in the console (default),
3 – complete information printed in the console;

d_dp accuracy level of the information printed in the console from dynamic programming algorithm (scalar):
0 – silent mode,
1 – optimal solution printed in the console,
2 – optimal and final DP tableau printed in the console (default),
3 – complete information printed in the console;

Ab initial basic matrix (when omitted, is calculated automatically), of size $n_L \times n_L$,
fo output format (0 – rational, 1 – real numbers),
price price of a single basic plank (when omitted, put to 1), scalar,
met objective function generation method: 0 – $W_1$, 1 – $W_2$, 2 – $W_3$.

Possible output values are summarized below:
1) objective function value,
2) decision variables (size $1 \times n$),
3) flag: 1 – optimal solution found, −1 – error occurred, the function is in infinite loop,
4) waste,
5) obtained cut planks (size $1 \times n_L$),
6) final basic matrix (size $n_L \times n_L$).

5.3. DynamicProgramming FUNCTION

Its first three arguments are mandatory and must be introduced in the given order, whereas the remaining arguments are optional, and their values should be given in key-value pairs.

c prices vector in a knapsack problem (length $n_L$),
w weights vector in a knapsack problem (length $n_L$),
D right-hand side constraint in a knapsack problem (a positive natural number),
d_dis accuracy level of the information printed in the console (scalar):
0 – silent mode,
1 – optimal solution printed in the console,
2 – optimal and final DP tableau printed in the console (default),
3 – complete information printed in the console;
fo output format (0 – rational, 1 – real numbers).

Possible output values are summarized below:
1) objective function value,
2) decision variables (size $1 \times n$).
6. GRAPHICAL USER INTERFACE

The GUI is intended to streamline the use of the program, enabling visualisation of the obtained results in an illegible way. Initially, the user can define the new problem, load the existing problem or load the solution of the previously defined problem. In Figure 1, the screen with definition of the new problem is given. Every component of this window is properly described with a tool tip string included. In addition, every window has the BACK button, enabling one to redefine the given data, and to obtain the new solution with, e.g., another method.

Figure 2 presents the results of a sample cutting stock problem. In this window the user can save the optimization results, redefine the input data to the problem or save the displayed solution diagram. The information displayed in the window repeats the input data given in the previous window.

7. SAMPLE CUTTING PROBLEMS

In order to present the possible results obtained from the program, three sample problems are formulated and solved. In Figure 3, the solution to cutting Problem 1 with:

\[
\begin{align*}
L &= [10, 18, 22, 30]^T, \\
D &= [121, 78, 213, 187]^T, \\
L &= 50
\end{align*}
\]

is presented with the objective function \(W_1\), and the obtained solution

\[
x^* = [390, 78, 214, 233]^T
\]

is far greater than the demand.

The solution for Problem 2:

\[
\begin{align*}
L &= [3, 4, 5, 6, 7, 8]^T, \\
D &= [56, 71, 41, 93, 30, 55]^T, \\
L &= 20
\end{align*}
\]

is presented in Figure 4 with \(W_2\). As can be seen, the input data allowed the zero waste to be obtained, and the obtained solution

\[
x^* = [61, 74, 44, 94, 33, 57]^T
\]

Finally, In Figure 5, the solution to cutting Problem 3 with:

\[
\begin{align*}
L &= [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]^T, \\
D &= [75, 102, 67, 89, 1, 52, 75, 102, 67, 89, 17, 13]^T, \\
\alpha &= 0.85, \\
L &= 120
\end{align*}
\]
is presented with the objective function $W_3$, and the obtained solution

$$x^* = [79, 102, 69, 94, 12, 54, 131, 104, 68, 92, 18, 15]^T$$.

From Figure 6, it can be seen that the increase in knapsack capacity and in number of variables causes the number of iterations of the Bottom-Up algorithm to increase, what has been a case of prior research. However, for a large problem with 500 variables and capacity equal to 500, the mean number of iterations becomes 687.5, what verifies the applicability of the dynamic programming algorithm presented here.

8. SUMMARY

The paper includes a thorough description of design stages of the cutting stock solver, accompanied by syntax of functions written in Matlab to enable further development of the presented application. Moreover, by making the application available and in open code, there exists a possibility to include new methods, such as for solving the knapsack problem or column generation, what should be attractive for researchers. Finally, some functions can be potentially replaced by heuristics to test the algorithm in action.

The presented program is fully functional, enabling fast calculation of the results accompanied by their visualisation.

For a discussion of limits of applicability, mean number of iterations per task, see the paper [6], where among other results Figure 6, presents the complexity of the core Bottom-Up algorithm, solving dynamic programming problem.

REFERENCES


© Authors and Poznańskie Towarzystwo Przyjaciół Nauk 2018, sait.cie.put.poznan.pl
Fig. 1. The main screen of the program

Fig. 2. Sample optimization results

© Authors and Poznańskie Towarzystwo Przyjaciół Nauk 2018, sait.cie.put.poznan.pl
Fig. 3. Solution to Problem 1

Fig. 4. Solution to Problem 2

© Authors and Poznańskie Towarzystwo Przyjaciół Nauk 2018, sait.cie.put.poznan.pl
Fig. 5. Solution to Problem 3

Fig. 6. Mean iteration count of the Bottom-Up algorithm [6]
ABSTRACT
The paper summarizes the procedure of solving 1D optimal cutting problems, giving details of coding it using dynamic programming, knapsack problem formulation and column generation approach. Finally, the software program for optimal 1D cutting support is described, which is the open code version enabling researchers to extend its capabilities. The paper ends by giving solutions to stated problems and the description of the GUI of the program. At the end of the paper, the reference to the other paper of the authors discussing the effectiveness of the proposed solution is given, tightly connected with this paper.

PROGRAM WSPOMAGAJĄCY PROCES OPTYMALNEGO DOCINANIA 1D
Mateusz Pacek, Dariusz Horla
W artykule przedstawiono procedurę rozwiązywania zadania optymalizacji docinania 1D, włączając szczegóły zakodowania algorytmu przy użyciu programowania dynamicznego, formalizmu zadania plecakowego oraz metody generowania kolumn. W końcu, opisano program wspomagania optymalnego docinania, w postaci programu w otwartym kodzie co pozwoli badaczom na jego dalsze rozwijanie. Artykuł zawiera rozwiązania przykładowych zadań optymalizacji, jak i opis interfejsu użytkownika. Zawarto również odnośnik do drugiego artykułu autorów, przedstawiającego efektywność zastosowanych metod.

Received: 2018-09-10
Accepted: 2018-11-03

© Authors and Poznańskie Towarzystwo Przyjaciół Nauk 2018, sait.cie.put.poznan.pl