

# Optymalizacja wykorzystania materiału w procesie rozkroju rur

Jakub Pelczar

22 stycznia 2017  
v0.1.1

## Spis treści

<b>1</b>	<b>Wstęp</b>	<b>2</b>
<b>2</b>	<b>Knapsack Problem - Problem plecakowy</b>	<b>3</b>
2.1	Różnorodność problemu plecakowego . . . . .	3
2.2	Możliwe rozwiązania . . . . .	4
<b>3</b>	<b>Cutting Stock Problem - Problem optymalnego rozkroju</b>	<b>5</b>
<b>4</b>	<b>Metoda "Brutal Force"</b>	<b>6</b>
4.1	Algorytm wyjściowy . . . . .	6
4.2	Rozszerzenie o szerokość cięcia . . . . .	7
4.3	Rozszerzenie o wiele długości bazowych . . . . .	7
4.4	Rozszerzenie o cenę materiału wsadowego . . . . .	7
4.5	Przykład . . . . .	8
4.6	Podsumowanie . . . . .	10
<b>5</b>	<b>Metoda "Delayed Column Generation"</b>	<b>11</b>
<b>6</b>	<b>Wyniki</b>	<b>14</b>
6.1	Porównanie . . . . .	14
6.2	Wnioski . . . . .	14
6.3	Podsumowanie . . . . .	14
<b>7</b>	<b>Opis implementacji</b>	<b>15</b>
7.1	Architektura . . . . .	15
7.2	Java . . . . .	15
7.3	Kotlin . . . . .	15
7.4	JavaFX . . . . .	15
<b>8</b>	<b>Zakończenie</b>	<b>16</b>

## 1 Wstep

## 2 Knapsack Problem - Problem plecakowy

Problem plecakowy jest zagadnieniem optymalizacyjnym. Problem ten swoją nazwę wziął z analogii do rzeczywistego problemu pakowania plecaka. Rozwiązując ten problem zarówno w praktyce jak i teorii trzeba zachować reguły określające ładowność plecaka dotyczące objętości i nośności plecaka. Knapsack Problem zaczął być intensywnie badany po pionierskiej pracy Dantzig [1] w późnych latach 50 XX wieku. Znalazł on natychmiast zastosowanie w przemyśle oraz w zarządzaniu finansami. Z teoretycznego punktu widzenia, problem plecakowy często występuje jako relaksacja różnorodnych problemów programowania całkowitego [3].

### 2.1 Różnorodność problemu plecakowego

Wszystkie elementy z rodziny tego problemu wymagają pewnego zestawu elementów które mogą zostać wybrane w taki sposób że zysk zostanie zmaksymalizowany, a pojemność plecaka lub plecaków nie zostanie przekroczona. Wszystkie typy problemu należą do rodziny problemów  $NP$  – *trudnych* co oznacza, że raczej nispotymane jest rozwiązanie problemu z użyciem algorytmów wielomianowych. Możliwe są różne wariaty problemu zależna od rozmieszczenia elementów oraz plecaków:

- *Problem plecakowy 0-1* - każdy element może być wybrany tylko raz. Problem polega na wyborze  $n$  elementów dla których suma profitów  $p_j$  jest największa, bez konieczności osiągnięcia całkowitej pojemności  $c$ . Może być sformułowany jako problem maksymalizacji:

$$\begin{aligned} \text{maksymalizacja} \quad & \sum_{j=1}^n p_j x_j, \\ \text{w odniesieniu do} \quad & \sum_{j=1}^n w_j x_j \leq c, \\ & x_j \in \{0, 1\}, \quad j = 1, \dots, n \end{aligned} \tag{2.1}$$

- *Ograniczony problem plecakowy* - każdy element może być wybrany ograniczoną ilość razy.
- *Problem plecakowy wielokrotnego wyboru* - elementy powinny być wybierane z klas rozłącznych.
- *Wielokrotny problem plecakowy* - wiele plecaków jest wypełnianych równocześnie.
- *Wielokrotnie ograniczony problem plecakowy* - najbardziej ogólny typ który jest problemem programowania całkowitego z dodatnimi współczynnikami.

## 2.2 Możliwe rozwiązania

Dopóki problem plecakowy należy do problemów  $NP$ –*trudnych* nie jest znane inne dokładne rozwiązanie niż wyliczenie przestrzeni rozwiązań. Użycie poniższych technik może ograniczyć pracochłonność otrzymania rozwiązania:

- *Metoda podziału i ograniczeń* - pełna enumeracja rozwiązań, ale ograniczenia są użyte do znalezienia węzłów które nie mogą doprowadzić do poprawy rozwiązania. Metoda ta często jest stosowana do problemu plecakowego od momentu gdy Kolesar [2] zaprezentował pierwszy algorytm w 1967 roku.
- *Programowanie dynamiczne* - może być traktowane jako enumeracja wszerek z pewnymi zasadami dominacji. Czasem testy brzegowe są dodawane do algorytmu programowania dynamicznego, wtedy algorytm ten staje się "zaawansowaną" formą metody podziału i ograniczeń.
- *Przestrzeń stanów relaksacji* - jest to relaksacja metody programowania dynamicznego w której współczynniki są skalowane przez ustaloną wartość. Dzięki tej metodzie zmniejsza się czas oraz złożoność algorytmu, ale rozwiązanie traci optymalność. Algorytm ten jest często wykorzystywany jako wydajny algorytm aproksymacji problemu plecakowego.
- *Przetwarzanie wstępne* - pewna liczba zmiennych zostaje ustalona jako wartość optymalna, używając testów brzegowych do wykluczenia pewnych wartości z rozwiązania.

### 3 Cutting Stock Problem - Problem optymalnego rozkroju

## 4 Metoda "Brutal Force"

### 4.1 Algorytm wyjściowy

Metoda ta opiera się zarówno na intuicji jak i na rozwiązaniu zaproponowanym przez Dantzig dla problemu plecakowego [1]. Jest to metoda która w prosty sposób - nie używając złożonych modeli matematycznych, pozwala osiągnąć optymalny rozkrój materiału.

Pierwszym krokiem jest posortowanie elementów wyjściowych malejąco względem ich długości  $l_1 \geq l_2 \geq \dots \geq l_m$  i umieszczenie w ten sposób w kolejce.

Drugim krokiem jest pobranie pierwszego elementu z kolejki i sprawdzenie, jak wiele razy jego długość zawiera się w długości elementu bazowego. Obliczone zostaje ile materiału pozostało w elemencie bazowym po docięciu najdłuższych elementów. Następnie pobierany jest kolejny odcinek z kolejki. Następuje sprawdzenie ile razy zawiera się on w pozostałej długości.

$$\begin{aligned}a_1 &= \lfloor L/l_1 \rfloor, \\a_2 &= \lfloor (L - l_1 a_1)/l_2 \rfloor, \\a_3 &= \lfloor (L - (l_1 a_1 + l_2 a_2))/l_3 \rfloor, \dots\end{aligned}\tag{4.1}$$

Kroki te powtarzane są dopóki kolejka się nie skończy.

Każdy element wyjściowy posiada określoną liczebność jaką powinien osiągnąć pod koniec procesu cięcia. Jeśli na danym etapie procesu cięcia wymagana liczba elementów danego typu spada do zera, wówczas jest on pomijany w dalszej pracy algorytmu. Konieczne jest sprawdzenie czy liczba uzyskanych elementów danego typu jest mniejsza lub równa od wymaganej:

- Jeśli stwierdzenie jest prawdziwe - długość z której elementy są wycinane zostanie zmniejszona o liczbę wystąpień elementu pomnożoną przez jego długość, a licznik wymaganych odcinków danej długości zostanie zmniejszony o odpowiednią liczbę wystąpień
- Jeśli stwierdzenie jest fałszywe - długość z której elementy są wycinane zostanie zmniejszona o liczbę pozostałych wykrojów pomnożoną przez długość elementu, a licznik wymaganych odcinków danej długości zostanie ustawiony na zero.

Po zakończeniu przebiegu algorytmu dla jednego układu rozkroju, można określić ile razy będzie on użyty. Zostaje to wyznaczone poprzez obliczenie

$$g = \text{floor}\{\min\{z_i/a_i\}\}, i \in [0..m], g \in Z\tag{4.2}$$

gdzie  $g$  to liczba ile razy dany schemat może zostać użyty,  $z$  to liczebność wyjściowego elementu  $i$  która pozostała do wycięcia,  $a$  to ilość wykrojów elementu  $i$  w bieżącym układzie,  $m$  to liczba długości umieszczonych w

rozkroju. Następnie licznik wymaganych odcinków elementu  $i$  zostaje zmniejszony o  $ga_i$ .

Cały proces powtarzany jest do momentu aż wszystkie wymagane elementy zostaną wycięte.

## 4.2 Rozszerzenie o szerokość cięcia

W warunkach rzeczywistych elementy wycinane są za pomocą ostrza które ma niezerową grubość. Wówczas metodę obliczania należy rozszerzyć jeśli ma odpowiadać warunkom rzeczywistym. Szerokość cięcia wlicza się w odpad. Jest kilka przypadków wliczania szerokości ostrza.

Jeżeli element jest równy długości bazowej wówczas nie wlicza się szerokości cięcia. Natomiast jeżeli materiał bazowy ma zostać pocięty na kilka elementów wówczas do każdego dolicza się szerokość cięcia. Szczególnym przypadkiem jest, gdy ostatni element wraz z szerokością ostrza jest dłuższy niż długość odcinka, który został po wycięciu wcześniejszych elementów.

Gdyby szerokość cięcia nie została uwzględniona w obliczeniach wówczas dla elementu wejściowego o długości 6000mm i wymaganych odcinkach 4500mm oraz 1500mm, obie długości zostały wycięte z jednego segmentu materiału bazowego. Skutkiem takiego postępowania byłby element krótszy o szerokość ostrza. Zazwyczaj długość ta może być akceptowana jako tolerancja dokładności maszyny. Jednak dla poprawności obliczeń wielkość ta powinna zostać uwzględniona.

## 4.3 Rozszerzenie o wiele długości bazowych

Dla zmniejszenia odpadu można użyć kilku długości bazowych. Rozszerzenie to wprowadza następującą zmianę algorytmu: obliczenia układu muszą zostać powtórzone dla każdego elementu wejściowego. Następnie wybierany jest ten rozkrój, który daje mniejszy odpad. Modyfikacja ta znacząco wpływa na wydajność metody. Jeżeli  $n$  oznacza złożoność obliczeniową podstawowego algorytmu, a  $m$  oznacza liczbę odcinków wejściowych, wówczas nowa złożoność obliczeniowa wynosi  $m * n$ .

## 4.4 Rozszerzenie o cenę materiału wsadowego

Rozszerzenie to wprowadza zmianę koncepcyjną. Każdy element bazowy posiada cenę za metr bieżący materiału, umożliwia to obliczenie kosztu odpadu i wybranie tańszej opcji wykroju.

## 4.5 Przykład

### 1. Dane wejściowe

- 6000mm - 3\$/mb
- 7000mm - 2\$/mb
- szerokość cięcia: 10mm

### 2. Dane wyjściowe

- 1x3500mm
- 1x3000mm
- 3x2000mm
- 5x500mm

### 3. Przebieg algorytmu

- Pierwszy rozkrój
  - 3500mm mieści się raz w 6000mm. Zostaje  $2500 - 10 = 2490$ mm.
  - 3000mm nie mieści się w 2490mm.
  - 2000mm mieści się raz w 2490mm. Zostaje  $490 - 10 = 480$ mm.
  - 500mm nie mieści się w 480mm.
  - Rozkrój 6000mm: 3500mm, 2000mm. Odpad  $6000 - 5500 = 500 * 0.003 = 1.5$ \$
  - —————
  - 3500mm mieści się dwa razy w 7000mm. Dostępny jest jeden odcinek 3500mm. Zostaje  $3500 - 10 = 3490$ mm.
  - 3000mm mieści się raz w 3490mm. Zostaje  $490 - 10 = 480$ mm.
  - 2000mm nie mieści się w 480mm.
  - 500mm nie mieści się w 480mm.
  - Rozkrój 7000mm: 3500mm, 3000mm. Odpad  $7000 - 6500 = 500 * 0.002 = 1.0$ \$
  - —————
  - Wybrano rozkrój 3500mm, 2000mm na długości 7000mm ze względu na mniejszy koszt odpadu.
  - Do realizacji pozostało: 0x3500mm; 0x3000mm; 3x2000mm; 5x500mm
- Drugi rozkrój
  - 2000mm mieści się trzy razy w 6000mm. Uwzględniając szerokość cięcia - zostaną użyte tylko dwa elementy od długości 2000mm. Zostaje  $2000 - 2 * 10 = 1980$ mm.



- 500mm mieści się trzy razy w 1980mm. Zostaje  $480 - 3 * 10 = 450\text{mm}$ .
- Rozkrój 6000mm: 2x2000mm, 3x500mm. Odpad  $6000 - 5500 = 500 * 0.003 = 1.5\$$
- —————
- 2000mm mieści się trzy razy w 7000mm. Zostaje  $1000 - 3 * 10 = 970\text{mm}$ .
- 500mm mieści się raz w 970mm. Zostaje  $470 - 10 = 460\text{mm}$ .
- Rozkrój 7000mm: 3x2000mm, 500mm. Odpad  $7000 - 6500 = 500 * 0.002 = 1.0\$$
- —————
- Wybrano rozkrój 3x2000mm, 500mm na długości 7000mm ze względu na mniejszy koszt odpadu
- Do realizacji pozostało: 0x3500mm, 0x3000mm, 0x2000mm, 4x500mm
- Trzeci rozkrój
  - 500mm mieści się dwanaście razy w 6000mm. Dostępne są cztery element 500mm. Zostaje  $6000 - 4 * 500 - 4 * 10 = 3960\text{mm}$ .
  - Rozkrój 6000mm: 4x500mm. Odpad  $6000 - 4 * 500 = 4000 * 0.003 = 12\$$
  - —————
  - 500mm mieści się czternaście razy w 7000mm. Dostępne są cztery elementy 500mm. zostaje  $7000 - 4 * 500 - 4 * 10 = 4960\text{mm}$
  - Rozkrój 7000mm: 4x500mm. Odpad  $7000 - 4 * 500 = 5000 * 0.002 = 10\$$
  - —————
  - Wybrano rozkrój 4x500 na długości 7000mm ze względu na mniejszy koszt odpadu
  - Do realizacji pozostało: 0x3500mm, 0x3000mm, 0x2000mm, 0x500mm
- Podsumowanie
  - Rozkroje : 3500mm, 2000mm na długości 7000mm; 3x2000mm, 500mm na długości 7000mm; 4x500 na długości 7000mm.
  - Suma odpadów:  $6000 * 0.002 = 12\$$

## 4.6 Podsumowanie

Przedstawiony algorytm jest intuicyjny oraz zwraca poprawne wyniki. Główną wadą jest brak świadomości o następnym kroku oraz kolejnych wykrojach. Dla przykładu: Zoszło 1000mm materiału, do dyspozycji (z długości mniejszych niż 1000mm) jest odcinek 900mm oraz dwa elementy 480mm. Algorytm przydzieli odcinek 900mm, jednak lepszym wyborem byłoby użycie dwóch odcinków 480mm.

## 5 Metoda "Delayed Column Generation"

$$L \geq l_1 a_1 + \dots + l_m a_m \quad (5.1)$$

$$b_1 a_1 + \dots + b_m a_m > c \quad (5.2)$$

1. Określenie  $m$  początkowych rozkrojów i ich kosztu w następujący sposób: dla każdego  $i$  wybranie długości bazowej  $L_j$  dla której  $L_j > l_i$  i określenie  $i$ -tego rozkroju jako wycięcie  $a_{ii} = \lfloor L_j / l_i \rfloor$  elementów o długości  $l_i$  z długości  $L_j$ . Koszt  $i$ -tego rozkroju będzie równy kosztowi  $c_j$  długości  $L_j$  z której  $i$ -ta operacja wycina odcinki o długości  $l_i$ .
2. Uformowanie macierzy  $B$

$$\begin{array}{cccccc} 1 & -c_1 & -c_2 & \dots & -c_m \\ 0 & a_{11} & 0 & \dots & 0 \\ 0 & 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a_{mm} \end{array}$$

gdzie  $a_{ii}$  jest ilością odcinków o długości  $l_i$  wyciętych w  $i$ -tym rozkroju z długości bazowej o koszcie  $c_j$ . Ostatnie  $m$  kolumn jest powiązane z rozkrojami. Dane te będą aktualizowane gdy zostanie znaleziony wynik który poprawi rozwiązanie.

Utworzenie  $m+1$  wymiarowych wektorów kolumnowych  $S_1, \dots, S_m$  odnoszących się do zmiennych dodatkowych, gdzie  $S_i$  posiada same zera z wyjątkiem wiersza  $(i+1)$  w którym jest  $-1$ . Dodatkowo utworzenie  $m+1$  wymiarowego wektora kolumnowego  $N'$  który jako pierwszy element przyjmuje 0, a w następnych  $i$ -tych wierszach posiada wartość  $N_i$ .

Obliczenie  $B^{-1}$  która wynosi:

$$\begin{array}{cccccc} 1 & c_1/a_{11} & c_2/a_{22} & \dots & c_m/a_{mm} \\ 0 & 1/a_{11} & 0 & \dots & 0 \\ 0 & 0 & 1/a_{22} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1/a_{mm} \end{array}$$

Niech  $N = B^{-1} \cdot N'$ . Sprawdzając czy pierwszy element z  $B^{-1} \cdot P$  jest dodatni można określić czy istnieje możliwość polepszenia rozwiązania. Wektor kolumnowy  $P$  jest wektorem złożonym ze zmiennych nieużytych w bieżącym rozwiązaniu, np. pierwszy element to negatywny koszt, a pozostałe  $m$  wierszy jest równe zmiennym  $a_{ij}$ .

3. Z powyższego punktu wynika że jeśli  $i$ -ta zmienna dodatkowa która nie wchodzi w skład rozwiązania może ulepszyć rozwiązanie wtedy i tylko wtedy gdy  $(i + 1)$  element pierwszego wiersza  $\mathbf{B}^{-1}$  jest ujemny.
4. Jeśli nie jest możliwe polepszenie rozwiązania należy określić czy wprowadzenie nowego rozkroju ulepszy bieżące rozwiązanie. Jest to możliwe poprzez sprawdzenie czy dla  $L$  z kosztem  $c$  istnieje rozwiązanie nierówności 5.1 oraz 5.2, gdzie  $b_1, \dots, b_m$  to ostatnie  $m$  elementów w pierwszym wierszu  $\mathbf{B}^{-1}$ . Jeśli te nierówności nie posiadają rozwiązania dla dowolnej długości  $L_1, \dots, L_k$  z kosztem odpowiednio  $c_1, \dots, c_m$  wtedy bieżące rozwiązanie jest minimum. Rozwiązanie i jego koszt jest określone poprzez  $\mathbf{N}$ , gdzie pierwszy wiersz to koszt, a pozostałe  $m$  wierszy jest, w kolejności, odpowiednimi wartościami  $m$ -tej kolumny z  $\mathbf{B}^{-1}$ .

Jeśli nowy rozkrój poprawi rozwiązanie wtedy formowany jest nowy wektor  $\mathbf{P}$  ze współczynnikami, w kolejności  $-c, a_1, a_2, \dots, a_m$ .

5. Wprowadzenie zarówno dodatkowej zmiennej jak i nowego rozkroju może poprawić rozwiązanie. W obu przypadkach  $\mathbf{P}$  będzie kolumnowym wektorem ze zmiennymi. Dla określenia nowych  $\mathbf{B}^{-1}$  oraz  $\mathbf{N}$  które opisują ulepszone rozwiązanie i jego koszt, co pozawala na przejście przez kroki 3, 4 oraz kontynuację kroku 5 w następujący sposób: Obliczenie  $\mathbf{B}^{-1} \cdot \mathbf{P}$  - niech elementy wynikowe będą elementy  $y_1, \dots, y_m, y_{m+1}$  oraz niech elementami bierzącego wektora  $\mathbf{N}$  będą  $x_1, \dots, x_m, x_{m+1}$ . Ustalenie  $i, i \geq 2$  dla każdego  $y_i > 0, x_i \geq 0$  oraz  $x_i/y_i$  jest najmniejsze i przypisanie tej wartości do zmiennej  $k$ . Minimalny stosunek powinien być zerem aby można było wykorzystać metodę degeneracji.

Jeśli stosunek nie jest równy zero wtedy  $k$ -ty element wektora  $\mathbf{P}, y_k$  będzie elementem wokół którego zajdzie eliminacja Gaussa odbywająca się równocześnie na  $\mathbf{B}^{-1}, \mathbf{B}^{-1} \cdot \mathbf{P}$  oraz  $\mathbf{N}$ . Eliminacja ta przebiega na macierzy  $(m + 1) \times (m + 3)$  wymiarowej  $\mathbf{G}$  uformowanej z  $\mathbf{B}^{-1}$  poprzez dołączenie kolumn  $\mathbf{B}^{-1} \cdot \mathbf{P}$  oraz  $\mathbf{N}$ . Pierwsze  $m + 1$  kolumn  $\mathbf{G}$  formuje nową macierz  $\mathbf{B}^{-1}$ , a kolumna  $m + 2$  to nowy wektor  $\mathbf{N}$ . Zależność między kolumnami  $\mathbf{B}^{-1}$  a rozkrojami lub zmiennymi dodatkowymi jest aktualizowana poprzez usunięcie  $k$ -tej kolumny i podmienieniu jej na nowy rozkrój lub zmienną dodatkową.

Degeneracja w razie wystąpienia może być obsługiwana w tradycyjny sposób. Pewne środki ostrożności powinny zostać podjęte w celu uniknięcia cykliczności. Nowa kolumna  $\mathbf{N}^1$  z dodatnimi elementami  $x'_1, \dots, x'_{m+1}$  która jest niezależna od  $\mathbf{N}$  jest dołączana do  $\mathbf{G}$  i wybór takiego  $y_i > 0$  dla którego  $x_i = 0$  który jest elementem osiowym jest dokonywany na podstawie takiego  $i$  dla którego  $x'_i > 0$  oraz  $x'_i/y_i$  jest najmniejsze. Gdy element osiowy zostanie wybrany, wówczas eliminacja Gaussa zachodzi tak jak w poprzednim przypadku na powiększonej macierzy

$\mathbf{G}$ . Dodatkowa kolumna jest przechowywana w  $\mathbf{G}$  dopóki istnieje takie  $i$  dla którego  $x_i/y_i$  jest dodatnie i skończone, jeśli warunek ten jest spełniony wówczas kolumna zostaje usunięta. Powinno to nastąpić w przypadku gdy nie istnieje takie  $i$  dla którego  $x_i/y_i$  oraz  $x'_i/y_i$  są dodatnie i skończone. Wówczas powinna zostać dodana kolumna  $\mathbf{N}^2$  niezależna od  $\mathbf{N}$  oraz  $\mathbf{N}^1$ . Podobnie dowolna liczba kolumn może zostać dodana i usunięta gdy przestanie być potrzebna. Dopóki kolumny są niezależne w czasie dodawania i pozostają takie po eliminacji Gaussa, nie potrzeba więcej jak  $m$  nowych kolumn. Każda dodana kolumna definiuje nowy problem liniowy który eliminuje problem cykliczności tak długo aż degeneracja nie wystąpi.

## 6 Wyniki

### 6.1 Porównanie

### 6.2 Wnioski

### 6.3 Podsumowanie

## **7   Opis implementacji**

### **7.1   Architektura**

### **7.2   Java**

### **7.3   Kotlin**

### **7.4   JavaFX**

## 8   Zakończenie



## Spis rysunków

## Literatura

- [1] G. B. Dantzig. Discrete variable extremum problems. *Operations Research*, 2:266 – 288, 1957.
- [2] P. J. Kolesar. A branch and bound algorithm for the knapsack problem. *Managment science*, 13:723 – 735, 1967.
- [3] D. Pisinger. *Algorithms for Knapsack Problems*. Praca doktorska, Wydział Informatyki, Uniwersytet Kopenhaski, 1995.