Optymalizacja wykorzystania materiału w procesie rozkroju rur

Jakub Pelczar

26 lutego 2017
v0.3

Spis treści

1 Wstęp **3**

2 Knapsack Problem - Problem plecakowy **4**
2.1 Zastosowanie 4
2.2 Różnorodność problemu plecakowego 4
2.3 Możliwe rozwiązania 7
2.3.1 Metoda podziału i ograniczeń 8
2.3.2 Programowanie dynamiczne 11

3 Cutting Stock Problem - Problem optymalnego rozkroju **14**
3.1 Metoda "Delayed Column Generation" 14
3.1.1 Wprowadzenie 14
3.1.2 Algorytm 17
3.1.3 Metody użyte w implementacji 19
3.1.4 Przykład 20
3.1.5 Podsumowanie 23
3.2 Metoda "Brutal Force" 24
3.2.1 Algorytm wyjściowy 24
3.2.2 Rozszerzenie o szerokość cięcia 25
3.2.3 Rozszerzenie o wiele długości bazowych 26
3.2.4 Rozszerzenie o cenę materiału wsadowego 26
3.2.5 Przykład 26
3.2.6 Podsumowanie 28

4 Wyniki **29**
4.1 Porównanie 29
4.2 Wnioski 33

5 Opis implementacji **34**
5.1 Architektura 34
5.2 Java 34
5.3 Kotlin 34
5.4 JavaFX 34

6 Zakończenie **35**

1 Wstęp

2 Knapsack Problem - Problem plecakowy

Problem plecakowy jest zagadnieniem z zakresu optymalizacji. Problem ten swoją nazwę wziął z analogii do rzeczywistego problemu pakowania plecaka. Rozwiązując go 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 Dantziga[3] w późnych latach 50 XX wieku. Znalazł on natychmiast zastosowanie w przemyśle oraz w zarządzaniu finansami. Z teoretycznego punktu widzenia, problem plecakowy często występuję jako relaksacja różnorodnych problemów programowania całkowitoliczbowego[10].

2.1 Zastosowanie

Problem plecakowy stosowany jest nie tylko w sytuacji wynikającej bezpośrednio z nazwy. Znajduje on zastosowanie w wielu dziedzinach życia oraz nauki. Diffi i Helman[4] w 1976 roku oraz Merkle i Helman[9] w 1978 roku zaproponowali problem plecakowy jako podstawę do enkrypcji kluczy prywatnych. Jednakże klucze oparte na tym algorytmie w latach późniejszych zostały złamane przez środowisko kryptograficzne i jego miejsce zajęły standardy które są bardziej odporne na złamanie (przykładowo XTR).

"Knapsack Problem" jest stosowany również podczas załadunku kontenerów służacych do przewozu materiałów drogą morską. Ładowność oraz gabaryty ładowanych elementów są ograniczane przez budowę i wytrzymałość kontenera.

Problem ten stosowany jest również w dziedzinie finansów. Jest on podstawowym narzędziem do optymalizacji portfela inwestycyjnego. Poprzez uogólnienie i modyfikacje problemu plecakowego zjawiska ekonomiczne mogą być modelowane z większą dokładnością. Przykładowo możliwe jest zakupienie 0, 1, 2 lub więcej akcji inwestycyjnych, a zakup kolejnych akcji może przynieść obniżenie przychodu.

Wiele problemów związanych z planowaniem może być przyrównana do problemu plecakowego, dla przykładu czas wykonywania operacji na maszynie jest zasobem deficytowym. Jest on szczególnie uwydatniony gdy od aktywności maszyny zależy zysk przedsiębiorstwa. Poprzez rozwiązanie problemu plecakowego możliwe jest przewidzenie zapotrzebowania na materiały podaczas procesu tak aby warunki zamówienia zostały spełnione[1].

Kolejnym zagadnieniem wynikającym z problemu plecakowego jest problem optymalnego rozkroju, zostanie on przedstawiony w rozdziale 3.

2.2 Różnorodność problemu plecakowego

Wszystkie elementy z rodziny tego problemu wymagają pewnego zestawu elementów które mogą zostać wybrane w taki sposób aby zysk został zmaksymalizowany, a pojemość placaka lub wielu plecaków nie została przekroczona. Wszystkie typy problemu należą do rodziny problemów $\mathcal{NP}$-Trudnych co oznacza, że mozliwe jest rozwiązanie problemu z użyciem algorytmów wielomianowych. Możliwe są różne warinaty problemu zależne od rozmieszczenia elementów oraz ilości plecaków[10]:

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

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

  gdzie x_j jest wartością binarną. Jeżeli $x_j = 1$ wtedy j-ty element powinien znaleźć się w plecaku, w innym przypadku $x_j = 0$.

- *Ograniczony problem plecakowy* - każdy element może być wybrany ograniczoną ilość razy. Zmianą w obecnym problemie względem problemu 0-1 jest ograniczona m_j ilość elementów j:

$$\begin{aligned} \text{maksymalizacja} \quad & \sum_{j=1}^{n} p_j x_j, \\ \text{w odniesieniu do} \quad & \sum_{j=1}^{n} w_j x_j \leq c, \\ & x_j \in \{0, 1 \ldots, m_j\}, \quad j = 1, \ldots, n \end{aligned} \tag{2.2}$$

- *Nieograniczony problem plecakowy* - jest rozszerzeniem problemu ograniczonego o nielimitowaną liczbę dostępnych elementów:

$$\begin{aligned} \text{maksymalizacja} \quad & \sum_{j=1}^{n} p_j x_j, \\ \text{w odniesieniu do} \quad & \sum_{j=1}^{n} w_j x_j \leq c, \\ & x_j \in N_0, \qquad j = 1, \ldots, n \end{aligned} \tag{2.3}$$

  Każda zmienna x_j w metodzie niograniczonej zostanie ograniczona poprzez pojemność c, gdy waga każdego z elementów jest równa przynajmniej jeden. W ogólnym przypadku transformacja problemu nieograniczonego w ograniczony nie przynosi korzyści

- *Problem plecakowy wielokrotnego wyboru* - elementy powinny być wybierane z klas rozłącznych. Problem ten jest generalizacją problemu 0-1. Możliwy jest wybór dokładnie jednego elementu j z każdej grupy N_i, $i = 1, \ldots, k$:

$$\begin{aligned}
&\text{maksymalizacja} && \sum_{i=1}^{k} \sum_{j \in N_i} p_{ij} x_{ij}, \\
&\text{w odniesieniu do} && \sum_{i=1}^{k} \sum_{j \in N_i} w_{ij} x_{ij} \leq c, \\
& && \sum_{j \in N_i} x_{ij} = 1, \qquad i = 1, \ldots, k, \\
& && x_j \in \{0, 1\}, \qquad i = 1, \ldots, k, \quad j \in N_i.
\end{aligned} \tag{2.4}$$

  Zmienna binarna $x_{ij} = 1$ określa że j-ty element został wybrany z i-tej grupy. Ograniczenie $\sum_{j \in N_i} x_{ij} = 1, \quad i = 1, \ldots, k$ wymusza wybór dokładnie jednego elementu z każdej grupy.

- *Wielokrotny problem plecakowy* - możliwość wypełnienia wielu plecaków. Jeśli jest możliwość załadowania n elmentów do m pleckaów o różnych pojemnościach c_i w taki sposób że zysk będzie jak największy:

$$\begin{aligned}
&\text{maksymalizacja} && \sum_{i=1}^{k} \sum_{j \in N_i} p_{ij} x_{ij}, \\
&\text{w odniesieniu do} && \sum_{j=1}^{n} w_j x_{ij} \leq c_i, \qquad i = 1, \ldots, m \\
& && \sum_{j \in N_i} x_{ij} \leq 1, \qquad i = 1, \ldots, k, \\
& && x_j \in \{0, 1\}, \qquad i = 1, \ldots, m, \quad j = 1, \ldots, n.
\end{aligned} \tag{2.5}$$

  Zmienna $x_{ij} = 1$ określa że j-ty element powinien zostać umiesczony w i-tym plecaku, podczas gdy ogranicznie $\sum_{j=1}^{n} w_{ij} x_{ij} \leq c_i$ zapewnia że restrykcja dotycząca pojemności plecaka zostanie zachowana. Ogranicznie $\sum_{j \in N_i} x_{ij} \leq 1$ zapewnia że każdy element zostanie wybrany tylko raz.

- *Bin-packing problem* - bardzo często spotykana wersja problemu plecakowego. Problem ten polega na umieszczeniu n elementów w jak najmniejszej liczbie opakowań:

$$\begin{aligned}
&\text{maksymalizacja} && \sum_{i=1}^{n} y_i && \\
&\text{w odniesieniu do} && \sum_{j=1}^{n} w_j x_{ij} \leq c y_i, && && i = 1, \ldots, n, \\
& && \sum_{i=1}^{n} x_{ij} = 1, && && j = 1, \ldots, n, \\
& && y_i \in \{0, 1\}, && && i = 1, \ldots, n, \\
& && x_{ij} \in \{0, 1\} && i = 1, \ldots, m, && j = 1, \ldots, n,
\end{aligned} \tag{2.6}$$

gdzie y_i określa czy i-te opakowanie zostało użyte, a x_{ij} stanowi czy j-ty element powinien zostać umieszczony w i-tym opakowaniu

- *Welokrotnie ograniczony problem plecakowy* - najbardziej ogólny typ, który jest problemem programowania całkowitoliczbowego z dodatnimi współczynnikami:

$$\begin{aligned}
&\text{maksymalizacja} && \sum_{j=1}^{n} p_j x_j, && \\
&\text{w odniesieniu do} && \sum_{j=1}^{n} w_j x_j \leq c_i, && i = 1, \ldots, m, \\
& && x_j \in N_0, && j = 1, \ldots, n.
\end{aligned} \tag{2.7}$$

2.3 Możliwe rozwiązania

Problem plecakowy należy do grupy problemów $\mathcal{NP}$-Trudnych. Rozwiązanie problemów z tej grupy jest co najmniej tak trudne, jak rozwiązanie każdego problemu z całej klasy $\mathcal{NP}$. Problem $\mathcal{NP}$-Trudny to problem obliczeniowy dla którego znalezienie rozwiązania problemu możliwe jest z wielomianową złożonością obliczeniową. Problemy $\mathcal{NP}$-Trudne obejmują zarówno problemy decyzyjne jak również problemy przeszukiwania czy też problemy optymalizacyjne.

Rozwiązanie problemu plecakowego jest możliwe przy użyciu różnych metod:

- *Metoda podziału i ograniczeń* - Metoda ta często jest stosowana do problemu plecakowego od momentu gdy Kolesar [8] zaprezentował pierwszy algorytm w 1967 roku.

- *Programownaie dynamiczne* - Gdy zostaną dodane warunki brzegowe wtedy algorytm ten staje się "zaawansowaną" formą metody podziału i ograniczeń.

- *Relaksacja przestrzeni stanów* - relaksacja programowania dynamicznego gdzie współczynniki są skalowane przez pewną stałą wartość.

2.3.1 Metoda podziału i ograniczeń

Algorytm ten polega na wypisaniu wszystkich możliwych rozwiązań używając struktury drzewa. Algorytm przechodzi kolejno po gałęziach które reprezentują podzbiory rozwiązania. Każda gałąź jest sprawdzana zadanymi warunkami brzegowymi i zostaje odrzucona jeśli nie poprawia rozwiąznaia. Przedstawione zostanie rozwiązanie nieograniczonego problemu plecakowego (2.3) [2]. Współczynniki $w_1, \ldots, w_m$, $p_1, \ldots, p_m$ oraz c są nieujemne. Stosunek p_j/w_j jest wartością jednej jednostki długości j-tego elementu. Stosunek ten jest nazywany *wydajnością* zmiennej x_j. Pierwszym krokiem algorytmu jest posortowanie zmiennych w porządku malęjącym względem wydajności:

$$p_1/w_1 \geq p_2/w_2 \geq \cdots \geq p_m/w_m \tag{2.8}$$

Dla posortowanych elementów każde rozwiązanie optymalne (2.3) spełnia warunek:

$$c - \sum_{j=1}^{m} w_j x_j < w_m \tag{2.9}$$

Głównym elementem algorytmu jest stworzenie drzewa wyliczeń oraz przeprowadzenie jego redukcji. Przykładowo dla problemu który zawiera 13 rozwiązań:

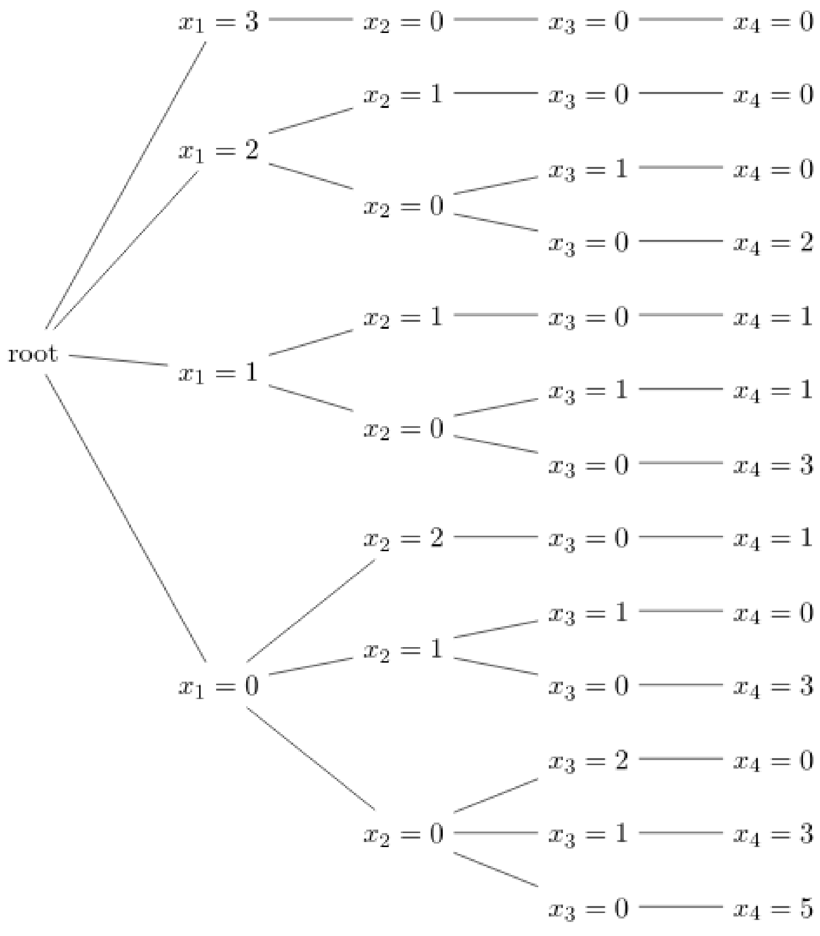
$$\begin{array}{ll} \text{maksymalizacja} & 4x_1 + 5x_2 + 5x_3 + 2x_4 \\ \text{w odniesieniu do} & 33x_1 + 49x_2 + 51x_3 + 22x_4 \leq 120 \\ & x_j \in N_0 \end{array}$$

drzewo będzie miało 13 liści (rys. 2.1). Jeśli dany węzeł posiada więcej niż jedno dziecko, wówczas potomek o większej przechowywanej wartości zostaje umieszczony wyżej. Każdy następny węzeł jest obliczany według wzoru:

$$\begin{aligned} x_j &= \lfloor (c - \sum_{i=1}^{j-1} w_i x_i)/w_j \rfloor \qquad i = 1, 2, \ldots, m \\ x_1 &= \lfloor c/w_1 \rfloor \end{aligned} \tag{2.10}$$

Podczas poszukiwania węzłów które nie mogą polepszyć rozwiązania i gałęzi które dają szansę na rozwiązanie optymalne $x_1, x_2, \ldots, x_m$ ustawione zostaje $k = m - 1$. Jeśli zachodzi taka potrzeba zmienna k jest dekrementowana dopóki nie zostanie znalezione takie x_k, że $x_k > 0$. Wówczas $x_k = x_{k-1}$, a wartości $x_{k+1}, x_{k+2}, \ldots, x_m$ są otrzymywane ze wzoru (2.10).

Dla bieżącego rozwiązania $x_1^*, \ldots, x_m^*$ zachodzi $\sum_{i=1}^{m} p_i x_i^* = M$. Maksymalne k takie, że $k \leq m - 1$ oraz $x_k > 0$ zostaje określone przechodząc od



Rysunek 2.1: Drzewo wyliczeń możliwych rozwiązań

węzłów $x_1, x_2, \ldots, x_m$ w kierunku korzenia. Podobnie jak wcześniej, niech $\bar{x}_i = x_i$ dla $i = 1, 2, \ldots, k-1$ oraz $\bar{x}_k = x_k - 1$ będą zmiennymi kandydującymi do rozwiązania. Aby okreslić czy $\bar{x}_i$ polepszy rozwiązanie x_i^*. Zgodnie z (2.8) dla każdej zmiennej $x_{k+1}, x_{k+2}, \ldots, x_m$ wydajność wynosi maksymalnie p_{k+1}/w_{k+1}, tak więc

$$\sum_{i=k+1}^{m} p_i \bar{x}_i \leq \frac{p_{k+1}}{w_{k+1}} \sum_{i=k+1}^{m} w_i \bar{x}_i$$

połączone razem z (2.3) zwraca:

$$\sum_{i=1}^{m} p_i \bar{x}_i \leq \sum_{i=1}^{m} w_i \bar{x}_i + \frac{p_i}{w_i}(c - \sum_{i=1}^{k} w_i \bar{x}_i). \tag{2.11}$$

Zgodnie z zasadami drzewa wyliczeń, nierówność

$$\sum_{i=1}^{k} p_x \bar{x}_i + \frac{p_{k+1}}{w_{k+1}}(c - \sum_{i=1}^{k} w_i \bar{x}_i) \leq M \tag{2.12}$$

określa że ścieżka $\bar{x}_1, \ldots, \bar{x}_k$ jest niegorsza niż pozostałe. Jeśli wszystkie współczynniki $p_1, \ldots, p_m$ są dodatnimi liczbami całkowitymi, wówczas również M jest liczbą całkowitą, a słaba nierówność (2.12) może zostać zastąpiona mocną

$$\sum_{i=1}^{k} p_x \bar{x}_i + \frac{p_{k+1}}{w_{k+1}}(c - \sum_{i=1}^{k} w_i \bar{x}_i) < M + 1 \tag{2.13}$$

Dla wcześniejszego przykładu powyższy krok mający na celu redukcję drzewa przyjmuje postać:

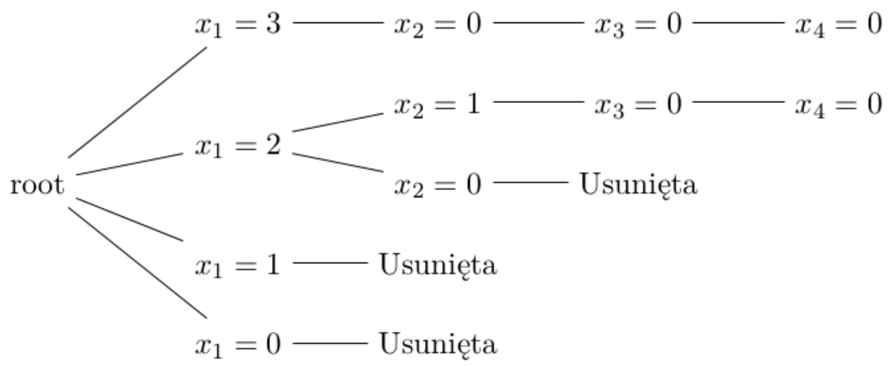
$$\begin{aligned} x_1 &= \lfloor 120/33 \rfloor = 3 \\ x_2 &= \lfloor (120-99)/49 \rfloor = 0 \\ x_3 &= \lfloor (120-99)/51 \rfloor = 0 \\ x_4 &= \lfloor (120-99)/22 \rfloor = 0 \end{aligned}$$

Z powyższego wynika że początkowe rozwiązanie to $x_1^* = 3, x_2^* = x_3^* = x_4^* = 0$ oraz $M = 12$. Początkowo $k = 3$, następnie wystepuje redukcja k dopóki nie zostanie znalezione takie $k = 1$ dla którego istnieje $x_k > 0$. Wówczas $x_1 = 3$ zostaje zaminione na $x_1 = 2$. Przed sprawdzeniem gałęzi $x_1 = 2$ przeprowadzony zostaje test (2.13) z $k = 1$ oraz $\bar{x}_1 = 2$. Wówczas lewa strona nierówności wynosi

$$8 + \frac{5}{49}(120 - 66) = 13.5$$

i jest nie mniejsza niż $M + 1 = 13$ z czego wynika że gałąź może być warta sprawdzenia. Następnie zostaje obliczona kolejna ścieżka

$$\begin{aligned} x_2 &= \lfloor (120-66)/49 \rfloor = 1 \\ x_3 &= \lfloor (120-115)/51 \rfloor = 0 \\ x_4 &= \lfloor (120-115)/22 \rfloor = 0 \end{aligned}$$



Rysunek 2.2: Zredukowane drzewo wyliczeń możliwych rozwiązań

i zastąpuje ona poprzednie rozwiązanie $x_1^* = 2, x_2^* = 1, x_3^* = x_4^* = 0$ oraz $M = 13$. Powtórzony zostaje krok z redukcją $k = 3$ dopóki nie zostanie znalezione takie $k = 2$ dla którego istanieje $x_k > 0$. Wówczas $x_2 = 1$ zostaje zamienione na $x_2 = 0$. Aby określić czy ścieżka $x_1 = 2, x_2 = 0$ jest warta sprawdzenia, zostaje przeprowadzony test (2.13) z $k = 2$ oraz $\bar{x}_1 = 2, \bar{x}_2 = 0$. Lewa strona nierówności wynosi

$$8 + \frac{5}{51}(120 - 66) = 13.3$$

Jest ona mniejsza niż $M + 1 = 14$, więc gałąź ta jest odcinana. Następnie k dalej jest zmniejszane, a kroki sa powtarzane. Dla $x_1 = 1$ wynik testu to $12.9 < 14$, a dla $x_1 = 0$ wynik to $12.2 < 14$ więc gałęzie te są odcinane. Tak więc optymalnym rozwiązaniem jest $x_1^* = 2, x_2^* = 1, x_3^* = x_4^* = 0$. Drzewo wyliczeń zostało zredukowane do postaci rys. 2.2.

Jeśli odcięta jest gałąź $\bar{x}_1, \ldots, \bar{x}_k$ wówczas odcięta również zostaje pozostała część gałęzi bez przeprowadzania dodatkowych testów.

Algorytm dla metody podziału i ograniczeń do rozwiązania problemu plecakowego, został przedstawiony poniżej

2.3.2 Programowanie dynamiczne

Metoda ta używana jest w przypadku gdy problem można podzielić na małe podproblemy które mogą zostać rozwiązane rekursywnie. Rozwiązanie optymalne podproblemu jest również optymalnym rozwiązaniem problemu głównego. Przedstawione zostanie rozwiązanie problemu plecakowego rodzaju 0-1 [7].

Jeśli elementy są oznaczone jako $1, \ldots, n$ wtedy podproblem będzie odpowiedzialny za znalezienie optymalnego rozwiązania dla $S_k = \{1, 2, \ldots, k\}$.

Algorytm 1 Metoda podziału i ograniczeń - problem plecakowy

1: M := 0
2: k := 0
3: **for** j := k+1 TO m **do**
4: $\quad x_j = \lfloor (c - \sum_{i=1}^{j-1} w_i x_i)/w_j \rfloor$
5: k := m
6: **if** $\sum_{i=1}^{m} p_i x_i > M$ **then**
7: $\quad$ M := $\sum_{i=1}^{m} p_i x_i$
8: $\quad$ **for** j := 1 TO m **do**
9: $\qquad x_j^* = x_j$
10: **if** $k = 1$ **then**
11: $\quad$ stop
12: **else**
13: $\quad k = k - 1$
14: **if** $x_k = 0$ **then**
15: $\quad$ idź do linii 10
16: **else**
17: $\quad x_k = x_k - 1$
18: **if** $!\sum_{i=1}^{k} p_x \bar{x}_i + \frac{p_{k+1}}{w_{k+1}}(c - \sum_{i=1}^{k} w_i \bar{x}_i) < M + 1$ **then**
19: $\quad$ idź do linii 3
20: **else**
21: $\quad$ idź do linii 10

Niemożliwe jest opisanie rozwiązania końcowego S_n na podstawie podproblemów S_k. Rekursywne sformułowanie podproblemu:

$$B[k,w] = \begin{cases} B[k-1,w] \quad \text{jeśli} \quad w_k > w, \\ max\{B[k-1,w], B[k-1,w-w_k] + b_k\} \quad \text{jeśli} \quad w_k \leq w. \end{cases} \tag{2.14}$$

Z powyższego równania wynika że najlepszy podzbiór podproblemu S_k z całkowitą wagą w jest najlepszym podzbiorem dla S_{k-1} którego całkowita waga wymosi w lub jest najlepszym podzbiorem dla S_{k-1} którego całkowita waga wynosi $w - w_k$ plus k-ty element. Złożoność programowania dynamicznego to $O(n * W)$. Algorytm jako dane wejściowe przyjmuje maksymalną wartość ciężaru W, oraz dwie listy: listę wag $w_1, \ldots, w_n$ oraz odpowiadającą jej listę zysku $b_1, \ldots, b_n$.

Algorytm 2 Programowanie dynamiczne - problem plecakowy 0-1

```
for w := 0 TO W do
    B[0,w] := 0
for i := 1 TO n do
    B[i,0] := 0
for i := 1 TO n do
    for w := 0 TO W do
        if w_i ≤ w then
            if b_i + B[i − 1, w − w_i] > B[i − 1, w] then
                B[i, w] := b_i + B[i − 1, w − w_i]
            else
                B[i, w] := B[i − 1, w]
        else
            B[i, w] := B[i − 1, w]
```

3 Cutting Stock Problem - Problem optymalnego rozkroju

Problem optymalnego rozkroju jest problemem wykroju zadanej liczby elementów z wielu elementów podstawowych takich, jak rury, arkusze papieru lub metalu, w taki sposób aby zminimalizować niewykorzystany materiał (odpad). Jest to problem optymalizacyjny znajdujący zastosowanie głównie w przemyśle. W odniesieniu do złożoności obliczeniowej jest to problem z rodziny problemów $\mathcal{NP}$-Trudnych, który może zostać zredukowany do problemu plecakowego (rozdział 2). W rozdziale niniejszym zostanie opisany jednowymiarowy problem optymalnego rozkoroju.

3.1 Metoda "Delayed Column Generation"

Metoda ta została zparoponowana przez Gilmore'a i Gomorego w 1961 roku [5]. Gdy problem optymalnego rokroju zostanie sformułowany jako problem programowania całkowitoliczbowego wówczas liczba zmiennych wchodzących w skład równań powoduje że rozwiązanie jest nieosiąglane. Dla przykładu gdy podstawowa długość to 200 z której ma zostać wycięte 40 różnych elementów o długościach od 20 do 80 wówczas liczba różnych wzorców rozkroju może osiągnąć nawet 100 milionów. Czas potrzebny do przejścia po samych rozkrojach byłby niosiagalny. Metoda ta pozwala na ciągłą generację nowych rozwiązań. Jest ona również metodą która znosi restrykcję liczb całkowitych w trakcie obliczania wyniku, dlatego wynik zostaje zaokraglony w górę, co odnosi skutek w tym że jest produkowane więcej lub tyle samo elementów niż jest wymagane przez zlecenie. Wynikiem tej metody jest rozwiązanie najbliższe optymalnemu.

3.1.1 Wprowadzenie

Założeniem metody jest że zamówienie N_i elementów długości l_i, gdzie $i = 1, 2, \ldots, m$, wyciętych z rur długości początkowych $L_1, L_2, \ldots, L_k$, dla którego spełniony jest warunek, że isntiej takie j, że dla każdego i spełnina jest nierówność $L_j \geq l_i$. Całkowity koszt rozkrojów jest całkowitym kosztem użytych elementów podstawowych. Celem rozwiązania problemu jest otrzymanie tylu wykrojów ile jest wymaganych przez zamówienie przy jak najmniejszym koszcie. Warunkiem koniecznym aby zamówinie zostało zrealizowane jest nierówność

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_i \geq N_i, \qquad i = 1, \ldots, m$$

gdzie a_{ij} oznacza krotność długości l_i w danym schemacie rozkroju x_i. Funkcja kosztu która powinna zostać zminimalizowana wynosi

$$c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{3.1}$$

gdzie c_i to koszt długości podstawowej, z której jest pobierany i-ty wykrój. Wprowadzenie dodatkowych zmiennych $x_{n+1}, \ldots, x_{n+m}$ pozwalają opisać problem optymalnego rozkroju jako problem znalezienia takich liczb całkowitych $x_1, \ldots, x_{n+m}$ spełniających

$$a_{i1}x_1 + \cdots + a_{in}x_n - x_{n+i} = N_i, \qquad i = 1, \ldots, m \tag{3.2}$$

$$x_j \geq 0, \qquad j = 1, \ldots, n+m \tag{3.3}$$

dla których funkcja (3.1) jest jak najmniejsza.

Takie sformułowanie problemu jest niepraktyczne ze względu na ograniczenie do liczb całkowitych oraz z uwagi na fakt, iż n może być bardzo duże nawet gdy ilość k elementów podstawowych, jak i ilość m zamówionych długości jest umiarkowana.

Jeśli zostanie usunięty warunek całkowitoliczbowości rozwiązania wówczas rozwiązanie będzie należało do zbioru liczb rzeczywistych dodatnich. Rozwiązanie to może zostać zaokrąglone w górę lecz wtedy może zostać wyprodukowane więcej elementów niż zostało zamówione. Rozwiązanie może być również zaokrąglane na przemina w górę i w dół, a elementy które nie spełniają założeń zamówienia są dodawane do wykrojów metodą *ad hoc*. Gdy wartości niecałkwite są duże wówczas zaokrąglenie jej nie wpływa znacząco na koszt, jednak gdy wartości są rzędu dziesiątek wówczas zaokrąglenie ma znaczny wpływ na koszty. Omawiana metoda znosi ograniczenie dla liczb całkowitych.

Usunięty warunek całkowitoliczbowości odnosi skutek w tym, że zmienne dodatkowe mogą zostać usunięte z równiania (3.2). Dopóki rozwiązania (3.2) oraz (3.3) zawierają dodatnie zmienne dodatkowe wówczas istnieje rozwiązanie o takim samym koszcie w którym nie zawierają się dodatnie zmienne dodatkowe. Niech $\bar{x}_1, \ldots, \bar{x}_n, \bar{x}_{n+1}, \ldots, \bar{x}_{n+m}$ będzie rozwiązaniem (3.2) oraz (3.3) dla którego $\bar{x}_{n+1} \neq 0$. Dla tego rozwiązania istnieje takie i, dla którego $a_{1i}\bar{x}_i \geq \bar{x}_{n+1}$, to jest, i-ty schemat rozkroju należy do rozwiązania w przynajmniej takiej liczebności aby zamówinie długości l_1 było spełnione. Jeśli nie istnieje takie i, które spełnia warunek, wówczas niech j-ta zmienna przyjmuje niezerową wartość $\bar{x}_j$ oraz niech k-ty rozkrój będzie identyczny jak j-ty z wyłączeniem uwzględniania długości l_1. W takim przypadku w k-tym rozkroju długość l_1 która została uwzględniona w j-tym rozkroju traktowana jest jako odpad. Rozwiązanie $\bar{x}_1{}', \ldots, \bar{x}_n{}', \bar{x}_{n+1}{}', \ldots, \bar{x}_{n+m}{}'$ z tym samym kosztem co poprzednio zostało uzyskane poprzez przypisanie $\bar{x}_i{}' = \bar{x}_i$ dla $i \neq j,\ k,\ n+1$: $\bar{x}_j{}' = 0, \bar{x}_k{}' = \bar{x}_k + \bar{x}_j$ oraz $\bar{x}_{n+1}{}' = \bar{x}_{n+1} - a_{1j}\bar{x}_j$ ponieważ koszt zmiennych x_j oraz x_k jest taki sam. W nowym rozwiązaniu zmienna x_{n+1} została zredukowana. Jeśli nie została zmniejszona o tyle aby $a_{1i}\bar{x}_i{}' \geq \bar{x}_{n+1}{}'$ wtedy powyższy proces jest powtarzany, dopóki nie zostanie znalezione rozwiązanie w którym jedna zmienna nie spełnia nierówności. Jeśli $\mathrm{a}_{1i}\bar{x}_i{}' \geq \bar{x}_{n+1}{}'$ jest spełnione, wówczas zmienna dodatkowa x_{n+1} może być traktowana jako zmienna z przechowywaną wartością 0 w rozwiązaniu z takim smaym kosztem jak powyższe rozwiązanie. Niech k-ty rozkrój będzie schematem identyczny jak j-ty rozkrój z wyłączeniem długości l_1 oraz niech określa nowe rozwiązanie $\bar{x}_1{}', \ldots, \bar{x}_n{}', \bar{x}_{n+1}{}', \ldots, \bar{x}_{n+m}{}'$ poprzez przypisanie $\bar{x}_i{}' = \bar{x}_i$ dla $i \neq j, k, n+1$, $\bar{x}_j{}' = \bar{x}_j - (\bar{x}_{n+1})/a_{1j}, \bar{x}_k{}' = \bar{x}_k + (\bar{x}_{n+1})/a_{1j}$ oraz $\bar{x}_{n+1}{}' = 0$. Ponieważ współczynniki odpowiedzialne za koszt są identyczne dla x_j oraz x_k, nowe rozwiązanie posiada taki sam koszt jak poprzednie rozwiązanie.

Zniesienie warunku całkowitoliczbowości rozwiązania pozawala pominąć zmienne dodatkowe, jednak w pewnych przypadkach jest zalecane pozostawienie ich. Bez zmiennych dodatkowych każde minimalne rozwiązanie zawiera zazwyczaj m schematów rozkroju, podczas gdy rozwiązanie ze zmiennymi dodatkowymi może zawierać mniej niż m rozkrojów. Opisywana metoda nie znosi zmiennych dodatkowych.

Metoda simplex jest stosowana do obliczenia dopuszczalnego rozwiązania (3.2) w odniesieniu do (3.3) dla którego (3.1) jest namjniejsze. Dla podstawowego rozwiązania (3.3) oraz (3.1), metodą simplex sprawdzane są inne zmienne które mogą zastąpić pewne zmienne w bierzącym rozwiązaniu. Niech bierzącym rozwiązniem będzie $x_1, x_2, \ldots, x_m$. Niech $\boldsymbol{P}_{\mathrm{i}}$ będzie wektorem $[a_{1i}, a_{2i}, \ldots, a_{mi}]$ oraz niech c_i bedzie kosztem w (3.1) który jest powiązany ze zmienną x_i. Jeśli x_i jest zmienną dodatkową wówczas koszt wynosi 0, a wektor ma jedną niezerową współrzędną wynoszącą -1. Niech $\boldsymbol{P} = [a_1, a_2, \ldots, a_m]$ określa nowy schemat rozkroju który używa długości bazowej L o koszcie c. Następnie niech $\boldsymbol{A}$ będzie macierzą której kolumnami są wektory $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_m$. Ponieważ $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_m$ określą podstawę macierzy, pomocniczy wektor kolumnowy $\boldsymbol{U}$ spełnia układ równań

$$\boldsymbol{A} \cdot \boldsymbol{U} = \boldsymbol{P}. \tag{3.4}$$

Nowy schemat rozkroju może zostać użyty w rozwiązaniu jako jego ulepszenie wtedy i tylko wtedy, gdy

$$\boldsymbol{C} \cdot \boldsymbol{U} > c \tag{3.5}$$

gdzie $\boldsymbol{C}$ jest wektorem wierszowym ze współczynnikami $c_1, c_2, \ldots, c_m$. Jeśli wektor wierszowy $\boldsymbol{C} \cdot \boldsymbol{A}^{-1}$ posiada współczynniki $b_1, \ldots, b_2$, wtedy z równań (3.4) oraz (3.5) można wywnioskować że istnieje taki rozkrój z elementu podstawowego o długości L, który może poprawić rozwiązanie wtedy i tylko wtedy, gdy istnieją nieujemne liczby całkowite $a_1, \ldots, a_m$ spełniające nierówności

$$L \geq l_1 a_1 + \cdots + l_m a_m \tag{3.6}$$

$$b_1 a_1 + \cdots + b_m a_m > c. \tag{3.7}$$

$\boldsymbol{C} \cdot \boldsymbol{A}^{-1}$ zawsze jest częścią rozwiązania normalnej metody simplex.

Jeśli istnieje taka nieujemna liczba całkowita a_i która spełnia nierówności (3.6) oraz (3.7), wówczas istnieje taka nieujemna liczba całkowita która jest rozwiązaniem nierówności (3.6) dla której $b_1a_1+\cdots+b_ma_m$ jest maksymalne. Problem wyboru nowej zmiennej dla metody simplex może zostać wyrażony poprzez rozwiązanie k problemów pomocniczych (po jednym dla każdej długości bazowej), które są całkowitoliczbowymi problemami programowania liniowego. Problemy te mogą zostać rozwiązane poprzez programowanie dynamiczne lub metodą *ad hoc*.

Jako, że maksymalizacja $b_1a_1 + \cdots + b_ma_m$ w odniesieniu do (3.6) jest generalizacją problemu plecakowego, dlatego można rozwiązać go metoda opisaną przez Dantziga [3] (patrz rozdział 2.3.2). Niech $F_s(x)$ będzie wartością maksymalną $b_1a_1 + \cdots + b_sa_s$ w odniesieniu do nierówności $x \geq l_1a_1 + \cdots + l_sa_s$, wówczas

$$F_{s+1}(x) = max_r\{rb_{s+1} + F_s(x - rl_{s+1})\},$$

gdzie r może zostać wybrane z zakresu $0 \leq r \leq \lfloor x/ls + 1 \rfloor$. Tylko jedno kompletne obliczenie wyrażenia programowania dynamicznego jest niezbędne aby wprowadzić nową zmienną do metody simplex. Gdy najdłuższym elementem jest L_1, wówczas automatycznie zostaną obliczone pozostałe długości.

Programowanie dynamiczne często wymaga więcej obliczeń niż jest konieczne. Aby przyspieszyć proces możliwe jest użycie metody podobnej do zapropnowanej przez Dantziga [3] (patrz rozdział 3.2). Niech $i_1, \ldots, i_m$ będą takie, że $b_{i1}/l_{i1} \geq b_{i1}/l_{i2} \geq \cdots \geq b_{im}/l_{im}$. Następnie obliczone zostają współczynniki $a_{i1} = [L/l_{i1}], a_{i2} = [(L-l_{i1}a_{i1})/l_{i2}], a_{i3} = [(L-(l_{i1}a_{i1}+l_{i2}a_{i2}))/l_{i3}]$, *etc.* Dopiero gdy proste metody nie dostarczą rozwiązania, powinny zostać użyte bardziej złożone metody, jak programowanie dynamiczne.

3.1.2 Algorytm

1. Określnie m początkowych rokrojów i ich kosztu przebiega w następujący sposób: dla każdego i wybranie długości bazowej L_j dla której $L_j > l_i$ i określenie i-tego rokroju jako wycięcia $a_{ii} = [L_j/l_i]$ elementów o długości l_i z L_j. Koszt i-tego rozkroju będzie równy cenie c_j długości L_j z której i-ta operacja wycina odcinki o długości l_i.

2. Uformowanie macierzy $\boldsymbol{B}$

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

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 odpowiada kolejmnym rozkrojom. Dane te będą aktualizowane gdy zostanie znaleziony wynik który zmniejszy koszt rozwiązania.

Utworzenie m $m+1$ wymiarowych wektorów kolumnowych $\boldsymbol{S_1}$,...,$\boldsymbol{S_m}$ odnoszących się do zmiennych dodatkowych, gdzie $\boldsymbol{S_i}$ zawiera same zera z wyjątkiem wiersza $(i+1)$ który przechowuje wartość -1. Stworzony również zostaje $m+1$ wymiarowy wektor kolumnowy $\boldsymbol{N'}$ który jako pierwszy element przyjmuje 0, a w następnych i-tych wierszach posiada wartości N_i.

Obliczenie macierzy $\boldsymbol{B}^{-1}$ która wynosi:

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

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

3. Z powyższego puntku 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 $\boldsymbol{B}^{-1}$ jest ujemny.

4. Jeśli nie jest możliwe zmniejszenie kosztu rozwiązania należy określić czy wprowadznie nowego rozkroju poprawi rozwiązanie. Jest to możliwe poprzez sprawdznie czy dla L z kosztem c istnieje rozwiązanie nierówności (3.6) oraz (3.7), gdzie $b_1, \dots, b_m$ to ostatnie m elementów z pierwszego wiersza $\boldsymbol{B}^{-1}$. Jeśli te nierówności nie posiadają rozwiązania dla dowolnej długości $L_1, \dots, L_k$ z kosztem odpowiednio $c_1, \dots, c_m$, wówczas bieżące rozwiązanie jest optymalne. Rozwiązanie i jego koszt jest określone poprzez $\boldsymbol{N}$, gdzie pierwszy wiersz odpowiada cenie, a pozostałe m wierszy jest, w kolejności, odpowiednimi wartościami m-tej kolumny z $\boldsymbol{B}^{-1}$.

   Jeśli nowy rozkrój zmniejsza koszt rozwiązania, zostaje uformawany nowy wektor $\boldsymbol{P}$ o współczynnikach, w kolejności $-c, a_1, a_2, \dots, a_m$.

5. Wprowadznie zarówno dodatkowej zmiennej jak i nowego rozkroju może poprawić rozwiązanie. W obu przypadkach $\boldsymbol{P}$ będzie wektorem kolumnowym. Określenia nowych $\boldsymbol{B}^{-1}$ oraz $\boldsymbol{N}$ które opisują ulepszone rozwiązanie i jego koszt, zostaje osiągnięte poprzez przejście kroków 3, 4 oraz kontynujacje kroku 5 w nastepujący sposób: Obliczenie $\boldsymbol{B}^{-1} \cdot \boldsymbol{P}$

- niech wynikiem będą elementy $y_1, \ldots, y_m, y_{m+1}$ oraz niech elementami bierzącego wektora $\boldsymbol{N}$ będą $x_1, \ldots, x_m, x_{m+1}$. Ustalenie i, $i \geq 2$ dla którego $y_i > 0$, $x_i \geq 0$ oraz x_i/y_i jest najminiejsze, a następnie przypisanie tej wartości do zmiennej k.

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

3.1.3 Metody użyte w implementacji

Do uzyskania maksymalnego rozwiązania spełniającego nierówności (przykładowo 3.7 oraz 3.6) które zostanie przypisane do $\boldsymbol{P}$ zostały użyte metody:

1. Dwufazowa metoda simplex - metoda to znajduje zastosowanie gdy bierzące rozwiązanie układu jest ujemne. Zwykła metoda sympleks jest użyta w drugiej fazie omawianej procedury. Faza pierwsza polega na przeprowadzeniu obliczeń metodą simplex ze zmienioną funcją celu. Jeśli zmienna wchodząca w skład rozwiązania układu jest ujemna wówczas do danego równania dodawana jest dodatkowa sztuczna zmienna. Funkcja celu wówczas przyjmuje postać sumy zmiennych które zostały dodane jako sztuczne do równań o ujemnym rozwiązaniu. Po obliczeniu wartości fazy pierwszej, nastepuje ponowne przekształcenie funkcji celu i przeprowadzenie normlanej procedury sympleks, jako fazy 2.

2. Metoda podziału i ograniczeń - metoda ta pozwala osiągnąć wyniki całkowite z rozwiązań układów nierówności. Polega ona na budowie drzewa binarnego. Każdy liść staje się rodzicem poprzez stworznie dwóch węzłów oraz sprawdzenie dwóch warunków. Lewy potomek tworzony jest z dodatkowym warunkiem $x_i \leq \lfloor c_i \rfloor$ gdzie c_i jest zmienną niecałkowitą wchodzącą w skład rozwiązania. Prawy potomek posiada warunek $x_i \geq \lceil c_i \rceil$. Następnie dla każdego węzła przeprowadzana jest metoda sympleks. Jeśli dany węzeł posiada rozwiązanie wówczas procedura ta jest powtarzana, aż do osiągnięcia wyniku całkowitego. Poszczególne warunki dołączane są do układu nierówności który przekazywany jest do kolejnych potomków. Jeśli tworzenie drzewa binarnego jest zakończone, wówczas jako rozwiązanie wybierany jest liść z jak największą wartością zwróconą przez metodę simplex.

3.1.4 Przykład

Zamówione zostało 20 elementów o długości 2, 10 o długości 3 oraz 20 o długości 4. Jako długości bazowe zostały określone elementy o długości 5 z ceną 6, 6 z ceną 7 oraz o długości 9 z ceną 10.
Początkowo:

$$\boldsymbol{B} = \begin{bmatrix} 1.0 & -6.0 & -6.0 & -6.0 \\ 0.0 & 2.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}, \qquad \boldsymbol{N'} = \begin{bmatrix} 0.0 \\ 20.0 \\ 10.0 \\ 20.0 \end{bmatrix}$$

$$\boldsymbol{B}^{-1} = \begin{bmatrix} 1.0 & 3.0 & 6.0 & 6.0 \\ 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}, \qquad \boldsymbol{N} = \begin{bmatrix} 240.0 \\ 10.0 \\ 10.0 \\ 20.0 \end{bmatrix}$$

Długości bazowe będą próbowane w kolejności malejącej ponieważ im dłuższy element, tym więcej możliwości rozkroju. Pierwszy układ nierówności:

$$\begin{aligned} 2.0x_1 + 3.0x_2 + 4.0x_3 &\leq 9.0 \\ 3.0x_1 + 6.0x_2 + 6.0x_3 &> 10.0 \end{aligned}$$

Rozwiązaniem nierówności jest $(0.0, 3.0, 0.0)$. Wówczas wektor $\boldsymbol{P} = [-10.0, 0.0, 3.0, 0.0]$ oraz

$$\boldsymbol{G} = \begin{bmatrix} 1.0 & 3.0 & 6.0 & 6.0 & 240.0 & 8.0 \\ 0.0 & 0.5 & 0.0 & 0.0 & 10.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 10.0 & 3.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 20.0 & 0.0 \end{bmatrix}$$

gdzie ostatnią kolumną jest $\boldsymbol{B}^{-1}\boldsymbol{P}$. Element osiowy wokół którego zajdzie eliminacja Gaussa to wartość z ostatniej kolumny wynosząca 3.0. Macierzą po eliminacji Gaussa $\boldsymbol{G'}$ jest:

$$\boldsymbol{G'} = \begin{bmatrix} 1.0 & 3.0 & 3.33 & 6.0 & 213.33 & 0.0 \\ 0.0 & 0.5 & 0.0 & 0.0 & 10.0 & 0.0 \\ 0.0 & 0.0 & 0.33 & 0.0 & 3.33 & 1.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 20.0 & 0.0 \end{bmatrix}$$

Zmieniona zostaje druga nierówność na $3.0x_1{+}3.33x_2{+}6.0x_3 > 10.0$. Wektor $\boldsymbol{P}$ dla takiego układu wynosi $[-10.0, 3.0, 1.0, 0.0]$ oraz

$$\boldsymbol{G} = \begin{bmatrix} 1.0 & 3.0 & 3.33 & 6.0 & 213.33 & 2.33 \\ 0.0 & 0.5 & 0.0 & 0.0 & 10.0 & 1.5 \\ 0.0 & 0.0 & 0.33 & 0.0 & 3.33 & 0.33 \\ 0.0 & 0.0 & 0.0 & 1.0 & 20.0 & 0.0 \end{bmatrix}$$

gdzie element osiowy wynosi 1.5. Po eliminacji Gaussa:

$$\boldsymbol{G'} = \begin{bmatrix} 1.0 & 2.22 & 3.33 & 6.0 & 197.78 & 0.0 \\ 0.0 & 0.33 & 0.0 & 0.0 & 6.67 & 1.0 \\ 0.0 & 0.11 & 0.33 & 0.0 & 1.11 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 20.0 & 0.0 \end{bmatrix}$$

Zmodyfikowana nierówność wynosi $2.22x_1 + 3.33x_2 + 6.0x_3 > 10.0$. Wektor $\boldsymbol{P}$ dla takiego układu wynosi $[-10.0, 0.0, 0.0, 2.0]$ oraz

$$\boldsymbol{G} = \begin{bmatrix} 1.0 & 2.22 & 3.33 & 6.0 & 197.78 & 2.0 \\ 0.0 & 0.33 & 0.0 & 0.0 & 6.67 & 0.0 \\ 0.0 & 0.11 & 0.33 & 0.0 & 1.11 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 20.0 & 2.0 \end{bmatrix}$$

gdzie element osiowy wynosi 2.0. Po eliminacji Gaussa:

$$\boldsymbol{G'} = \begin{bmatrix} 1.0 & 2.22 & 3.33 & 5.0 & 177.78 & 0.0 \\ 0.0 & 0.33 & 0.0 & 0.0 & 6.67 & 0.0 \\ 0.0 & 0.11 & 0.33 & 0.0 & 1.11 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.5 & 10.0 & 1.0 \end{bmatrix}$$

Zmodyfikowana nierówność wynosi $2.22x_1 + 3.33x_2 + 5.0x_3 > 10.0$. Wektor $\boldsymbol{P}$ dla takiego układu wynosi $[-10.0, 1.0, 1.0, 1.0]$ oraz

$$\boldsymbol{G} = \begin{bmatrix} 1.0 & 2.22 & 3.33 & 5.0 & 177.78 & 0.56 \\ 0.0 & 0.33 & 0.0 & 0.0 & 6.67 & 0.33 \\ 0.0 & 0.11 & 0.33 & 0.0 & 1.11 & 0.22 \\ 0.0 & 0.0 & 0.0 & 0.5 & 10.0 & 0.5 \end{bmatrix}$$

gdzie element osiowy wynosi 0.5. Po eliminacji Gaussa:

$$\boldsymbol{G'} = \begin{bmatrix} 1.0 & 2.5 & 2.5 & 5.0 & 175.0 & 0.0 \\ 0.0 & 0.5 & 0.5 & 0.0 & 5.0 & 0.0 \\ 0.0 & 0.5 & 1.5 & 0.0 & 5.0 & 1.0 \\ 0.0 & 0.25 & 0.75 & 0.5 & 7.5 & 0.0 \end{bmatrix}$$

Układ nierówności:

$$\begin{aligned} 2.0x_1 + 3.0x_2 + 4.0x_3 &\leq 9.0 \\ 2.5x_1 + 2.5x_2 + 5.0x_3 &> 10.0 \end{aligned}$$

nie posiada rozwiązania całkowitoliczbowego. Wówczas brana jest następna długość podstawowa 6. Nowy układ wynosi:

$$\begin{aligned} 2.0x_1 + 3.0x_2 + 4.0x_3 &\leq 6.0 \\ 2.5x_1 + 2.5x_2 + 5.0x_3 &> 7.0 \end{aligned}$$

dla którego wektor $\boldsymbol{P}$ wynosi $[-7.0, 1.0, 0.0, 1.0]$ oraz

$$\boldsymbol{G} = \begin{bmatrix} 1.0 & 2.5 & 2.5 & 5.0 & 175.0 & 0.5 \\ 0.0 & 0.5 & 0.5 & 0.0 & 5.0 & 0.5 \\ 0.0 & 0.5 & 1.5 & 0.0 & 5.0 & 0.5 \\ 0.0 & 0.25 & 0.75 & 0.5 & 7.5 & 0.75 \end{bmatrix}$$

gdzie element osiowy wynosi 0.75. Po eliminacji Gaussa:

$$\boldsymbol{G'} = \begin{bmatrix} 1.0 & 2.33 & 3.0 & 4.67 & 170.0 & 0.0 \\ 0.0 & 0.33 & 0.0 & 0.33 & 0.0 & 0.0 \\ 0.0 & 0.33 & 1.0 & 0.33 & 10.0 & 0.0 \\ 0.0 & 0.33 & 1.0 & 0.67 & 10.0 & 1.0 \end{bmatrix}$$

Układ nierówności:

$$\begin{aligned} 2.0x_1 + 3.0x_2 + 4.0x_3 &\leq 6.0 \\ 2.33x_1 + 3.0x_2 + 4.67x_3 &> 7.0 \end{aligned}$$

nie posiada rozwiązania. Podobnie układ

$$\begin{aligned} 2.0x_1 + 3.0x_2 + 4.0x_3 &\leq 5.0 \\ 2.33x_1 + 3.0x_2 + 4.67x_3 &> 6.0 \end{aligned}$$

również nie posiada rozwiązania.

Następnie wykorzystywana jest metoda programowanie dynamicznego w celu określenia czy kolejny rozkrój może polepszyć rozwiązanie. Z metody tej wynika że możliwym jest ulepszenie rozwiązania poprzez rozkrój z długości 9. Jednak układ nierówności:

$$\begin{aligned} 2.0x_1 + 3.0x_2 + 4.0x_3 &\leq 9.0 \\ 2.33x_1 + 3.0x_2 + 4.67x_3 &> 10.0 \end{aligned}$$

nie posiada rozwiązania całkowitoliczbowego. Wektor $\boldsymbol{N}$ równy jest przedostatniej kolumnie ostatniej obliczonej macierzy $\boldsymbol{G'}$, czyli $[170.0, 0.0, 10.0, 10.0]$ oraz

$$\boldsymbol{B} = \begin{bmatrix} 1.0 & -10.0 & -10.0 & -7.0 \\ 0.0 & 3.0 & 1.0 & 1.0 \\ 0.0 & 1.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 1.0 \end{bmatrix}$$

Na podstawie $\boldsymbol{N}$ oraz $\boldsymbol{B}$ można uzyskać wynikowe rozkroje. Pierwszy element wektora $\boldsymbol{N}$ okrśla że koszt zbioru rozkrojów wynosi 170. Następne elementy są ilością kolejnych rozkrojów, tak więc pierwszy schemat nie będzie brany pod uwagę, a dwa następne zostaną wykonane 10 razy. Macierz $\boldsymbol{B}$ jest analizowana od drugiej kolumny. Pierwszy wiersz równy jest kosztowi długości z której ma być wykonany rozkrój, pomnożonemu przez -1. Następnie wiersze w kolumnach określają ile elementów o danej długości powinno znaleźć się w rozkroju.

Rozwiązaniem powyższego przykładu jest układ rozkrojów:

1. elementy: $[2.0, 3.0, 4.0]$, ilość: 10, odpad: 0, długość bazowa: 9
2. elementy: $[2.0, 4.0]$, ilość: 10, odpad: 0, długość bazowa: 6

3.1.5 Podsumowanie

W drugiej części artykułu poświęconemu problemowi optymalnego rozkroju [6], Gilmore oraz Gomory opisali wyniki eksperymentów wykorzystujących różne warianty metody zawartej w części pierwszej. Problem który został użyty do testów jest problemem z przemysłu papierniczego. W podstawowym zbiorze 20 problemów długości bazowe miały tę samą długość 200 in lub mniej. Liczba elementów wynikowych była z przedziału od 20 do 40. Długości elementów wynosiły od 20 in. do 80 in. z dokładnością do 1/4 in. Liczba noży wynosiła pięć, siedem lub dziewięć.

Średnia liczba iteracji metody simplex dla tego problemu to w przybliżeniu 130. Jednak ich rozpiętość była duża od 20 do 300. Taka zmienność jest powszechna dla problemów programowania linowego. Problemy które są niemal identyczne mogą zachowywać się bardzo odmiennie w odniesieniu do metody sympleks. Zgodnie z przewidywaniami, problemy z mniejszą liczbą elementów wchodzących w skład rozkroju, potrzebują mniej iteracji. Trend ten jest niedeterministyczny, dla przykładu 35 elementowy rozkrój wymagał 197 iteracji, gdzie problem pokrewny dla 40 elementów wymagał ich tylko 161.

Gilmore oraz Gomory zbadali czy nowy schemat rozkroju powinien być akceptowany przy spełnieniu warunku $\sum b_i a_i > 1$, czy lepsze jest pozostanie przy maksymalizacji $\sum b_i a_i$. Pierwsza metoda prowadzi do większej liczby iteracji, jednak ich czas jest mniejszy niż w przypadku metody drugiej. Wynik eksperymentu odpowiedział na pytanie czy lepiej zastosować więcej krótszych iteracji, czy lepiej mniej ale dłuższych? W 19 na 20 przypadków średni czas potrzebny na rozwiązania problemu z użyciem metody drugiej był mniejszy niż metody pierwszej. Metodę tą można opisać jako znalezienie schematu który najbardziej wpłynie na obniżenie kosztu i zmniejszenie odpadu. Najprostsza implementacja tej metody wymaga wypisania bardzo dużej liczby schemtów rozkroju dla każdej iteracji. Aby zredukować liczbę nakładu obliczeniowego na każdą iteracje, można ostateczne elementy pogrupować w bardzo małe zbiory. Jeśli nowy rozkrój będzie zawierał długość, która jest wymagana w małej ilości, wówczas schemat zostanie użyty tylko kilka razy. Wykorzystanie tego ulepszenia nie ma znaczącego wpływu na wynik. Gilmore oraz Gomory zaproponowali użycie metody medianowej. Metoda ta polega na równym podziale wymaganych elementów ze względu na liczbę elementów wynikowych - na wymagane w dużej lub małej ilości. W każdej drugiej iteracji schemat albo używa jedynie elementów wymaganych wiele razy lub maksymalizuje ulepszenie pośród wszystkich rozkrojów. Metoda medianowa była szybsza w 13 na 20 przypadków. Problemy dla których metoda ta była wolniejsza były małymi problemami które wymagały mało czasu do wykonania. Średnio czas został zredukowany o 40%.

W przypadkach testowych odpad wahał się od 0.1% do 10% oraz zachowywał się nieprzewidywalnie. Zauważone zostało również że problemy z dużym odpadem były rozwiązywane szybciej. Typowy problem z małym odpadem w początkowych iteracjach drastycznie malał, a następnie nieznacznie spadał dalej. Aby przyspieszyć obliczenia Gilmore oraz Gomory zaporponowali aby zakończyć obliczenia jeśli odpad wynosi poniżej 0.1% po 10 iteracjach. Dzięki zastosowaniu tego skrótu czas wykoywania zmniejszył się o 90% przy zachowaniu maksyamalnie 0.5% odpadu przy przedwczensym zakończeniu algorytmu.

Podczas badania zachowania algorytmu dla wielu długości początkowych, zauważono spadek ilości odpadu. Dla pojedynczej długości 168 in. odpad wynosił 7%. Po dodaniu dodatkowych elementów 145 in., 140 in, oraz 124 in. odpad zmniejszył się do 1.4%. Czas wykonania wzrósł od 144% do 211%.

Limitacja liczby noży nie zmieniła wartości rozwiązania. Po zniesieniu ograniczenia dla 19 przypadków z 20 ilość odpadu pozostała taka sama, przy jednoczesnej zmianie schematów rozkrojów.

3.2 Metoda "Brutal Force"

3.2.1 Algorytm wyjściowy

Metoda ta opiera się zarówno na intuicji jak i na rozwiązaniu zaproponowanym przez Dantziga dla problemu plecakowego [3]. Jest to metoda która w prosty sposób - nie używając złożonych modeli matematycznych, pozwala osiągnąć optymalny rozkrój materiału.

Pierwszym krokiem jest posortowanie elementów wyściowych malejąco względem ich długości $l_1 \geq l_2 \geq ... \geq l_m$ 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.

$$\begin{aligned} a_1 &= [L/l_1], \\ a_2 &= [(L - l_1 a_1)/l_2], \\ a_3 &= [(L - (l_1 a_1 + l_2 a_2))/l_3], ... \end{aligned} \tag{3.8}$$

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. Koniecznie 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

$$g = \lfloor min\{z_i/a_i\} \rfloor, \qquad i \in [0..m], g \in Z \tag{3.9}$$

gdzie g to liczba ile razy dany schemat może zostać użyty, z to liczbność wyjściowego elementu i która pozostała do wycięcia, a to ilość wykrojów elementu i w bierzącym układzie, m to liczba długości umieszczonych w rozkroju. Następnie licznik wymaganych odcinków elemntu i zostaje zmniejszony o ga_i.

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

3.2.2 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 elmentów 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 toleracncja dokładności maszyny. Jednak dla poprawności obliczeń wielkość ta powinna zostać uwzględniona.

3.2.3 Rozszerzenie o wiele długości bazowych

Dla zmniejszenia odpadu można użyć kilku długości bazowych. Rozszerzenie to wprowadza następująca 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 $m * n$.

3.2.4 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.

3.2.5 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 sie 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 posostało: 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 $480 - 3 * 10 = 450$mm.
- Rozkrój 6000mm: 2x2000mm, 3x500mm. Odpad $6000 - 5500 = 500 * 0.003 = 1.5\$$
- ————
- 2000mm mieści się trzy razy w 7000mm. Zostaje $1000 - 3 * 10 = 970$mm.
- 500mm mieści się raz w 970mm. Zostaje $470 - 10 = 460$mm.
- Rozkój 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 posostało: 0x3500mm, 0x3000mm, 0x2000mm, 4x500mm

• Trzeci rozkrój

- 500mm mieści się dwanaście razy w 6000mm. Dostępne są cztery element 500mm. Zostaje $6000 - 4 * 500 - 4 * 10 = 3960$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 $7000 - 4 * 500 - 4 * 10 = 4960$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 posostało: 0x3500mm, 0x3000mm, 0x2000mm, 0x500mm

* Podsumowanie
  - Rozkroje : 3500mm, 2000mm na długości 7000mm; 3x2000mm, 500mm na długości 7000mm; 4x500 na długości 7000mm.
  - Suma odpadów: $6000 * 0.002 = 12\$$

3.2.6 Podsumowanie

Przedstawiony algorytm jest intuicyjny oraz zwraca poprawne wyniki. Główną wadą jest brak świadomości o następnym kroku oraz kolejnych wykrojach. Dla przykładu: Zosatło 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.

4 Wyniki

Niniejsza sekcja przedstawia i porównuje wyniki otrzymane z eksperymentu. Metody wchodzące w skład porównania to "Delayed Column Generation" (rozdział 3.1). oraz "Brutal Force" (rozdział 3.2). Warunki przeprowadzenia testu:

- Losowo generowane odcinki wynyikowe o długości od 1 do 21 cm, przy liczebności od 1 do 200 elementów.
- Losowo generowane 5 długości początkowych od 22 do 42 cm, o koszcie od 1$. Każda długość posiada inną cenę.
- Obie metody testowane są takimi samymi danymi.
- Wykonano 27 różnych rozkrojów.
- Czas wykonania mierzony od dostarczenia danych do zwrócenia wyniku, bez uwzględnianienia czasu przygotowania danych oraz ich zapisu.
- Warunki sprzętowe:
  - Procesor: Intel Core i5-6500u @ 2.30 GHz x 2 z technologią HT.
  - RAM: 16 GB (15.2 GB).
  - System operacyjny: Linux Mint 18.1 Cinammon 64-bit.
- Język implementacji: Kotiln 1.0.5 (JVM), Java 8 (Oracle Java 1.8_121).
- Aplikacja jednowątkowa.

4.1 Porównanie

Dane z tabeli 1 przedstawiają porównanie podstawowych statystyk dla każdego kroku eksperymentu. Natomiast tabela 2 przedstawia średnie wartości statystyk przedstawionych w poprzedniej tabeli.

Dane przedstawione w powyższych tabelach wskzaują jednoznacznie że, metoda brutalnej siły (dalej BF) jest szybsza niż druga metoda użyta w porównaniu. Tabela 2 wskazuje iż metoda opóźnieonej generacji kolumn (dalej DCG) jest średni prawie 30156 razy wolniejsza niż metoda BF. Ma to związek z nakładem obliczeniowym metody DCG. Metoda ta wykonuje wiele obliczeń macierzowych, dla każdej itereacji zachodzi odwracanie macierzy, mnożenie wektorów, eliminacja Gaussa oraz rozwiązywanie układu nierówności dwufazowa metoda sympleks. Najwyższy czas wykonania metody DCG wynodi 446814 ms, czyli ponad 7 minut. Najmniejszy czas wykonania tej samej metody przy innych danych wejściowych i zachowaniu warunków testu wynosi 4664 ms, czyli 4,7 s. Rozbieżność czasów wykonania metody DCG wskazują na silną zależność między danymi wejściowymi, a czsem wykonania. Czas

Tabela 1: Wyniki

Czas (ms)		**Koszt**		**Odpad**	
DCG	*BF*	*DCG*	*BF*	*DCG*	*BF*
136396	11	1830	2835	0	0
27688	2	1719	2342	304	13
190893	3	3279	3421	109	33
113044	2	1936	5397	69	20
14453	5	1821	2342	819	8
446814	1	2912	4254	3947	3
20758	3	4544	5050	1729	3
101468	2	3024	6658	54	1
272598	1	2324	2560	44	0
18424	1	2365	4001	877	64
284007	1	1802	4000	46	40
36820	6	3078	3255	393	115
25840	8	4068	6981	325	14
42254	16	948	1034	1480	102
4664	1	3174	3707	1434	1297
11725	3	1377	2904	46	2
34074	6	1161	1490	411	45
323568	7	3072	3638	81	1
124059	4	8128	8971	0	51
27697	1	830	3965	0	9
169184	2	2754	3255	0	18
227189	7	3184	5741	150	94
25436	5	1235	1850	35	21
232145	3	4485	4598	0	4
47524	2	2993	4046	1278	159
77913	2	5002	5196	23	10
201725	1	2366	3162	0	14
18760	2	5330	5398	64	12

Tabela 2: Średnie

	DCG	BF
Średni czas	116325.71	3.86
(DCG/BF) * 100%	3015851.85%	
Średni koszt	2883.61	4001.82
(BF/DCG) * 100%	138.78%	
Średni odpad	489.93	76.89
(DCG/BF) * 100%	637.16%	

wykonania metody BF jest bardzo niski, na poziomie kilkunastu milisekund, jest to związane ze spsobem wykonania. Główœnym elementem implmentacji tej metody jest przeszukiwaniem, przechodzenie oraz uzupełnianie tablic. Operacje te są znacznie szybsze niż operacje macierzowe. Mediana czasów obu metod pokazuje że, metoda DCG nadal jest dużo wolniejsza niż BF, jednak w innej skali niż porównanie średnich. Mediana dla metody DCG to 62718.5 ms, a dla BF to 2.5 s. Metoda DCG jest ponad 25087 razy wolniejsza niż metoda BF.

Kolejna część powyższych tabel odnosi się do średniego kosztu wykroju całkowitego. Koszt uzyskany metodą BF jest średnio 1.4 razy większy niż metodą DCG. Liczba ile razy koszt jednej metody zawiera się w drugej wydaje się mały. Jednak po sprawdzeniu wielkości kosztów wynika iż, różnica między ceną rozwiązania metodą DCG oraz metodą BF wynosi 1118, 21$. Rząd wielkości oznacza że, różnica w cenie jest znacząca. Metoda DCG jako główny cel ma minimalizację kosztu, natomiast metoda BF jak najmniejszą cenę odpadu w ujęciu bierzącecgo schematu rozkroju.

Ostatnie częsci powyższych tabel ukazują odpad powstały z rozkroju otrzymanymi schmatami. Odpad wyprodukowany z metody dCg jest ponad 6 razy większy niż odpad z metdoy BF. W jednym przypadku na 28, odpad z metdy DCG był mniejszy niż z metody BF. Tak jak zostało wspomnianie w powyższym akapicie metoda BF skupia się na minimalizacji kosztu odpadu, więc w ogólnym przypadku minimalizuje odpad.

Base Lengths	
Length	Cost
25	7
31	21
33	9
36	15
Request	
Length	Quantity
2	66
4	167
5	174

7	151
9	200
10	135
12	150
15	26
17	8

OUTPUT

Used base

Length	Quantity
25	305
33	21

Output table

Quantity	Waste	Used base	Elements					
14	0	25	2	2	2	2	5	12
76	0	25	4	4	5	12		
4	0	25	5	5	5	5	5	
16	0	25	4	7	7	7		
96	0	25	7	9	9			
68	0	25	5	10	10			
31	1	25	12	12				
13	1	33	2	15	15			
8	0	33	7	9	17			

Statistic

Exec time (ms)	272598
Total cost	2324
Waste	44
Waste percent	5.289733E-05

Output elements

Length	Quantity
2	69
4	168
5	178
7	152
9	200
10	136
12	152
15	26
17	8

Cost decreasing

Step	Cost
0	2755.0166
1	2692.1833
2	2661.5166
3	2437.5166
4	2343.0166
5	2324.7388
6	2317.6765
7	2314.585
8	2303.9917

4.2 Wnioski

5 Opis implementacji

5.1 Architektura

5.2 Java

5.3 Kotlin

5.4 JavaFX

6 Zakończenie

Spis rysunków

2.1 Drzewo wyliczeń możliwych rozwiązań 9
2.2 Zredukowane drzewo wyliczeń możliwych rozwiązań 11

Literatura

[1] J. J. Bartholdi. The knapsack problem. In D. Chhajed and T. J. Lowe, editors, *Building Intuition*, chapter 2, pages 19 – 31. Springer US, 2008.

[2] V. Chvatal. *Linear Programming.* W.H. Freeman and Company, New York, 1984.

[3] G. B. Dantzig. Discrete variable extremum problems. *Operations Research*, 2:266 – 288, 1957.

[4] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644 – 654, 1976.

[5] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operation research*, 9(6):849 – 859, Nov. – Dec. 1961.

[6] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem - part II. *Operation research*, 11(6):863 – 888, Nov. – Dec. 1963.

[7] S. Goddard. Lecture about dynamic programming 0-1 knapsack problem. `http://cse.unl.edu/~goddard/Courses/CSCE310J/`.

[8] P. J. Kolesar. A branch and bound algorithm for the knapsack problem. *Managment science*, 13:723 – 735, 1967.

[9] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24:525 – 530, 1978.

[10] D. Pisinger. *Algorithms for Knapsack Problems.* PhD thesis, Dept. of Computer Science, University of Copenhagen, 1995.