VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MATHEMATICAL MODELING (CO2011)

Assignment

"Cutting stock problem"

Instructor(s): Mai Xuân Toàn

Students: Tăng Phồn Thịnh - 2213307
Đỗ Hoàng Phúc - 2212611
Nguyễn Trần Đăng Khoa - 2211635
Nguyễn Đặng Thức - 2213428
Lương Văn Khanh - 2211486

HO CHI MINH CITY, July 2024

Mục Lục

Danh sách kí hiệu

c Vector chứa các hệ số của hàm mục tiêu.

x Vector của các biến quyết định.

A Ma trận chứa các hệ số của các ràng buộc.

b Vector chứa các giá trị vế phải của các ràng buộc.

B Tập hợp các chỉ số cột của các biến cơ sở.

N Tập hợp các chỉ số cột của các biến không cơ sở.

B^{-1} Ma trận nghịch đảo của ma trận cơ sở B.

c_B Vector chứa các hệ số của hàm mục tiêu tương ứng với các biến cơ sở.

c_N Vector chứa các hệ số của hàm mục tiêu tương ứng với các biến không cơ sở.

a_j Cột thứ j của ma trận A.

r_N Vector các reduce cost của các biến không cơ sở.

$\overline{a}_j$ Vector các thành phần của a_j biểu diễn trong cơ sở B.

λ Bước nhảy trong hướng tối ưu.

d_j Vector chỉ hướng di chuyển.

Danh sách các từ viết tắt

ODE (First-Order) Ordinary Differential Equation

IVP Initial-Value Problem

LTE Local Truncation Error

DS Dynamical System

Fig. Figure

Tab. Table

Sys. System of Equations

Eq. Equation

e.g. For Example

i.e. That Is

Danh sách các hình ảnh

Danh sách các bảng

Danh sách thành viên & Lượng công việc

No.	Tên	MSSV	Công việc	% Hoàn thành
1	Tăng Phồn Thịnh	2213307	- Algorithm - Modeling - Analyze results	100%
2	Đỗ Hoàng Phúc	2212611	- Algorithm - Modeling - Analyze results	100%
3	Nguyễn Trần Đăng Khoa	2211635	- Introduction - Acknowledge - LaTeX	100%
4	Nguyễn Đặng Thức	2213428	- Algorithm - Modeling - Analyze results	100%
5	Lương Văn Khanh	2211486	- Algorithm - Modeling - Analyze results	100%

Table 1: Danh sách thành viên & Lượng công việc

1 Giới Thiệu

1.1 Tổng quan

Bài toán Cutting-Stock Problem (CSP) là một thách thức tối ưu hóa cổ điển xuất hiện trong nhiều ngành công nghiệp, bao gồm sản xuất giấy, kim loại và dệt may. Bài toán này liên quan đến việc cắt nguyên liệu thô, thường dưới dạng cuộn hoặc thanh dài, thành các chiều dài nhỏ hơn theo yêu cầu của khách hàng. Mục tiêu chính là giảm thiểu lãng phí bằng cách tối ưu hóa các mẫu cắt được sử dụng. Giải quyết hiệu quả CSP có thể dẫn đến tiết kiệm chi phí đáng kể và cải thiện việc sử dụng tài nguyên, điều này làm cho nó trở thành một trọng tâm quan trọng đối với các ngành công nghiệp phụ thuộc nhiều vào xử lý nguyên liệu thô.

Figure 1: Minh họa về bài toán cắt thanh một chiều trong sản xuất giấy.

1.2 Định nghĩa Vấn đề

Bài toán Cutting-stock Problem có thể được định nghĩa là tìm cách hiệu quả nhất để cắt một tập hợp nguyên liệu thô cho trước thành các miếng nhỏ hơn để đáp ứng một tập hợp nhu cầu cho các chiều dài khác nhau, đồng thời giảm thiểu phần dư thừa. Vấn đề này đòi hỏi phải xác định các mẫu cắt tối ưu để đáp ứng nhu cầu cho từng kích thước miếng, đồng thời sử dụng ít đơn vị nguyên liệu thô nhất có thể. Sự phức tạp nảy sinh từ nhu cầu cân bằng nhiều yêu cầu với nguồn cung cấp nguyên liệu thô hạn chế, trong khi vẫn giảm thiểu vật liệu thừa.

Về mặt toán học, CSP có thể được mô hình hóa như một bài toán quy hoạch tuyến tính nguyên, trong đó mục tiêu là giảm thiểu lãng phí (hoặc tối đa hóa việc sử dụng vật liệu) theo các ràng buộc đảm bảo đáp ứng tất cả các yêu cầu về nhu cầu. Công thức này cho phép sử dụng các kỹ thuật tối ưu hóa tiên tiến để tìm ra các giải pháp khó có thể đạt được bằng tay.

1.3 Lịch sử và Phát triển

Bài toán Cutting-stock Problem có nguồn gốc từ các vấn đề thực tế trong sản xuất và đã được nghiên cứu rộng rãi từ giữa thế kỷ 20. Một trong những công trình đầu tiên về vấn đề này được Richard Bellman đề xuất vào năm 1957 trong bối cảnh lập trình động. Bellman đã đưa ra các phương pháp để giải quyết các vấn đề tối ưu hóa bằng cách chia chúng thành các bài toán con nhỏ hơn, có thể giải quyết một cách lặp đi lặp lại, tạo nền tảng cho nhiều thuật toán tối ưu hóa hiện đại. Một cột mốc quan trọng khác trong sự phát triển của CSP là công trình của Gilmore

Figure 2: Richard Bellman

và Gomory vào đầu những năm 1960, trong đó họ giới thiệu một phương pháp lập trình tuyến tính để giải quyết các bài toán cắt thanh. Phương pháp này đã tạo ra một nền tảng mạnh mẽ cho việc áp dụng các kỹ thuật tối ưu hóa trong các bài toán công nghiệp thực tế, và được coi là một trong những bước đột phá quan trọng trong lĩnh vực này.

Trong những năm gần đây, sự phát triển của công nghệ máy tính và phần mềm tối ưu hóa đã giúp cải thiện khả năng giải quyết các bài toán cắt thanh phức tạp hơn, mở ra nhiều ứng dụng trong các ngành công nghiệp hiện đại.

1.4 Mục tiêu

Mục tiêu chính của bài tập lớn này là phát triển một giải pháp hiệu quả cho bài toán 1-D cutting-stock problem bằng cách mô hình hóa nó như một mô hình quy hoạch tuyến tính nguyên. Các mục tiêu bao gồm:

- **Giảm thiểu lãng phí nguyên liệu:** Bằng cách tối ưu hóa các mẫu cắt, bài tập nhằm giảm thiểu lượng nguyên liệu thô bị lãng phí trong quá trình cắt.

- **Giảm chi phí:** Việc sử dụng hiệu quả nguyên liệu có thể dẫn đến giảm chi phí đáng kể, điều này rất quan trọng để duy trì tính cạnh tranh trong các ngành công nghiệp sử dụng nhiều tài nguyên.
- **Nâng cao hiệu quả sản xuất:** Thực hiện các giải pháp cắt tối ưu có thể hợp lý hóa quy trình sản xuất, giảm thời gian sản xuất và cải thiện hiệu quả hoạt động tổng thể.
- **Cung cấp giải pháp mở rộng:** Phát triển mô hình và cách tiếp cận có thể được mở rộng để xử lý các vấn đề lớn hơn và phức tạp hơn trong các bối cảnh công nghiệp khác nhau.

Figure 3: Ví dụ về bài toán cắt thanh trong sản xuất thép.

2 Kiến thức nền

2.1 Quy hoạch tuyến tính (Linear Programming)

2.1.1 Giới Thiệu

Quy hoạch tuyến tính (LP) là một phương pháp tối ưu hóa được sử dụng để tìm giá trị tốt nhất (ví dụ: giá trị tối đa hoặc tối thiểu) của một hàm tuyến tính, với các ràng buộc được biểu diễn bằng các bất đẳng thức hoặc phương trình tuyến tính. Vấn đề LP tổng quát được định nghĩa như sau:

$$\text{(LP)} \quad z_{LP} = \max(cx : Ax \leq b,\, x \in \mathbb{R}^n) \tag{1}$$

Trong đó:

- A là ma trận hệ số có kích thước $m \times n$.
- b là vector hằng số có kích thước $m \times 1$.
- c là vector hệ số hàm mục tiêu có kích thước $1 \times n$.
- x là vector biến có kích thước $n \times 1$.

Vấn đề LP được coi là khả thi nếu tồn tại một vector x thỏa mãn tất cả các ràng buộc và nếu hàm mục tiêu không có giá trị tối ưu không giới hạn, thì tồn tại một nghiệm tối ưu.

Quy hoạch tuyến tính là nền tảng cho integer programming, nơi các biến bị ràng buộc phải là các số nguyên. Do đó, việc hiểu rõ LP là bước đầu tiên để giải quyết các vấn đề tối ưu hóa nguyên như Cutting-Stock Problem.

2.1.2 Ứng Dụng trong Cutting-Stock Problem

Trong Cutting-Stock Problem, mục tiêu là cắt các khổ vật liệu lớn thành các khổ nhỏ hơn với mục đích tối thiểu hóa phế liệu hoặc tối đa hóa lợi nhuận. Quy hoạch tuyến tính có thể được sử dụng để mô hình hóa vấn đề này bằng cách:

- Xác định hàm mục tiêu là tối đa hóa giá trị sử dụng của các mảnh cắt.
- Ràng buộc các khổ cắt sao cho không vượt quá kích thước của vật liệu ban đầu.

2.2 Đối ngẫu (Duality)

Đối ngẫu trong Quy hoạch tuyến tính liên quan đến việc tìm kiếm một chương trình tối ưu tương đương nhưng có cách biểu diễn khác. Vấn đề Đối ngẫu của bài toán nguyên thủy (primal) giúp cung cấp thêm thông tin và cải thiện sự hiểu biết về cấu trúc của các giải pháp tối ưu.

2.2.1 Vấn Đề Nguyên Thủy và Đối ngẫu

Vấn đề nguyên thủy (primal) được định nghĩa như sau:

$$\text{(P)} \quad z_{LP} = \max(cx : Ax \leq b,\, x \in \mathbb{R}^n) \tag{2}$$

Đối ngẫu của nó được định nghĩa là:

$$\text{(D)} \quad w_{LP} = \min(ub : uA \geq c,\, u \in \mathbb{R}^m_+) \tag{3}$$

Trong đó u là vector biến Đối ngẫu. Mối quan hệ giữa vấn đề nguyên thủy và Đối ngẫu được thể hiện qua các định lý Đối ngẫu, như định lý Đối ngẫu yếu và định lý Đối ngẫu mạnh, cho thấy rằng giá trị của hàm mục tiêu trong vấn đề nguyên thủy không lớn hơn giá trị trong vấn đề Đối ngẫu.

2.2.2 Lợi Ích của Đối ngẫu

Đối ngẫu cung cấp các ràng buộc bổ sung và giúp kiểm tra tính hợp lệ của nghiệm. Trong bối cảnh Cutting-Stock Problem, phân tích Đối ngẫu có thể được sử dụng để tìm các cận dưới hoặc cận trên cho giá trị tối ưu của vấn đề và giúp xác định xem liệu một nghiệm đã tìm thấy có phải là tối ưu hay không.

2.3 Thuật Toán Simplex

Thuật toán Simplex là một phương pháp lặp để giải quyết bài toán Quy hoạch tuyến tính dạng chuẩn. Nó tìm kiếm nghiệm tối ưu của một hàm mục tiêu tuyến tính, được biểu diễn dưới dạng:

$$\max z = c^T x \tag{4}$$

với các ràng buộc:

$$Ax = b, \tag{5}$$
$$x \geq 0, \tag{6}$$

trong đó A là ma trận các hệ số có kích thước $m \times n$, b là vector hằng số có kích thước $m \times 1$, c là vector hệ số hàm mục tiêu có kích thước $n \times 1$, và x là vector biến có kích thước $n \times 1$.

2.3.1 Một số định nghĩa về Nghiệm Cơ Bản và Điểm Cực Trị (basic solutions & extreme points)

- Tập khả thi: Gọi $S = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ là tập khả thi của bài toán Quy hoạch tuyến tính (LP). Vì ma trận A có hạng đầy đủ, nếu tập khả thi không rỗng thì ta phải có $m \leq n$. Không mất tính tổng quát, ta giả sử $m < n$.
- Cơ sở: Gọi $A = (B, N)$, trong đó B là ma trận vuông cấp m có hạng đầy đủ, tức là $\det(B) \neq 0$. Khi đó, B được gọi là một **cơ sở**.
- Nghiệm cơ bản: Giả sử $x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$. Ta có $Bx_B + Nx_N = b$. Bằng cách đặt $x_N = 0$, ta được $x_B = B^{-1}b$. Vậy $x = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$ được gọi là một **nghiệm cơ bản**. Các phần tử của x_B được gọi là các **biến cơ bản**, và các phần tử của x_N được gọi là các **biến không cơ bản**.
- Nghiệm cơ bản khả thi: Nếu nghiệm cơ bản cũng là một nghiệm khả thi, tức là $B^{-1}b \geq 0$, thì $x = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$ được gọi là một **nghiệm cơ bản khả thi**.
- Điểm cực trị: Một điểm $\hat{x} \in S$ được gọi là một **điểm cực trị** của S khi và chỉ khi $\hat{x}$ là một **nghiệm khả thi cơ bản**.

- Điểm cực trị kề nhau: Hai điểm cực trị được gọi là **kề nhau** nếu chúng chỉ khác nhau bởi một biến cơ bản.

Theorem 1 (Định lý cơ bản của LP) *Nếu tập khả thi S của bài toán LP có ít nhất một điểm cực trị và tồn tại một nghiệm tối ưu, thì tồn tại một nghiệm tối ưu là một điểm cực trị.*

Từ định lý trên, ta có các kết luận sau:

- Tập khả thi của một bài toán LP dạng chuẩn luôn có ít nhất một điểm cực trị.
- Giá trị tối ưu của một bài toán LP hoặc bằng $-\infty$, hoặc đạt được tại một điểm cực trị (nghiệm khả thi cơ bản) của tập khả thi.

Trong phương pháp đơn hình, nếu tồn tại một phần tử $r_i < 0$, chúng ta có thể để biến không cơ bản hiện tại x_i trở thành một biến cơ bản với $x_i > 0$ (biến vào).

- Bằng cách chọn phù hợp biến cơ bản để trở thành biến không cơ bản (biến ra), ta có thể đạt được một nghiệm cơ bản khả thi mới có giá trị hàm mục tiêu nhỏ hơn giá trị của nghiệm cơ bản khả thi hiện tại $\hat{x}$.
- Về mặt hình học, phương pháp đơn hình di chuyển từ một điểm cực trị tới một trong những điểm cực trị kề nó.
- Vì chỉ có một số lượng hữu hạn các điểm cực trị, phương pháp sẽ kết thúc sau hữu hạn bước tại một nghiệm tối ưu hoặc phát hiện ra rằng bài toán vô nghiệm hoặc không bị chặn.

2.3.2 Nguyên Lý Hoạt Động

Thuật toán Simplex hoạt động trên ý tưởng di chuyển giữa các đỉnh của đa diện khả thi (feasible polytope) cho đến khi đạt được nghiệm tối ưu. Mỗi đỉnh tương ứng với một tập nghiệm cơ bản khả thi (basic feasible solution).

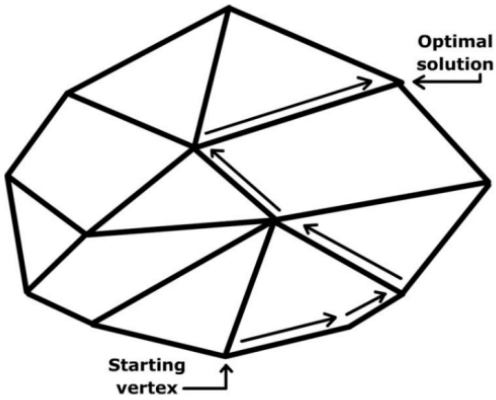


Figure 4: Simplex method

2.3.3 Bước Điển Hình của Thuật Toán

Quy trình của Simplex có thể được mô tả theo các bước sau:

1. Khởi tạo: Tính một cơ sở ban đầu B và nghiệm khả thi cơ bản tương ứng:
$$x = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$$
2. Kiểm tra tối ưu: Nếu $r_N = (c_N^T - c_B^T B^{-1} N) \geq 0$, dừng thuật toán, x là nghiệm tối ưu. Ngược lại, chuyển đến bước 2.
3. Chọn biến vào: Chọn chỉ số j sao cho $c_j^T - c_B^T B^{-1} a_j < 0$. Nếu $\overline{a}_j = B^{-1}a_j \leq 0$, dừng thuật toán, bài toán LP vô nghiệm. Ngược lại, chuyển đến bước 3.
4. Tính bước nhảy: Tính bước nhảy λ:
$$\lambda = \min_{\{i|\overline{a}_{ij}>0\}} \frac{b_i}{\overline{a}_{ij}}$$
Cập nhật nghiệm:
$$x := x + \lambda d_j, \text{ với } d_j = \begin{bmatrix} B^{-1}a_j \\ e_j \end{bmatrix}$$
Quay lại bước 1.

2.3.4 Ví Dụ minh họa

Tối ưu hóa hàm mục tiêu:

$$\text{maximize } z = 3x_1 + 2x_2$$

Với các ràng buộc:

$$\begin{aligned} x_1 + x_2 &\leq 4, \\ 2x_1 + x_2 &\leq 5, \\ x_1, x_2 &\geq 0. \end{aligned}$$

Chuyển Đổi Dạng Chuẩn

Thêm biến trượt s_1 và s_2 để chuyển đổi các bất đẳng thức thành đẳng thức:

$$\begin{aligned} x_1 + x_2 + s_1 &= 4, \\ 2x_1 + x_2 + s_2 &= 5. \end{aligned}$$

Nghiệm cơ bản khả thi ban đầu là $(x_1, x_2, s_1, s_2) = (0, 0, 4, 5)$.

Bảng Đơn Hình Ban Đầu

Biến cơ bản	x_1	x_2	s_1	s_2	Vế phải
s_1	1	1	1	0	4
s_2	2	1	0	1	5
z	-3	-2	0	0	0

Xác Định Biến Vào và Ra

Biến vào: x_1 (hệ số âm lớn nhất trong hàng hàm mục tiêu).
Biến ra: s_2 (tỷ lệ nhỏ nhất $\frac{4}{1}, \frac{5}{2}$).

Chuyển Đổi Bảng Đơn Hình Mới

Biến cơ bản	x_1	x_2	s_1	s_2	Vế phải
s_1	0	0.5	1	-0.5	1
x_1	1	0.5	0	0.5	2.5
z	0	-0.5	0	1.5	7.5

Biến vào tiếp theo: x_2.
Biến ra tiếp theo: s_1.

Bảng Đơn Hình Cuối Cùng

Biến cơ bản	x_1	x_2	s_1	s_2	Vế phải
x_2	0	1	2	-1	2
x_1	1	0	-1	1	1
z	0	0	1	1	9

Nghiệm tối ưu là $x_1 = 1$, $x_2 = 2$ với giá trị hàm mục tiêu tối đa $z = 9$.

2.3.5 Ưu Điểm và Hạn Chế

- **Ưu Điểm:**
  - *Hiệu quả trong thực tế:* Thường giải quyết các bài toán tuyến tính lớn một cách hiệu quả.
  - *Khả năng phân tích Đối ngẫu:* Cung cấp thông tin Đối ngẫu và phân tích độ nhạy.
- **Hạn Chế:**
  - *Cycling:* Có thể gặp vòng lặp vô hạn trong một số trường hợp đặc biệt.
  - *Tốc độ hội tụ:* Không phải lúc nào cũng có thời gian chạy tốt nhất về lý thuyết cho các trường hợp xấu nhất.

2.4 Giải thuật Branch and Bound

2.4.1 Giới thiệu

Đây là phương pháp chia để trị. Chúng ta chia một bài toán lớn thành một vài bài toán nhỏ hơn. (Đây là phần "phân nhánh"). Phần chinh phục được thực hiện bằng cách ước lượng độ tốt của một nghiệm mà chúng ta có thể đạt được cho mỗi bài toán nhỏ hơn (để làm điều này, có thể chúng ta phải chia bài toán thêm nữa, cho đến khi chúng ta đạt được một bài toán mà chúng ta có thể xử lý), đó là phần "cận".

Chúng ta sẽ sử dụng *linear programming relaxation* để ước lượng nghiệm tối ưu của một bài toán integer programming.

- Đối với mô hình integer programming P, mô hình quy hoạch tuyến tính mà chúng ta có được bằng cách bỏ qua yêu cầu rằng tất cả các biến phải là số nguyên được gọi là *linear programming relaxation* của P.

2.4.2 Các bước thực hiện

- Chia một bài toán thành các bài toán con.
- Tính toán linear programming relaxation của một bài toán con.
  - Bài toán LP không có nghiệm khả thi, kết thúc.
  - Bài toán LP có nghiệm tối ưu nguyên; kết thúc. So sánh nghiệm tối ưu với nghiệm tốt nhất mà chúng ta biết (nghiệm hiện tại).
  - Bài toán LP có nghiệm tối ưu tệ hơn nghiệm hiện tại, kết thúc.

Trong tất cả các trường hợp trên, chúng ta biết tất cả những gì cần biết về bài toán con đó. Chúng ta nói rằng bài toán con đó đã được xử lý.

- Bài toán LP có nghiệm tối ưu mà không phải tất cả đều là số nguyên, tốt hơn nghiệm hiện tại. Trong trường hợp này, chúng ta sẽ phải chia nhỏ bài toán con hơn nữa và lặp lại.

2.4.3 Ví dụ minh họa

$$\begin{aligned} \text{Maximize} \quad & Z = -x_1 + 4x_2 \\ \text{Với các ràng buộc} & \\ -10x_1 + 20x_2 &\leq 22 \\ 5x_1 + 10x_2 &\leq 49 \\ x_1 &\leq 5 \\ x_1, x_2 \geq 0, \ & x_1, x_2 \text{ là số nguyên} \end{aligned}$$

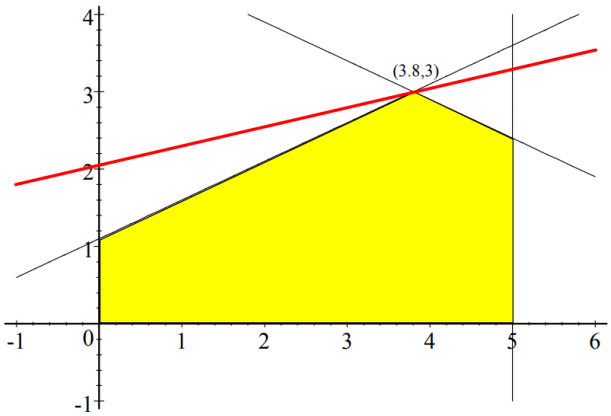


Figure 5: Feasible region

For the LP relaxation

$$\begin{aligned} \text{Maximize} \quad & Z = -x_1 + 4x_2 \\ \text{Với các ràng buộc} & \\ -10x_1 + 20x_2 &\leq 22 \\ 5x_1 + 10x_2 &\leq 49 \\ x_1 &\leq 5 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Giá trị tối ưu khi bỏ qua điều kiện nguyên là $(3.8, 3)$ với $Z = 28.2$. Vì vậy ta khảo sát 2 trường hợp: $x_1 \geq 4$ và $x_1 \leq 3$.

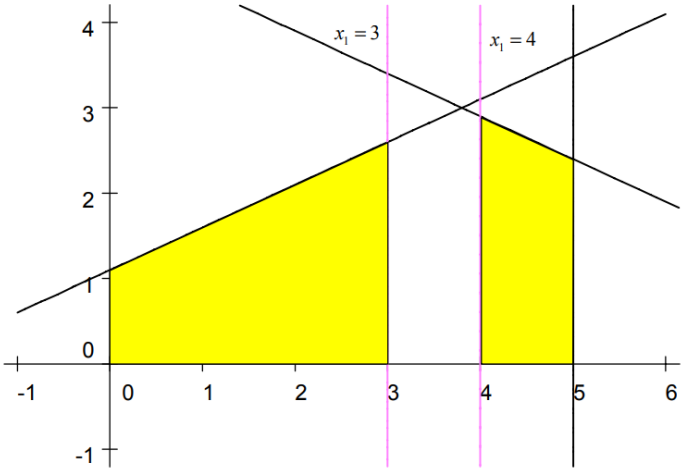


Figure 6: Feasible region

$$\begin{aligned} \text{Maximize} \quad & Z = -x_1 + 4x_2 \\ \text{Với các ràng buộc} & \\ -10x_1 + 20x_2 &\leq 22 \\ 5x_1 + 10x_2 &\leq 49 \\ x_1 &\leq 5 \\ x_1 &\geq 4 \\ x_2 &\geq 0 \end{aligned}$$

Giá trị tối ưu khi bỏ qua điều kiện nguyên là $(4, 2.9)$ với $Z = 7.6$. Vì vậy ta khảo sát 2 trường hợp: $x_1 \geq 4$ và $x_1 \leq 3$.

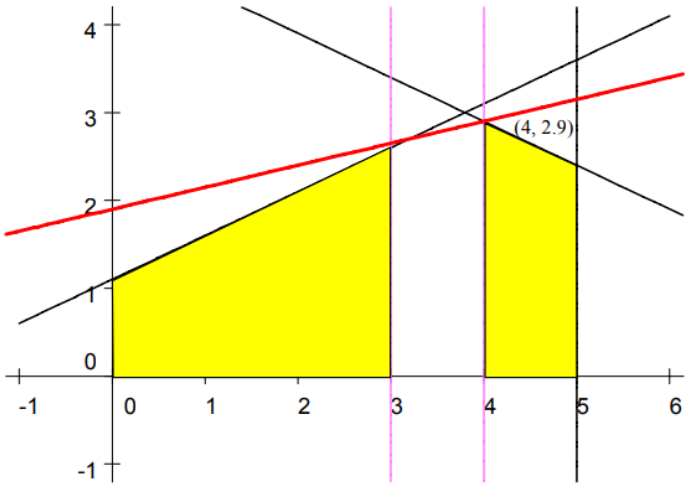


Figure 7: Feasible region

Xét bài toán:

$$\begin{aligned} \text{Max } Z &= -x_1 + 4x_2 \\ \text{subject to} & \\ -10x_1 + 20x_2 &\leq 22 \\ 5x_1 + 10x_2 &\leq 49 \\ 4 \leq x_1 &\leq 5 \\ x_2 &\geq 3 \end{aligned}$$

Bài toán trên không có nghiệm khả thi ($5x_1 + 10x_2 \geq 50$) nên bài toán quy hoạch tuyến tính (IP) ban đầu cũng không có nghiệm khả thi.

Xét bài toán:

$$\begin{aligned} \text{Max } Z &= -x_1 + 4x_2 \\ \text{subject to} & \\ -10x_1 + 20x_2 &\leq 22 \\ 5x_1 + 10x_2 &\leq 49 \\ 4 \leq x_1 &\leq 5 \\ 0 \leq x_2 &\leq 2 \end{aligned}$$

Bài toán có nghiệm tối ưu tại $(4, 2)$ với $Z = 4$. Đây cũng là nghiệm tối ưu của bài toán IP ban đầu. Hiện tại, giá trị tốt nhất của Z cho bài toán IP gốc là $Z = 4$.

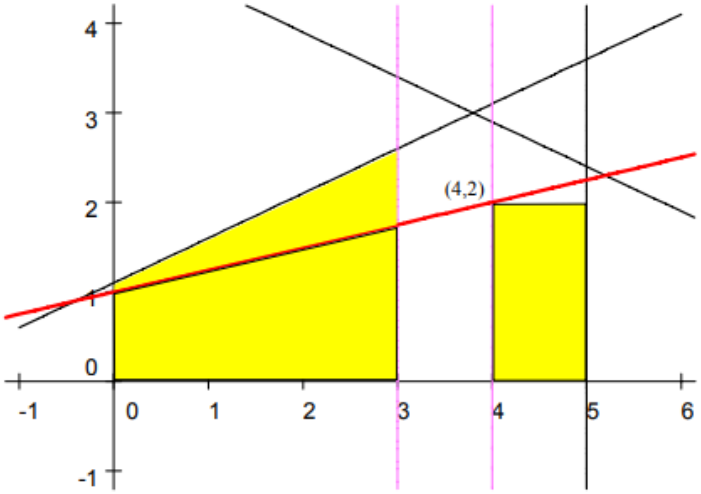


Figure 8: Feasible region

Bây giờ ta xét nhánh $0 \leq x_1 \leq 3$,

$$\begin{aligned} \text{Max } Z &= -x_1 + 4x_2 \\ \text{subject to} & \\ -10x_1 + 20x_2 &\leq 22 \\ 5x_1 + 10x_2 &\leq 49 \\ x_1 &\leq 3 \\ 0 &\leq x_i \end{aligned}$$

Giá trị tối ưu khi bỏ qua điều kiện nguyên là $(3, 2.6)$ với $Z = 7.4$. Vì vậy ta phân nhánh thêm 2 trường hợp: $x_2 \leq 2$ và $x_2 \geq 3$.

Xét bài toán:

$$\begin{aligned} \text{Max } Z &= -x_1 + 4x_2 \\ \text{subject to} & \\ -10x_1 + 20x_2 &\leq 22 \\ 5x_1 + 10x_2 &\leq 49 \\ x_1 &\leq 3 \\ x_2 &\geq 3 \end{aligned}$$

Bài toán trên không có nghiệm khả thi ($-10x_1 + 20x_2 \geq 30$). Do đó, bài toán lập trình nguyên (IP) ban đầu cũng không có nghiệm khả thi.

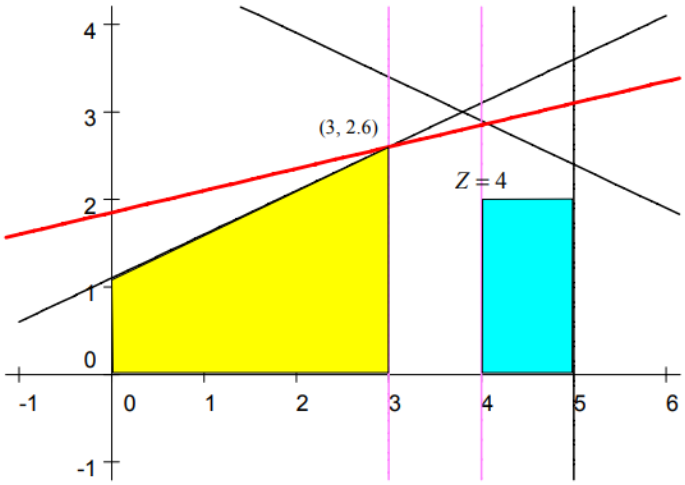


Figure 9: Feasible region

Xét bài toán:

$$\begin{aligned} \text{Max } Z &= -x_1 + 4x_2 \\ \text{subject to} & \\ -10x_1 + 20x_2 &\leq 22 \\ 5x_1 + 10x_2 &\leq 49 \\ 0 \leq x_1 &\leq 3 \\ 0 \leq x_2 &\leq 2 \end{aligned}$$

Bài toán có nghiệm tối ưu tại $(1.8, 2)$ với $Z = 6.2$.

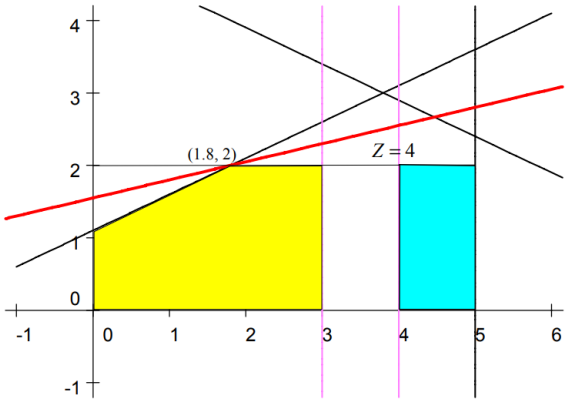


Figure 10: Feasible region

Chúng ta lại chia thành 2 trường hợp: $x_1 \geq 2 or x_1 \leq 1$ (chúng ta vẫn có $0 \leq x_2 \leq$).

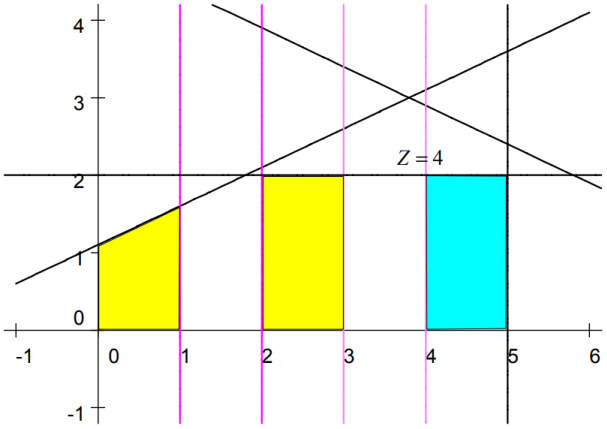


Figure 11: Feasible region

Xét bài toán:

$$\begin{aligned} \text{Max } Z &= -x_1 + 4x_2 \\ \text{subject to} & \\ -10x_1 + 20x_2 &\leq 22 \\ 5x_1 + 10x_2 &\leq 49 \\ 2 \leq x_1 &\leq 3 \\ 0 \leq x_2 &\leq 2 \end{aligned}$$

Giá trị tối ưu khi bỏ qua điều kiện nguyên là $(2, 2)$ với $Z = 6$. Bởi vì đây là nghiệm cho ra kết quả lớn nhất từ trước đến giờ nên sẽ trở thành best solution.

Xét bài toán:

$$\begin{aligned} \text{Max } Z &= -x_1 + 4x_2 \\ \text{subject to} & \\ -10x_1 + 20x_2 &\leq 22 \\ 5x_1 + 10x_2 &\leq 49 \\ 0 \leq x_1 &\leq 1 \\ 0 \leq x_2 &\leq 2 \end{aligned}$$

Bài toán có nghiệm tối ưu tại $(1, 1.6)$ với $Z = 5.4$. Do đó, bất kỳ nghiệm nguyên nào trong vùng khả thi này cũng không thể cho giá trị Z lớn hơn 5.4. Nhánh này đã được thăm dò xong.

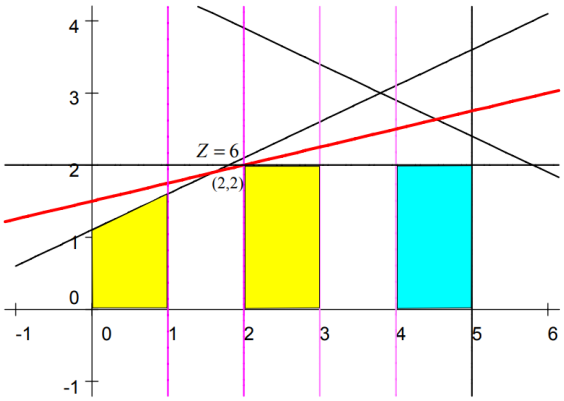


Figure 12: Feasible region

3 Mô hình hóa

3.1 Bài toán mẫu:

Một công ty sản xuất các thanh kim loại có chiều dài 100 đơn vị. Họ nhận được đơn đặt hàng cho các thanh có chiều dài 45, 36, 30 và 15 đơn vị. Nhu cầu tương ứng cho các chiều dài này lần lượt là 7, 5, 8 và 9.

Mục tiêu: Mục tiêu là xác định cách cắt các thanh dài 100 đơn vị để đáp ứng các đơn đặt hàng này trong khi giảm thiểu lãng phí.

Ta có thể thấy bài toán chia vật liệu này có rất nhiều cách giải để chia một thanh kim loại ban đầu thành những thanh có chiều dài theo nhu cầu và số lượng tương ứng. Và đây là các ứng dụng thường trong việc sản xuất hàng hóa vì vậy để đạt được lợi nhuận tối đa thì người ta phải giảm giá sản xuất , tức là giảm đi các chi phí dư thừa từ nguyên vật liệu. Dưới đây là một lời giải theo kiểu suy nghĩ đơn giản:

- Cắt 7 thanh theo 45-55 đơn vị độ dài, cắt tiếp từ thanh 55 thành 30-25.
- Cắt 5 thanh theo dang 36-64 sau đó cắt tiếp thành một thanh 30 và một thanh 34 đơn vị độ dài.
- Cắt tiếp 8 thanh theo dạng 30-70 sau đó chia 70 thành các phần khác theo yêu cầu,
- Cắt tiếp 9 thanh theo dạng 15-85 và phần còn lại có thể chia tiếp cho ra thành các phần khác.

Dễ thấy lời giải trên tuy đã đạt được những đoạn có chiều dài và với số lượng mong muốn nhưng đã sử dụng số lượng lớn nguyên liệu và không tính toán việc sử dụng các nguyên liệu thừa một các đúng đắn.

Trước khi thực hiện tối ưu hóa bài toán trên thì chúng ta cần đưa bài toán về dạng tổng quát để có thể giải quyết.

3.2 Cutting the stock with one stock:

Với bài toán "Cutting the stock" ta giải trong trường hợp này là trường hợp chỉ có một loại nguyên liệu ban đầu. Giả sử nguyên liệu ban đầu có chiều dài là L và có n chiều dài yêu cầu cần chia với chiều dài mỗi mỗi vật là l_i , $i = 1, ..., n$ và số lượng yêu cầu tương ứng là d_i ,$i = 1, ..., n$.

Mỗi thanh đều được cắt theo một dạng và số lượng thanh được cắt theo dạng j kí hiệu là x_j với $j \in \mathcal{J}$, $\mathcal{J}$ là tập hợp tất cả các pattern có thể tạo ra với m là số lượng patterns có thể chia được.

Mỗi dạng pattern j có ương ứng thì số lượng của vật i trong pattern đó là α_{ij}.

Ta có hàm mục tiêu như sau:

$$\sum_{j \in \mathcal{J}} x_j \tag{7}$$

hoặc có thể viết như sau:

$$\sum_{j \in \mathcal{J}} c_j * x_j \tag{8}$$

với c_j là lượng dư thừa của mỗi pattern.

Phương trình trên là phương trình tổng số lượng, phương trình dưới là phương trình tổng số lượng dư thừa.Vì giá giảm giá trị dư thừa sau khi bị cắt trong bài toán hiện tại nó giống với việc giảm thiểu tối đa số lượng thanh nguyên liệu. (Điều đó dễ thấy qua việc tổng chiều dài của các thanh mục đích cần cắt luôn không đổi khi ta giảm được số lượng thanh tức nghĩa giảm được chiều dài tổng nên giảm được phần bỏ đi).

Các điều kiện ràng buộc:

Ràng buộc về tổng số lượng các phần cần cắt ít nhất thỏa được yêu cầu.

$$\sum_{j \in \mathcal{J}} \alpha_{ij} x_j \geq d_i \qquad \forall i = 1, 2, ..., n \tag{9}$$

Tổng chiều dài của các phần trong 1 pattern luôn nhỏ hơn chiều dài trong thanh của các thanh đối với mọi pattern.

$$\sum_{i=1}^{n} \alpha_{ij} l_i \leq L \qquad \forall j \in \mathcal{J} \tag{10}$$

$$\text{Với } x_j, \alpha_{ij} \in \mathbb{Z}$$

Chúng ta cần làm là dựa trên các ràng buộc giảm giá trị của hàm mục tiêu đi.

3.3 Cutting the stock with mutiple stock

3.3.1 Giảm lượng phần cần bỏ:

Giống với việc giảm lượng bị cắt bỏ đi đối với một 1 loại thanh nguyên liệu ban đầu nhưng ban đầu sẽ có thể có nhiều thanh nguyên liệu hơn để lựa chọn nhằm giảm thiểu số lượng phần bị cắt dư không cần thiết.

Các thanh nguyên liệu k sẽ có chiều dài L_k với k=1,2,...,m (m là số loại thanh).

Gọi $P_{j,k}$ là một loại pattern có kí hiệu j được cắt trên thanh nguyên liệu k, $c_{j,k}$ là phần bị dư thừa, và $x_{j,k}$ là số lượng của $P_{j,k}$

Số lượng mỗi một thanh nguyên liệu trong một loại pattern j trong thanh nguyên liệu k kí hiệu là α_{ijk}

$\implies c_{jk} = L_k - \sum_{i=1}^{n} \alpha_{ijk} l_i$

Hàm mục tiêu:

$$\sum c_{jk} * x_{jk} \qquad \forall j,k \tag{11}$$

Các điều kiện:

$$\sum_{\forall jk} \alpha_{ijk} x_{jk} \geq d_i \qquad \forall i = 1,2,...,n \tag{12}$$

$$\sum_{i=1}^{n} \alpha_{ijk} l_i \leq L_k \qquad \forall j,k \tag{13}$$

3.3.2 Giảm giá thành nguyên liệu đầu vào

Bài toán này thay vì sử dụng một chiều dài của thanh đầu vào thì sẽ sử dụng nhiều thanh có giá trị chiều dài đầu vào khác nhau với giá thành khác nhau để chia thành những phần nhỏ hơn và tìm được giá trị giá thành nhỏ nhất có thể sử dụng.
Gọi:

Chiều dài mỗi thanh nguyên liệu s là L_s và có giá trị là c_s. ($s \in S$ với S là tập các thanh nguyên liệu đầu vào)

Chiều dài của mỗi phần cần cắt là l_f và nhu cầu của phần đó là d_f ($f \in F$ với F là tập các phần cần cắt). P là tập hợp các mẫu cắt. x_p : là số lượng mẫu cắt p được sử dụng.
$a_{f,p}$: số lượng phần hoàn thành f trong mẫu cắt p.
$b_{s,p}$: biến nhị phân thể hiện cho mẫu cắt p có sử dụng thanh nguyên liệu s hay không.
Hàm mục tiêu:

$$\sum_{p \in P} \sum_{s \in S} c_x . b_{s,p} . x_p \tag{14}$$

Các ràng buộc:
Mỗi mẫu cắt chỉ được sử dụng một thanh nguyên liệu:

$$\sum_{s \in S} b_{s,p} = 1, \forall p \in P \tag{15}$$

Tổng chiều của các phần trong các thanh không được lớn hơn chiều dài thanh

$$\sum_{f \in F} a_{f,p} . l_f \leq \sum_{s \in S} b_{s,p} . L_s, \forall p \in P \tag{16}$$

Ràng buộc về số lượng của của các thành phần:

$$\sum_{p \in P} a_{f,p} . x_p \geq d_i \tag{17}$$

Các điều kiền kiện của các biến: $b_{s,p} \in 0,1 \forall s \in S, p \in P$; $a_{f,p} \geq 0 \forall f \in F, p \in P$;$x_p \geq 0 \forall p \in P$.
Và vì bài toán không có thứ tự xác định nên ta có thể thêm điều kiện ràng buộc này: $x_{p-1} \geq x_p$ $\forall P, p > 0$ để có thể giảm bớt các trường hợp tạo đối xứng.

4 Giải thuật

4.1 Solve cutting stock problem with 1 stock:

4.1.1 Column generation:

Thuật toán column generation dùng để giải các trường hợp mở rộng. Ta có dạng sau khi đã mô hình hóa của bài toán(Master Problem) :

$$\min \quad \sum_{j \in \mathcal{J}} c_j x_j \tag{18}$$

$$\textit{subject to:} \sum_{j \in \mathcal{J}} \alpha_{i,j} x_j \geq d_i \tag{19}$$

$$\sum_{i=1}^{n} \alpha_{i,j} l_i \leq L \tag{20}$$

$$x_j \geq 0, \quad j \in \mathcal{J} \tag{21}$$

Với c_j là lượng bị dư của pattern j và x_j là số lần lặp lại của pattern j. Tập $\mathcal{J}$ là tập tất cả các pattern có thể có khi mà cắt thanh nguyên thành các phần cần thiết.

Ta có số lượng phần tử của tập $\mathcal{J}$ rất lớn nên thay vì giải một hệ phương trình tuyến tính lớn ta chỉ tập trung vào một vấn đề nhỏ hơn (restricted master problem (RMP)) với $\tilde{\mathcal{J}} \subset \mathcal{J}$, Ta có công thức sau:

$$\min \quad \sum_{j \in \tilde{\mathcal{J}}} c_j x_j \tag{22}$$

$$\textit{subject to:} \sum_{j \in \tilde{\mathcal{J}}} \alpha_{i,j} x_j \geq d_i \quad [\pi] \tag{23}$$

$$\sum_{i=1}^{n} \alpha_{i,j} l_i \leq L \tag{24}$$

$$x_j \geq 0, \quad j \in \tilde{\mathcal{J}} \tag{25}$$

Sử dụng Simplex methods để có được x và giá trị đối ngẫu π.

Và việc được đặt ra là liệu những pattern trong $\mathcal{J}$ đã là tối ưu cho bài toán hay chưa.

$\bar{c_j} = c_j - \pi' \alpha_j \geq 0, \quad \forall j \in \mathcal{J}$, là điều kiện biểu thức khi đươc đã được tối ưu hóa. Vậy khi $\bar{c_j} < 0$ thì việc thêm biến x_j (tức nghĩa là thêm một cái pattern j mới) vào sẽ giúp cải thiện cho mô hình.

Algorithm 1 RMP(Restricted master problem)

Input: RMP với tập hơn ban đầu là $\bar{J}$
Output: $\bar{J}$ chứa tất cả các pattern để tối ưu hàm mục tiêu
Stop=false **while** *!Stop* **do**
  Giải RMP để tìm x và π
  Giải z=$min\{\bar{c_j}|j \in \bar{J}\}$ Tìm ra j* mới và z (**subproblem**)
  if $z < 0$ **then**
    $\bar{J} < -\bar{J} \cup j^* \quad with \quad \bar{c_j}* = z$
  end
  else
    Stop=True
  end
end

Chúng ta sẽ giải quyết việc tìm pattern mới. Sau khi được đối ngẫu từ bài toán ban đầu thì cũng ta sẽ có công thức của phần bỏ đi của pattern p kí hiệu bởi $\bar{c_p}$ như sau:

$$\bar{c_p} = 1 - (\pi_1, \ldots, \pi_n) \begin{pmatrix} \alpha_{1p} \\ \alpha_{2p} \\ \vdots \\ \alpha_{np} \end{pmatrix} \tag{26}$$

với a_{ip} là số lượng của các item i trong pattern p còn $\pi_i, i = 1, ..n$ là những giá trị đối ngẫu tương ứng với a_{ip}.

Ta có bài toán con:(subproblem)

$$z = min \qquad (1 - \sum_{i=1}^{n} \pi_i.k_i) \tag{27}$$

$$subjectto : \sum_{i=1}^{n} \ell_i k_i \leq L \tag{28}$$

$$k_i \in \mathbb{Z}_+ \quad i \in [n] \tag{29}$$

suy ra:

$$z = \qquad 1 - max(\sum_{i=1}^{n} \pi_i.k_i) \tag{30}$$

$$subjectto : \sum_{i=1}^{n} \ell_i k_i \leq L \tag{31}$$

$$k_i \in \mathbb{Z}_+ \quad i \in [n] \tag{32}$$

Vậy bài toán cần giải quyết là:

$$max(\sum_{i=1}^{n} \pi_i.k_i) \tag{33}$$

$$subjectto: \sum_{i=1}^{n} \ell_i k_i \leq L \tag{34}$$

$$k_i \in \mathbb{Z}_+ \quad i \in [n] \tag{35}$$

Đây là Knapsack Problem và vấn đề này là một vấn đề NP có thể giải bằng nhiều cách khác nhau.

Sau ghi giải bài toán này thì chúng ta có được tập các giá trị của $k_1, k_2, ..., k_n$ đấy là số lượng của mỗi mỗi iem tương ứng trong một pattern mới, kiểm tra xem có thỏa điều kiện dừng hay chưa để tiếp tục và thêm vào tập hợp $\bar{\mathcal{J}}$, nếu pattern mới tìm được thỏa điều kiện dừng sẽ dừng vòng lặp vì đã tìm đủ các pattern có thể tối ưu hóa cho việc cắt nhỏ thanh nguyên liệu ban đầu thành các thanh cần thiết.

Sau khi có được tập hợp $\bar{\mathcal{J}}$ chưa tất cả các pattern có thể làm nhỏ đi tổng số lượng. Bây giờ bài toán của chúng ta thay đổi thành:

$$\min \quad \sum_{j \in \bar{\mathcal{J}}} c_j x_j \tag{36}$$

$$subject\ to: \sum_{j \in \bar{\mathcal{J}}} \alpha_{i,j} x_j \geq d_i \tag{37}$$

$$\sum_{i=1}^{n} \alpha_{i,j} l_i \leq L \tag{38}$$

$$x_j \geq 0, \quad j \in \bar{\mathcal{J}} \tag{39}$$

Hình13 là mô tả quá trình thực hiện của giải thuật Column Generation:

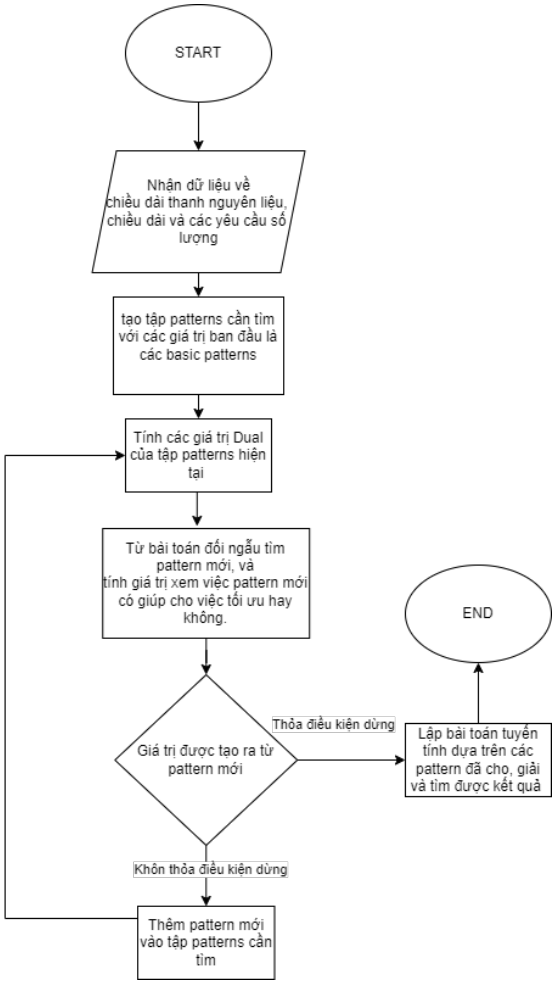


Figure 13: FC cho column generation

4.1.2 Một số giải thuật gần đúng:

Algorithm 2 First Fit Decreasing Algorithm

Data: danh sách các vật bao gồm chiều dài l_i và yêuyêu cầu về số lượng d_i, chiều dài của thanh nguyên liệu L

Result: trả về các cách chia

Sắp xếp lại danh sách theo chiều giảm dần

Khởi tạo Patterns là danh sách của pattern với mỗi pattern bao gồm [danh sách item, rest]

for *với mỗi item trong items* **do**
  for *pattern in patterns* **do**
    if *pattern->rest* $> l_{item}$ **then**
      $count = min(\lfloor pattern- > rest/l_{item} \rfloor, d_{item}$
      update pattern by : $pattern + count.item$ and $pattern- > rest - count.l_{item}$
      $d_{item} = d_{item} - count$
    end
  end
  while do
    $count = min(\lfloor pattern- > rest/l_{item} \rfloor, d_{item}$
    tạo newpattern với lượng count của item sau đó thêm newpattern vào patterns.
    $d_{item} = d_{item} - count$
  end
end

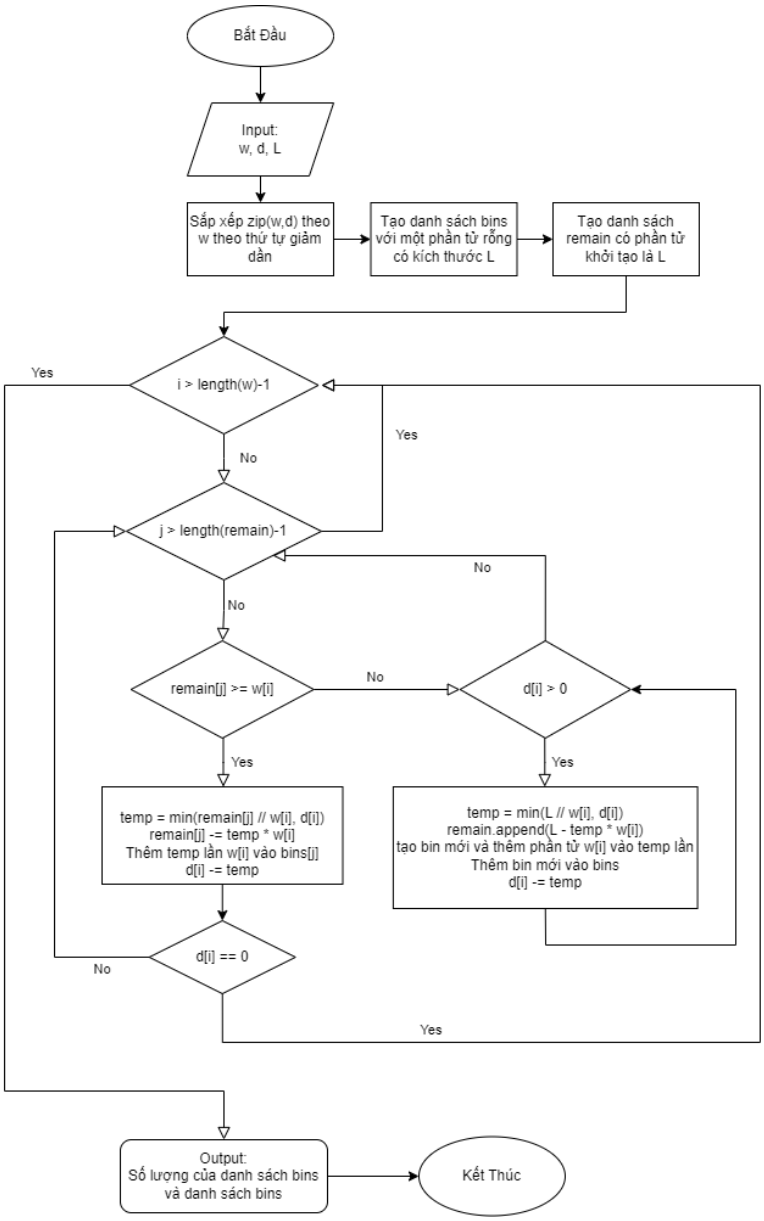


Figure 14: Flow chart cho thuật toán FFD

Algorithm 3 MGH (Modified Greedy Heuristic)

Data: danh sách các vật bao gồm chiều dài l_i và yêu cầu về số lượng d_i, chiều dài của thanh nguyên liệu L
Result: trả về các cách chia

Sắp xếp lại danh sách theo chiều giảm dần
Chia danh sách thành hai danh sách với chiều dài của các item là chẵn và lẻ. **if** *L là chẵn* **then**
  Thì danh sách các item sẽ được ghép nối thành danh sách chẵn trước và danh sách lẻ sau
end
else
  Danh sách item mới bắt đầu bằng danh sách lẻ trước.
end
Khởi tạo Patterns là danh sách của pattern với mỗi pattern bao gồm [danh sách item, rest]

for *với mỗi item trong items* **do**
  for *pattern in patterns* **do**
    if *pattern->rest* $> l_{item}$ **then**
      $count = min(\lfloor pattern- > rest/l_{item} \rfloor, d_{item}$
      update pattern by : $pattern + count.item$ and $pattern- > rest - count.l_{item}$
      $d_{item} = d_{item} - count$
    end
  end
  while do
    $count = min(\lfloor pattern- > rest/l_{item} \rfloor, d_{item}$
    tạo newpattern với lượng count của item sau đó thêm newpattern vào patterns.
    $d_{item} = d_{item} - count$
  end
end

Ngoài ra còn các giải thuật gần đúng khác như BFD (best fit decreasing), WFD(worst fit deacreasing), BPP (bin packing problem) một số biến thể giữa việc kết hợp các giải thuật lại.

4.2 Solve cutting stock problem with mutiple stock:

4.2.1 Giảm lượng bị dư thừa:

Bài toán cắt vật tư một chiều với nhiều kích thước vật liệu thô (One-Dimensional Cutting Stock Problem with Multiple Stock Size 1DMCSP) là mở rộng tự nhiên của bài toán One-Dimensional Cutting Stock Problem (1DCSP) trong đó các thanh vật liệu thô có thể có nhiều kích thước khác nhau. Bài toán 1DMCSP có thể được đặt trưng bằng các dữ liệu sau:

$$(m, M, l = (l_1; ...; l_m), b = (b_1; ...; b_m), L = (L_1; ...; L_M))$$

trong đó: m là số kiểu vật liệu thành phẩm được cắt từ vật liệu thô. Với mỗi kiểu vật liệu thành phẩm j thì l_j là chiều dài và b_j là đơn hàng cho loại vật liệu thành phẩm đó; M là số các loại vật liệu thô sẽ được sử dụng và với mỗi L_j trong L là chiều dài của thanh vật liệu thô được sử dụng. Bài toán đặt ra tìm cách cắt sao cho lượng rác thải ra là ít nhất hay cũng chính là tổng

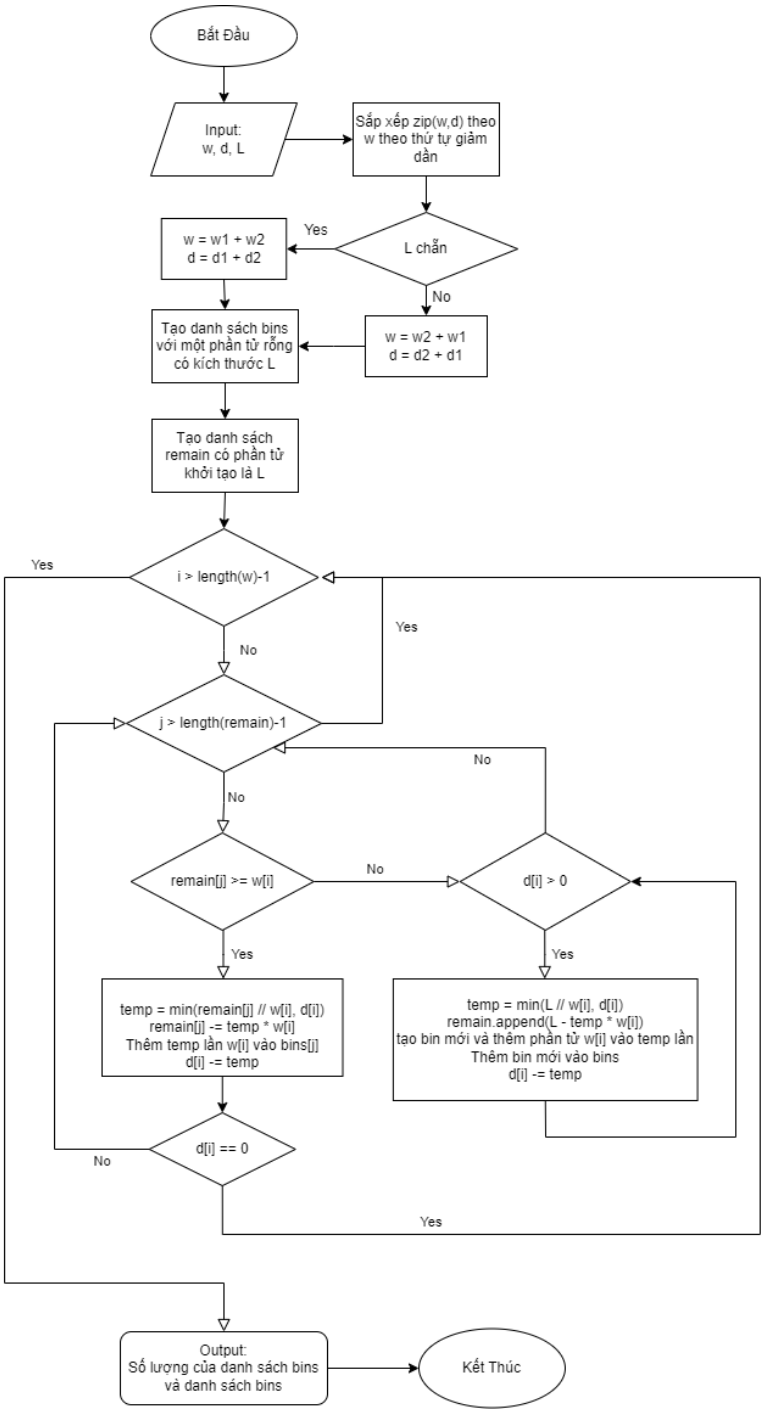


Figure 15: Flow chart cho thuật toán MGH

chiều dài các thanh số lượng thanh vật liệu thô sử dụng là ít nhất mà vẫn đáp ứng được đơn hàng.

Một phương án cắt được biểu diễn bởi một vector cột $A_j = (a_{1j}; ...; a_{mj})$ với $a_{ij} \in \mathbb{N}$, $i = \overline{1, m}$. Mỗi phương án cắt A_j sẽ ứng một loại thanh vật liệu thô có chiều dài nhất định trong L. Tập hợp các phương án cắt sẽ tạo nên tập các cách cắt $A = (A_1, ..., An)$, n ở đây chúng ta không cần xác định một cách cụ thể, vì chúng ta sẽ dụng kỹ thuật Column Generation để giải quyết bài toán này bởi đây là một bài toán quy hoạch tuyến tính cỡ lớn, với n có thể rất lớn. Để giải và mô hình hóa bài toán này ta sẽ tạo ra một vector cột (hoặc hàng để nhìn thấy sự tương ứng trên dưới) C với kích thước $n \times 1$, với n chính là số cột của ma trận A ở trên. Với mỗi cột trong ma trận A hay chính là với mỗi phương án cắt A_j thì tương ứng là phần tử C_j trong C, phần tử C_j này sẽ lưu giá trị chiều dài của thanh vật liệu thô sẽ được cắt bởi cách cắt A_j. Giá trị của C_j chính là giá trị của một trong số các phần tử của L. Ta cũng sẽ gọi tập phương án $x = (x_1; ...; x_n)$, với mỗi x_j trong x là số thanh vật liệu thô có chiều dài C_j sẽ được cắt bởi cách cắt A_j Từ đây ta sẽ mô hình hóa bài toán như sau:

$$\min \quad C^T x \tag{40}$$

$$\textit{subject to:} \quad Ax \geq b \tag{41}$$

$$(l^T A)^T \leq C \tag{42}$$

$$x_j \in \mathbb{N}, \quad j = \overline{1, n} \tag{43}$$

Đến đây thì ta có thể áp dụng các bước giải tương tự như 1DCSP, tuy nhiên có thể do tính relaxaion không được thỏa mãn mà thuật toán chưa được hoàn hảo, vẫn còn chưa được tối ưu nhất ở một số trường hợp cần có sự cải tiến và phát triển thêm, một ý tưởng có thể kết hợp với thuật toán Column Generaton là thuật toán Genetic để đưa ra thuật toán hoàn hảo hơn.

Một số điểm đặt biệt quan trọng và khác với 1DCSP cần chú ý khi giải đó là hệ số hàm mục tiêu, và công thức áp dụng cho việc tạo sinh cột.

Biến mới được thêm vào cần có reducedCost < 0 và nhỏ nhất: $C_N^T - C_B^T B^{-1} N$.

Hay ta cần tìm phép cắt mới A_j sao $\min\{C_j - C_B^T B^{-1} A_j\}$ và từ đây ta xây dựng được bài toán Column Generating Subproblem và bài toán này có thể được giải dễ dàng thông qua lập trình. Đến đây xem như bài toán được giải quyết.

Algorithm 4 Column Generation Algorithm for 1DMCSP

Input: L, l, b are column vectors
Output: Optimal solution for 1DMCSP

$m = length(l)$; $A = []$; $C = []$;
% Khởi tạo ma trận A bằng cách đơn giản nhất gồm m cách cắt A_j cho mỗi chiều dài thanh vật liệu thô trong L, đồng thời khởi tạo C thể hiện phép cắt A_j dùng để cắt cho thanh vật liệu thô có chiều dài bao nhiêu.
for j = *1:numel(L)* **do**
  $At = \text{diag}(\lfloor L(j)/l \rfloor)$; ▷ Temporary variable
  $A = [A\ At]$; ▷ Nối hai ma trận theo chiều ngang
  $Ct = \text{ones}(m, 1) \times L(j)$; ▷ Temporary variable
  $C = [C;\ Ct]$; ▷ Nối hai ma trận theo chiều dọc
end
$nA = \text{size}(A, 2)$;
$lb2 = \text{zeros}(m, 1)$; ▷ Chặn dưới của x khi giải intliprog với ràng buộc chính $A_2 x \leq b_2$
$A_2 = l^T$; $b_2 = L$;
$reducedCost = -\infty$; $reducedCostTolerance = 0$;
while *reducedCost* $<$ *reducedCostTolerance* **do**
  $lb = \text{zeros}(nA, 1)$; ▷ Ma trận cột với nA số 0
  $f = C$; ▷ Hệ số hàm mục tiêu bài toán gốc
  $A = -A$; $b = -b$; ▷ Đổi dấu để khớp với format trong Matlab
  % Ma trận B là ma trận cơ sở được chọn khi giải solve $intlinprog(f, A, b, lb)$
  $f_2 = C^T B^{-1}$; ▷ Hệ số hàm mục tiêu bài toán phụ
  $reducedCostlist = []$; $newpatternlist = []$;
  for j = *1:numel(L)* **do**
    $b_2 = L(j)$;
    % values, reducedCost, pexitflag lần lượt là nghiệm x, giá trị hàm mục tiêu và pexitflag>0 nếu bài toán intlinprog được giải thành công.
    $[values, reducedCost, pexitflag] = solve \quad intlinprog(f2, A2, b2, lb2)$;
    if *pexitflag* $>$ *0* **then**
      $reducedCost = b_2 + reducedCost$; ▷ Continue if this reducedCost is negative
      $reducedCostlist = [reducedCostlist \quad reducedCost]$;
      $newpatternlist = [newpatternlist \quad values]$;
    end
    else
      $reducedCost = +\infty$;
      $reducedCostlist = [reducedCostlist \quad reducedCost]$;
      $newpatternlist = [newpatternlist \quad values]$;
    end
  end
  $reducedCost = \min(reducedCostlist)$;
  $index = \text{find}(reducedCostlist == reducedCost, 1)$; ▷ Một vị trí có giá trị bằng reducedCost
  $newpattern = newpatternlist(:, index)$;
  if *reducedCost* $<$ *reducedCostTolerance* **then**
    $A = [A \quad newpattern]$;
    $nA = nA + 1$;
    $C = [C; L(index)]$;
  end
end
$[values, logsUsed, exitflag] = solve \quad intlinprog(f, A, b, lb)$;
% Xử lý kết quả value nhận được và C ta được kết quả bài toán.

Ngoài thuật toán trên do chúng em tự xây dựng thì chúng em cũng tìm hiểu thêm một thuật toán khác đã được xây dựng trước đó để giải quyết bài toán này. Đó là thuật toán AG-CG là sự kết hợp của Genetic Algorithm và Column Generation. Genetic Algorithm là một thuật toán hết sức thú vị được lấy cảm hứng từ thuyết tiến hóa, tuy những bước thực hiện tưởng chừng như không mang ý nghĩa gì nhưng lại mang lại kết quả hết sức bất ngờ. Thuật toán này được sử dụng để tìm ra các giải pháp tối ưu cho các vấn đề mà thông thường khó để giải quyết bằng các phương pháp truyền thống. Một thuật toán di truyền (genetic algorithm) là một phương pháp tối ưu hóa được lấy cảm hứng từ quá trình lựa chọn tự nhiên. Thuật toán này được sử dụng để tìm ra các giải pháp tối ưu cho các vấn đề mà thông thường khó để giải quyết bằng các phương pháp truyền thống. Thuật toán di truyền bắt đầu bằng việc tạo ra một quần thể ban đầu các giải pháp ứng viên (các cá thể). Mỗi giải pháp đại diện cho một câu trả lời có thể cho vấn đề đó. Sau đó, quá trình lựa chọn sẽ được thực hiện để đánh giá sức khỏe (fitness) của từng cá thể trong quần thể. Sức khỏe là một đo lường về việc giải quyết vấn đề. Các cá thể có sức khỏe cao sẽ được chọn để tiến tới thế hệ tiếp theo. Tiếp theo là giai đoạn lai ghép, trong đó các cặp cá thể được chọn dựa trên sức khỏe của họ và tạo ra con chung bằng cách kết hợp thông tin di truyền của họ. Sau đó, giai đoạn đột biến được thực hiện để đưa vào các thay đổi ngẫu nhiên nhỏ trong thông tin di truyền của con chung, giữ cho quần thể có đa dạng gen di truyền. Cuối cùng, quá trình thay thế sẽ thực hiện việc thay thế thế hệ cũ bằng thế hệ mới. Thế hệ mới bao gồm các con chung tạo ra thông qua lai ghép và đột biến cũng như có thể một số cá thể ở thế hệ trước có gen tốt.

Thuật toán GA-CG chúng em đã tham khảo từ "Tạp chí Tin học và Điều khiển học, T.25, S.3 (2009), 214–230". Dưới đây là những mô tả sơ bộ về thuật toán này ngoài ra cần có những toán tử (hay hàm) để thực hiện thuật toán Gen được mô tả ở bài tạp chí.

Ý tưởng phân tách bài toán 1DMCSP thành các bài toán cơ sở là bài toán $1\text{DCSP}_G(m, L_k, l, b^k)$để có thể áp dụng được phương pháp giải sử dụng kỹ thuật tạo sinh cột của Gilmore và Gomory. Các nghiệm tối ưu của từng bài toán cơ sở sẽ được kết hợp lại để tạo nên nghiệm chấp nhận được cho bài toán 1DMCSP. Việc tìm nghiệm tối ưu cho bài toán 1DMCSP sẽ dựa trên việc gán dãn nhiệm với không gian tìm kiếm là tập các phần hoạch của vector đơn hàng.

Thuật toán GA-CG

Input: m, M, l, L, b, c

Output: Nghiệm tối ưu của bài toán $1DMCSP(m, M, l, b, L, c)$ và các phương án cắt tương ứng với nghiệm tối ưu đó.

Step 0. Khởi tạo quần thể gồm K cá thể $G_0 = \{s_0^1, ..., s_0^K\}$. Việc khởi tạo này có thể hiện được bằng việc tạo ra K phần hoạch khác nhau của b, mỗi phần hoạch sẽ tương ứng với một đối tượng của quần thể. Các cá thể được biểu diễn như sau:

$$s_i^0 = (b_{i1}^0, ..., b_i M^0), b_{ij}^0 \in \mathbb{Z}_+, \sum_{j=1}^{M} b_{ij}^0 = b, i = 1, ..., K. \tag{44}$$

Step 1. Giải các bài toán $1\text{DCSP}_G(m, L_k, l, b_{ik}^t), i = 1, ..., K, k = 1, ..., M, t$ là thứ tự bước lặp (thứ tự của quần thể) bằng phương pháp bạo sinh cột. Tính mức độ thích nghi $f(s_i^t)$ cho từng cá thể của quần thể.

Step 2. Lựa chọn các cha-mẹ trong G_t theo mức độ thích nghị để ghép cặp theo toán tử hôn phối để tạo nên tập các hậu thế G_t' với K phần tử.

Step 3. Tác động toán tử đột biến vào $G_t \cup G_t'$ để nhận được G_t''.

Step 4. Thực hiện tính toán giống như trong **Step 1** cho các cá thể của G_t''.

Step 5. Áp dụng toán tử chọn lọc lên $G_t \cup G_t''$ để chọn ra K cá thể có mức độ thích nghi lớn nhất tạo quần thể mới G_{t+1}.

Step 6. Nếu điều kiện dừng chưa thỏa mãn quay lại **Step 2**. Ngược lại thuật toán dừng và cho nghiệm tối ưu cũng như tập các phương án cắt tương ứng với nghiệm tối ưu.

Để tiện cho việc so sánh và đánh giá giải thuật thì nhóm em có làm thử một thuật toán đơn giản theo hướng của FFD dành cho mutiple stock:

Algorithm 5 Modified First Fit Decreasing Algorithm(MFFDMS) with mutiple stock

Data: danh sách các vật bao gồm chiều dài l_i và yêu cầu về số lượng d_i, chiều dài của tập thanh nguyên liệu có chiều dài L_k

Result: trả về danh sách patterns

Danh sách yêu cầu và chiều dài sẽ được sắp xếp ra ví dụ thanh có l=1 và yêu cầu là 5 thì sẽ được chuyển thành 1,1,1,1,1. (listitems).

Danh sách patterns được tạo từ các pattern trong đó chứa danh sách các item chứa trong pattern đó và một biến để hiển thị đang sử dụng thanh nguyên liệu bao nhiêu.

function: add item into pattern Khoảng trống đủ update -> trả về True

Nếu mà khoảng trống trong pattern không đủ thì tăng lên mỗi lần một thanh lớn hơn nhỏ nhất xem có thể nhét item vào không.

Nếu thêm vào được thì trả về True.

Nếu không thêm vào được trả về False.

for *item in listitems* **do**
  for *pattern into patterns* **do**
    if *add item into pattern* **then**
      break;
    end
  end
  Nếu mà không có thêm vào pattern nào thì thêm một newpattern vào danh sách patterns với chiều dài thanh nguyên liệu là nhỏ nhất có thể.

end

4.2.2 Giảm giá thành:

Trong phần này chúng ta sẽ giới thiệu và mở rộng một mô hình cắt vật liệu thô với nhiều kích thước khác nhau mới. Chúng bao gồm thêm chi phí của từng loại vật liệu thô. Chi phí này sẽ làm thay đổi hàm mục tiêu của mô hình cơ bản. Thay vì giảm thiểu số lượng vật liệu thô bị lãng phí thì mục tiêu là giảm thiểu chi phí của vật liệu thô cần sử dụng. Điều này cũng dẫn đến sự gia tăng đáng kể về tổng số mẫu cắt (patterns) cần xem xét khi giải quyết mô hình.

Cho một danh sách các vật liệu thô có sẵn, chúng ta phải tối ưu hóa và tính toán xem cần cắt bao nhiêu mỗi loại vật liệu thô để đáp ứng nhu cầu của khách hàng với chi phí tối thiểu.

Linear Dual Method thường được sử dụng để giải quyết vấn đề cắt vật liệu thô bằng cách lặp đi lặp lại việc tạo ra các patterns mới để cải thiện giải pháp cho bài toán.

Bilinear Model cũng là một cách tiếp cận khác để giải quyết vấn đề cắt vật liệu thô. Phương pháp này tạo các patterns bằng cách xem xét sự tương tác (bilinear terms) giữa các biến, sẽ dẫn đến tạo ra các patterns phức tạp hơn và tối ưu hóa hơn.

Algorithm 6 dưới đây sử dụng 2 phương pháp tạo patterns trên để giải quyết bài toán cắt vật liệu thô. Đầu tiên là tạo các patterns cơ bản như bài toán dùng Column Generate, sau đó sử dụng Linear Dual Method để tạo thêm patterns. Và cuối cùng là sử dụng Bilinear Model để bổ xung thêm các patterns phức tạp hơn (tổ hợp của các kích thước được yêu cầu) nếu có, nhằm tìm ra giải pháp tối ưu hóa nhất cho bài toán.

Linear Dual Method
Step 1: Calculate the dual prices based on the current set of patterns dual_prices = calculate_dual_prices(finish, patterns) **Step 2:** Solve the subproblem using dual prices Find a new pattern that minimizes the reduced cost new_pattern = solve_knapsack_problem(stocks, dual_prices) **Step 3:** Check the reduced cost of the new pattern reduced_cost = calculate_reduced_cost(new_pattern, dual_prices) **Step 4:** If the reduced cost is negative, return the new pattern if reduced_cost $<$ 0: return new_pattern **Step 5:** If no new pattern with negative reduced cost is found, terminate the process

Bilinear Model
Step 1: Initialize the coefficients for the bilinear model coefficients = initialize_coefficients(finish, patterns) **Step 2:** Define the bilinear optimization problem Objective: Minimize the bilinear objective function bilinear_objective = define_bilinear_objective(coefficients) **Step 3:** Solve the bilinear optimization problem new_pattern = solve_bilinear_problem(stocks, bilinear_objective) **Step 4:** Evaluate the feasibility and efficiency of the new pattern if is_feasible(new_pattern, stocks, finish) and is_efficient(new_pattern, patterns): return new_pattern **Step 5:** If the new pattern is not feasible or efficient, return null

Algorithm 6 Algorithm using pattern generation

Data: danh sách **stocks** gồm chiều dài và giá cả tương ứng của các thanh, danh sách **finish** gồm chiều dài và dố lượng yêu cầu các thanh cần cắt
Result: kết quả chia của thuật toán

patterns = make patterns(*stocks*, *finish*) ▷ khởi tạo các tập patterns khởi đầu
Phase 1: Khởi tạo các patterns sử dụng dual method
new_pattern = generate pattern dual(stocks, finish, patterns)
while *new_pattern is not in patterns* **do**
  append new_pattern to patterns
  new_pattern = generate pattern dual(stocks, finish, patterns)
end
$(x, cost)$ = cut patterns(stocks, finish, patterns)
Phase 2: Generate patterns using bilinear method
new_pattern = generate pattern bilinear(stocks, finish, patterns)
while *new_pattern is not in patterns* **do**
  append new_pattern to patterns
  new_pattern = generate pattern bilinear(stocks, finish, patterns)
end
$(x, cost)$ = cut_patterns(stocks, finish, patterns)
non_zero_indices = empty list ▷ Lấy chỉ số của các patterns khác 0
for *với mỗi (index, value) in enumerate(x)* **do**
  if *value* $>$ *0* **then**
    append index to non_zero_indices
  end
end

5 Phân tích

5.1 So sánh thuật toán column generation với các thuật toán gần đúng:

Đối với các thuật toán gần đúng thì dễ hiện thực và sẽ cho ra kết quả không sai lệch quá nhiều so với các testcase nhỏ, nhưng khi vấn đề được scale càng lớn thì nó sẽ càng bộc lộ các khiếm điểm là không tối ưu hơn nhiều khi so với thuật toán column generation.

Đối với thuật toán column generation, thì chúng ta luôn bắt đầu bằng một số ràng buộc cơ bản và tiếp tục tạo ra các ràng buộc mới. Vấn đề khó giải quyết nhất là sử dụng hệ số nào để xác định được các ràng buộc mới có thể làm tối ưu cho hàm mục tiêu. Và sau khi giải quyết được vấn đề tạo ra những ràng buộc mới (pattern mới) thì mọi thứ đều có thể đưa về bài toán tuyến tính nguyên và giải quyết nhanh hơn so với việc liệt kệ toàn bộ pattern. Phía dưới à hình **??**bao gồm một thông số sau khi nhóm chúng em chạy thử thuật toán FFD,MGH, Column generation: Một số lưu ý:

- Thông số time được đo bằng hàm trong thư viện timeit, và thời gian chỉ mang tính chất tham khảo có thể không phải là thời gian tối ưu nhất cho mỗi thuật toán.
- Đối với thông số waste thì sẽ được tính bằng công thức
  $watse$ = số lượng sử dụng * chiều dài - tổng chiều dài của các phần cần cắt

results.csv

	A	B	C	D	E	F	G
1	Test Case	Time_FFD	Waste_FFD	Time_MGH	Waste_M...	Time_Column_generation	Waste_Column_generation
2	data0.txt	0.00044450000859797	130	3.900006413459778e-05	130	0.3189925999613479	130.0
3	data1.txt	4.850002005696297e-05	7	3.679993096739054e-05	7	0.06927440001163632	7.0
4	data2.txt	0.0006490999367088079	11481	0.00031670008320361376	21481	0.06969440006650984	12481.0
5	data3.txt	2.9499991796910763e-05	0	1.950003206729889e-05	0	0.11789900006260723	0.0
6	data4.txt	4.4199987314641476e-05	64	1.910002902150154e-05	3	0.09348539996426553	64.0
7	data5.txt	6.260001100599766e-05	65	5.239993333816528e-05	65	0.09568609995767474	65.0
8	data6.txt	0.000452299951575696...	10000	0.0004419999895617366	10000	0.15407499996945262	10000.0
9	data7.txt	0.0004989999579265714	233225	0.0004477999173104763	233225	0.05126779992133379	233225.0
10	data8.txt	0.000425099977292120...	29635	0.00042699999175965786	44635	0.18839220004156232	24635.0
11	data9.txt	0.0005644999910145998	178715	0.0006121000042185187	186715	0.13666930003091693	98715.0
12	data10.txt	0.0003293999470770359	5070	0.0003538999008014798	5070	0.1345606999238953	53070.0

Figure 16: Một số thông số

Khi nhìn vào hình thì ta có thấy có hai giá trị mà cột waste của thuật toán column generation lớn hơn rất nhiều với hai thuật toán kia. Dây có thể là kết quả của việc đặt điều kiện là số lượng tạo ra $\geq$ số lượng yêu cầu và việc này cũng có thể xảy ra trong một số trường hợp đặcc biệt khác như trường hợp data10.txt nó diễn ra ở pattern mà các thanh cần cắt nó nhỏ hơn rất nhiều so với thanh nguyên liệu. Đối với các testcase nhỏ thì MGH, FFD đều chạy khá là chuẩn xác và độ chính xác của nó khi các thông số được đặt là những số lớn dần lên và những trường hợp này ta có thể sư dụng column generation (có thể nó rơi vào trường hợp giống data10.txt nhưng điều đó khá khó xảy ra trong thực tế).

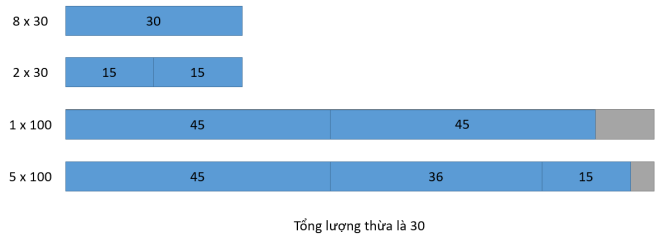
Đối với hai testcase data3.txt và data4.txt đây là tập dữ liệu trên wikipedia chỉ ra rằng thuật toán FFD có thể có thiếu sót.

5.2 Kết quả Algorithm 4 Column Generaton Algorithm for 1DMCSP

Chúng ta sẽ thử so sánh kết quả của Algortihm4 với thuật toán MFFDMS chúng em sửa đổi từ FFD để phù hợp với bài toán nhiều thanh nguyên liệu đầu vào qua các input đơn giản để cho thấy sự tối ưu của Algorithm4.
Input1:
logLength = [30; 70; 100];
lengthlist = [45; 36; 30; 15];
quantity = [7; 5; 8; 9];
Input2:
logLength = [70; 100];
lengthlist = [45; 36; 30; 15];
quantity = [7; 5; 8; 9];
Input3:
logLength = [100; 70; 50; 60; 25];
lengthlist = [45; 36; 30; 15; 17];
quantity = [7; 5; 8; 9; 4];

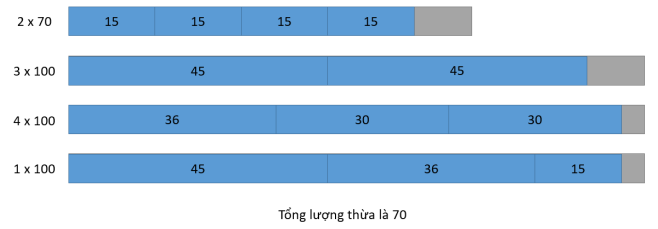
MFFDMS:
output1:
10 thanh 100
waste=100
output2:
10 thanh 100
waste=100
output3:
10 thanh 10 và 1 thanh 50
waste=112

Algorithm4:
Output1:
Optimal solution uses 10 logs with length is 30
Optimal solution uses 6 logs with length is 100
Cut 8 logs 30 with pattern
    1 cut(s) of length 30
    Waste of this pattern is 0
Cut 2 logs 30 with pattern
    2 cut(s) of length 15
    Waste of this pattern is 0
Cut 1 logs 100 with pattern
    2 cut(s) of length 45
    Waste of this pattern is 10
Cut 5 logs 100 with pattern
    1 cut(s) of length 45
    1 cut(s) of length 36
    1 cut(s) of length 15
    Waste of this pattern is 4
Total waste in this problem is 30.



Output2:

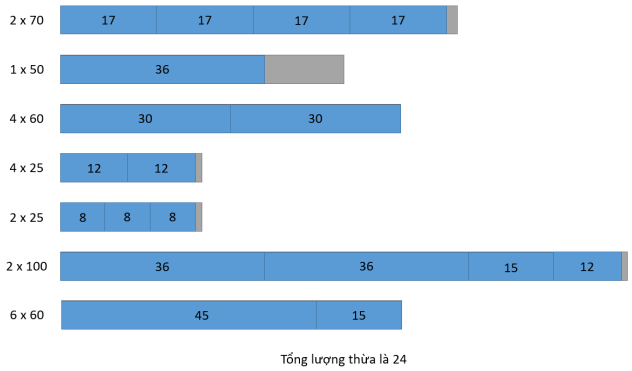
Optimal solution uses 2 logs with length is 70
Optimal solution uses 8 logs with length is 100
Cut 2 logs 70 with pattern
    4 cut(s) of length 15
    Waste of this pattern is 10
Cut 3 logs 100 with pattern
    2 cut(s) of length 45
    Waste of this pattern is 10
Cut 4 logs 100 with pattern
    1 cut(s) of length 36
    2 cut(s) of length 30
    Waste of this pattern is 4
Cut 1 logs 100 with pattern
    1 cut(s) of length 45
    1 cut(s) of length 36
    1 cut(s) of length 15
    Waste of this pattern is 4
Total waste in this problem is 70.



Output3:

Optimal solution uses 2 logs with length is 100
Optimal solution uses 1 logs with length is 70
Optimal solution uses 1 logs with length is 50
Optimal solution uses 11 logs with length is 60
Optimal solution uses 6 logs with length is 25
Cut 1 logs 70 with pattern
    4 cut(s) of length 17
    Waste of this pattern is 2
Cut 1 logs 50 with pattern
    1 cut(s) of length 36

Waste of this pattern is 14
Cut 4 logs 60 with pattern
2 cut(s) of length 30
Waste of this pattern is 0
Cut 4 logs 25 with pattern
2 cut(s) of length 12
Waste of this pattern is 1
Cut 2 logs 25 with pattern
3 cut(s) of length 8
Waste of this pattern is 1
Cut 2 logs 100 with pattern
2 cut(s) of length 36
1 cut(s) of length 15
1 cut(s) of length 12
Waste of this pattern is 1
Cut 7 logs 60 with patter
1 cut(s) of length 45
1 cut(s) of length 15
Waste of this pattern is 0
Total waste in this problem is 24.



Output3 true:
Optimal solution uses 10 logs with length is 100
Optimal solution uses 2 logs with length is 60
Cut 1 logs 100 with pattern
8 cut(s) of length 12
Waste of this pattern is 4
Cut 2 logs 100 with pattern
2 cut(s) of length 36
1 cut(s) of length 15
1 cut(s) of length 12
Waste of this pattern is 1
Cut 1 logs 100 with pattern
3 cut(s) of length 30
1 cut(s) of length 8

Waste of this pattern is 2
Cut 3 logs 100 with pattern
2 cut(s) of length 45
1 cut(s) of length 8
Waste of this pattern is 2
Cut 2 logs 100 with pattern
2 cut(s) of length 30
1 cut(s) of length 15
1 cut(s) of length 17
1 cut(s) of length 8
Waste of this pattern is 0
Cut 1 logs 60 with pattern
4 cut(s) of length 15
Waste of this pattern is 0
Cut 1 logs 60 with pattern
1 cut(s) of length 45
1 cut(s) of length 15
Waste of this pattern is 0
Cut 1 logs 100 with pattern
1 cut(s) of length 36
1 cut(s) of length 30
2 cut(s) of length 17
Waste of this pattern is 0
Total waste in this problem is 14.

Dễ thấy algorithm4 cho kết quả tốt hơn nhiều so với thuật toán đơn giản MFFDMS. Với thuật toán này thì mang lại kết quả tương đối chính xác. Nhiều trường hợp trả về kết quả chính xác và nếu chưa chính xác thì sai số cũng khá nhỏ có thể chấp nhận được. Một ưu điểm của thuật toán này là khá nhanh chỉ mất vài gây có thể cho kết quả.

5.3 So sánh thuật toán GA-CG và Algorithm 4

Dựa trên thuật toán sẵn có GA-CG nhóm chúng em đã thực hiện thuật toán trên Matlab và nhận thấy dường Algorithm 4 vượt trội hơn lẫn cả về độ chính xác và thời gian. Tuy nhiên nếu thời gian cho phép đủ lâu thì thuật toán GA-CG có thể tiến đến độ chính xác cao hơn Algorithm 4. Dưới đây là một số kết quả của GA-CG:

Input1:
logLength = [100; 70];
lengthlist = [45; 36; 30; 15];
quantity = [7; 5; 8; 9];
K = 50; % kích thước của quần thể
n_generation = 10; % số thế hệ

Output1:
Total waste in this problem is 70.
Uses 8 logs with length is 100
Cut 3 logs 100 with pattern
2 cut(s) of length 45
Waste of this pattern is 10
Cut 4 logs 100 with pattern
1 cut(s) of length 36

2 cut(s) of length 30
Waste of this pattern is 4
Cut 1 logs 100 with pattern
1 cut(s) of length 45
1 cut(s) of length 36
1 cut(s) of length 15
Waste of this pattern is 4
Uses 2 logs with length is 70
Cut 2 logs 70 with pattern
4 cut(s) of length 15
Waste of this pattern is 10

Input2:

logLength = [100; 70; 30];
lengthlist = [45; 36; 30; 15];
quantity = [7; 5; 8; 9];
K = 50; % kích thước của quần thể
n_generation = 10; % số thế hệ

Output2:

Total waste in this problem is 50.
Uses 5 logs with length is 100
Cut 3 logs 100 with pattern
2 cut(s) of length 45
Waste of this pattern is 10
Cut 1 logs 100 with pattern
1 cut(s) of length 36
2 cut(s) of length 30
Waste of this pattern is 4
Cut 1 logs 100 with pattern
1 cut(s) of length 45
1 cut(s) of length 36
1 cut(s) of length 15
Waste of this pattern is 4
Uses 3 logs with length is 70
Cut 3 logs 70 with pattern
1 cut(s) of length 36
1 cut(s) of length 30
Waste of this pattern is 4
Uses 7 logs with length is 30
Cut 3 logs 30 with pattern
1 cut(s) of length 30
Waste of this pattern is 0
Cut 4 logs 30 with pattern
2 cut(s) of length 15
Waste of this pattern is 0

Với kết quả trên thì thuật toán GA-CG chạy đúng với Input1 chỉ với K = 50 và n_generation = 10 nên thời gian khá ngắn nhưng kết quả lại chính xác. Với Input2 phức tạp hơn thì thuật toán cho kết quả chưa được chính xác tuy nhiên chúng ta có thể tăng K và n_generation lên để đạt được kết quả tốt hơn và hiển nhiên thời gian vẫn khá lâu.

Với các bài toán nhỏ thì về thời gian thuật toán GA-CG có chút lợi thế hơn khi có tốc độ vẫn chấp nhận được và kết quả cũng sẽ tiến dần về chính xác. Tuy nhiên với các bài toán lớn thì GA-CG chạy rất lâu để dẫn đến kết quả chính xác và dẫn đến kết quả cũng sai số nhiều hơn so với Algorithm 4. Thậm chí qua quá trình chạy thử một vài trường hợp như Input3 thì chúng em nhận thấy khi Algorithm 4 cho kết quả chưa chính xác thì dường như kết quả GA-CG cũng rất khó đạt được kết quả chính xác và nếu không đủ kiên nhẫn thì kết quả sẽ còn tệ hơn rất nhiều so với Algorithm 4.

Về bản chất thì thuật toán Gen chắc chắc sẽ cho được kết quả tốt nhất khi thời gian đủ lâu, vậy nên việc kết hợp Column Generation và Genetic Algorithm để tạo nên GA-CG là một ý tưởng rất hay và có thể dẫn đến được kết quả chính xác. Những nhận định về thuật toán GA-CG có thể sai khi code chúng em thực hiện chưa đủ tốt về thuật toán Gen, cũng như sự hạn chế về thời gian nghiên cứu, thời gian sử dụng phần mềm Matlab Online không cho chép chạy thử thuật toán GA-CG quá lâu. Tuy nhiên với sự trãi nghiệm của chúng em thì sự nổi trội của Algorithm 4 vẫn rất tốt, tìm năng phát triển của Algorithm 4 vẫn rất cao khi được nghiên cứu sâu hơn về những điểm gây ra sai sót của thuật toán, hoặc có thể tích hợp với Genetic Algorithm để tạo thuật toán tương tự như GA-CG nhưng kết quả sẽ tốt hơn.

5.4 So sánh với việc sử dụng Solver trên Microsoft Excel:

Đối với việc sử dụng Solver trong Microsoft Excel thì dễ dàng tiếp cận và sử dụng với mọi người có trình độ công nghệ thông tin cơ bản, tất cả đều chỉ cần sử dụng qua các hàm cơ bản và việc thiết lập hàm mục tiêu cũng khá là dơn giản. Nhược điểm của việc sử dụng solver là nó không có thể chạy được nhiều case trong thời gian ngắn vì đối với solver thì phải tạo đầy đủ các pattern theo chiều dài của thanh nguyên liệu và các thanh cần nên việc đó tốn nhiều thời gian.

stock size	100	70				
required size	45	36	30	15		
required number	7	5	8	9		
Pattern	45	36	30	15	số lượng	Waste
100	2	0	0	0	3	30
	1	1	0	1	1	4
	1	0	1	1	0	0
	0	2	0	1	0	0
	0	1	2	0	2	8
	0	1	1	2	0	0
	0	1	0	4	1	4
	0	0	3	0	0	0
	0	0	2	2	0	0
	0	0	1	4	1	10
	0	0	0	6	0	0
70	1	0	0	1	0	0
	0	1	1	0	1	4
	0	1	0	2	0	0
	0	0	2	0	1	10
	0	0	1	2	0	0
	0	0	0	4	0	0
	7	5	8	9		
					Tổng số lượng sử dụng	Tổng phần dư thừa
	7	5	8	9	10	70
	Số lượng thanh 70		2			
	Số lượng thanh 100		8			

Figure 17: Kết quả chỉ bao gồm tất cả các pattern

	stock size	100	70				
	required size	45	36	30	15		
	required number	7	5	8	9		
	Pattern	45	36	30	15	số lượng	Waste
	100	2	0	0	0	3	30
		1	1	0	1	1	4
		0	1	2	0	2	8
		0	1	0	4	1	4
		0	0	1	4	1	10
	70	0	1	1	0	1	4
		0	0	2	0	1	10
		7	5	8	9		
						Tổng số lượng sử dụng	Tổng phần dư thừa
		7	5	8	9	10	70
		Số lượng thanh 70		2			
		Số lượng thanh 100		8			

Figure 18: Kết quả bao gồm các pattern đã sử dụng

Sử dụng solver của Excel để giải quyết vấn đề của input 2. Hình 17 cho thấy tất cả các pattern, hình 18 chỉ cho thấy những pattern đã sử dụng. So với các pattern từ code phía trên thì sẽ có những patttern khác nhưng việc đó có thể phụ thuộc vào việc tạo ra các pattern của đoạn code, nhưng tổng lượng dư thừa nó vẫn được tối ưu như nhau. Việc sử dụng solver nó giống với việc sử dụng thuật toán vét cạn nên nhóm chúng em không hiện thực thuật toán này, khi đã có đủ các trường hợp sẽ đưa về phương trình đường thẳng và tối ưu số lượng.

6 Case study:

Áp dụng vấn đề "Cutting Stock" vào trong việc cắt gỗ. Đây là một vấn đề khá cổ điển việc tối ưu các đoạn gỗ có thể vừa tối ưu dược chi phí vừa giúp bảo vệ môi trường.

Và trong ứng dụng cho việc cắt gỗ thì nó phù với bài toán cutting stock with mutiple stock và chỉ cần giảm lượng dư thừa vì đa số gỗ có chiều dài không quá khá biệt nhau được tính giá trị theo đơn vị thể tích nên chúng ta không cần quan tâm đến giá trị của các thanh gỗ và chỉ cần tối ưu lượng cắt thừa.

Trong bài toán thực tế này chúng ta sẽ thực hiện chia các khối gỗ có kích thước là 100mm x 150mm x L_k (mm) và đây là bài toán ở Thái Lan và được tính theo đơn vị là Bath/$foot^3$ nên là sẽ đổi về đơn vị VND/m^3 để dễ cho việc tính toán và dễ hiểu (quy đổi theo tỉ giá ngày 8/11/2024, 1 Bath=712.89 VND, và $m^3 = 35,315 foot^3$). Vậy ta có tỉ giá khoảng 6,671,563 VND/m^3.

Số thức tự	Chiều dài(mm)
1	4800
2	5400
3	6000

Table 2: Bảng chiều dài của thanh nguyên liệu

Số thứ tự	Chiều dài	Số lượng yêu cầu
1	2316	789
2	2200	254
3	1650	468
4	1460	554
5	1140	4686
6	1120	1179
7	980	969

Table 3: Bảng chiều dài thanh cần cắt

	MFFDMS	Algortihm4	Solver in Excel
Waste	504960(mm)	205296(mm)	232896(mm)
Thể tích giảm được (m^3)	7.5744	3.07944	3.49344
Số tiền giảm được (triệu VND)	50.53	20.54	23.305

Table 4: Bảng so sánh

Kết quả của Solver in Excel nhóm em trích từ bảng kết quả sau đây trong bài báo study case of wooden pallet industry và sự tính waste của nhóm em bên cạnh.

Table 2. Optimal cutting plan generated by the developed model

Stock length (mm)	Required stock (piece)	No. of item to be cut with the length of (mm)						
		2316	2200	1650	1460	1140	1120	980
4800	394	2						
	1	1				2		
	1					1	3	
5400	107			2			1	1
	304					3		2
6000	254		1	1		1		1
	554				1	3	1	
	371					5		
	103						5	
Number of each item		789	254	468	554	4686	1179	969

waste
66192
204
300
0
6080
7620
0
111300
41200
232896

Figure 19: Bảng trích xuất từ bài báo

Đối với thuật toán MFFDMS cũng đã là một thuật toán đơn tối ưu hóa đơn giản nhưng có thể thấy nó có lượng waste cao hơn gấp đôi so với cả hai kết quả kia. Có thể chủ đề chính của bài báo chỉ nghiên cứu đến sự tác động thực tế của việc áp dụng vấn đề Cutting Stock Problem vào trong công nghiệp nên chỉ cần nêu ra những điều đó và không quá quan tâm đến việc tối ưu tối đa hay chưa.

Có thể thấy việc giảm thiểu tối đa các lượng dư thừa là rất cần thiết vì nó giúp giảm các chi phí phế phẩm và cũng có thể góp phần ảnh hưởng đế các chỉ số khác. Như về cây cối thì khi được scale thời gian thành năm, và số lượng lớn thì nó sẽ ảnh hưởng rất nhiều đến số lượng cây phải khai thác nó sẽ tác động đến các chỉ số về môi trường.

Và thực tế sẽ có nhiều các chi phí khác trong quá trình làm việc nên ngoài tối ưu nguyên vật liệu sử dụng họ phải tính toán nhiều thứ khác như khấu hao, chi phí nhân công, công nghệ kĩ thuật,.. để đạt được hiệu qua tối ưu.

7 Kết luận

Mục đích chính của việc nghiên cứu các thuật toán này là để giải quyết vấn đề cắt vật liệu thô như một bài toán quy hoạch tuyến tính nguyên.

Do số lượng mẫu cắt trong các ứng dụng thực tế khá lớn nên chúng em đã sử dụng phương pháp Column Generation để hiện thực lời giải cho vấn đề. Ưu điểm chính của thuật toán này là không cần liệt kê tất cả các khả năng. Thay vào đó, bài toán được hình thành dưới dạng bài toán RMP với ít biến nhất có thể, và các biến mới được thêm vào khi cần thiết, tương tự như phương pháp đơn hình. Và nếu có thể hình thành được công thức, việc sử dụng thuật toán Column Generation có thể tiết kiệm thời gian đáng kể. Ngoài ra, nghiên cứu này cũng giới thiệu thêm hai giải thuật cũng khá hiệu quả là First Fit Decreasing và Modified Greedy Heuristic.

Một số hình ảnh so sánh giữa các thuật toán cũng được đưa vào để nhìn nhận một cách trực quan hơn về thời gian chạy cũng như kết quả chia giữa chúng có thật sự tối ưu không. Tuy nhiên, vẫn còn một số hạn chế trong việc nghiên cứu như khi chạy các giải thuật, nếu gặp dữ liệu đầu vào quá lớn sẽ làm cho chương trình chạy lâu và quá tải, điều này khiến cho máy khó phân tích hết được.

Ngay hiện tại và trong tương lai, các mẫu cắt được tạo ra trên máy cắt để giảm thiểu số lượng vật liệu dư thừa do nhu cầu của khách hàng mà mỗi công ty nhận được trong thời gian có thể gấp rút. Vì việc tạo các mẫu cắt theo cách thủ công tốn khá nhiều thời gian và có thể không tối ưu. Vì thế, chúng ta nên cần nghiên cứu để có thể giảm thiểu hết mức hoặc loại bỏ những hạn chế trong việc sử dụng các thuật toán và cần hoàn thiện thêm nhiều về các giải pháp được chọn để nghiên cứu về vấn đề cắt vật liệu thô.

8 Tài liệu tham khảo

References

[1] Tạp chí Tin học và Điều khiển học, T.25, S.3 (2009), 214-230

[2] Statistics How To,"Lasso Regression: Simple Definition", *https://www.statisticshowto.com/lasso-regression/*

[3] Proclus Academy,"Robust Scaling: Why and How to Use It to Handle Outliers", *https://proclusacademy.com/blog/robust-scaler-outliers/*, 2022

[4] GradPad,"Interpreting the extra sum-of-squares F test", *https://www.graphpad.com/guides/prism/latest/curve-fitting/reg_interpreting_comparison_of_mod_2.htm*

[5] Sergiy Buntenko,"Column Generation for the Cutting Stock Problem", *https://www.youtube.com/watch?si=TD-8HePipFfaY1YG&v=O918V86Grhc&feature=youtu.be*

[6] "Extra Material: Cutting Stock, MO-BOOK: Hands-On Mathematical Optimization with AMPL in Python", *https://ampl.com/mo-book/notebooks/05/cutting-stock.html*

[7] Haessler, R. W., & Sweeney, P. E. (1991). "Cutting stock problems and solution procedures". European Journal of Operational Research, 54(2), 141-150.

[8] Giovanni Righini,"Column Generation", *http://homes.di.unimi.it/righini/Didattica/ComplementiRicercaOperativa/MaterialeCRO/CG.pdf*

[9] Gilmore, P.C., and Gomory, R.E. (1961), "A linear programming approach to the cutting stock problem", Operations Research 9, 848-859.

[10] Wascher, G., Haubner, H., Schumann, H., "An improved typology of cutting and packing problems". Eur. J. Oper. Res. 183, 1109–1130 (2007)

[11] P Wattanasiriseth and A Krairit 2019 IOP Conf. Ser . "An Application of Cutting-Stock Problem in Green Manufacturing: A Case Study of Wooden Pallet Industry", https://iopscience.iop.org/article/10.1088/1757-899X/530/1/012005/pdf

[12] Bin packing and cutting stock problem. https://scipbook.readthedocs.io/en/latest/bpp.html

code latex:Link