Autor: Jan Feręc

Problem plecakowy w 3D

1. Opis problemu

a. Treść zadania:

Dany jest zbiór prostopadłościanów A = {a_1, …, a_n} – *pudełek* reprezentowanych przez klasę *Cuboid* oraz obszar prostopadłościenny o zadanych dwóch bokach nie mniejszych od największego z boków prostopadłościanów w zbiorze – dla naszych testów przyjmujemy, że będzie to kwadrat, reprezentowanych przez parę liczb całkowitych – *pojemnik* (klasa *Bin*). Należy wyznaczyć, stosując różne metody heurystyczne, a także przeszukiwanie systematyczne, najmniejszą długość trzeciego boku obszaru - *height* pozwalającą na rozmieszczenie ortogonalne bez kolizji zbioru w obszarze. Porównać czas obliczeń i wyniki różnych metod.

b. Black-box:

i. *Wejście:*
Program powinien przyjmować na wejściu początkowe przyjęte rozmiary pojemnika oraz kontener (strukturę danych) zawierający wymiary pudełek w postaci trójek liczb całkowitych (a, b, c), które mają być uporządkowane w pojemniku.

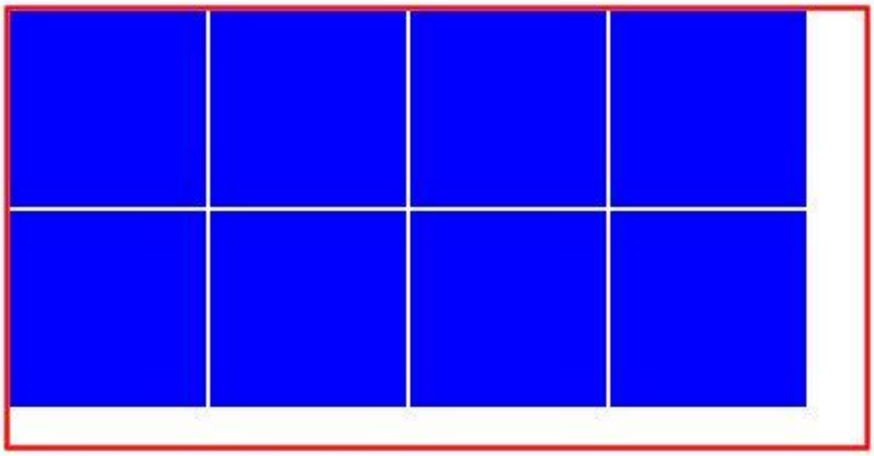
ii. *Wyjście:*
Każdy z algorytmów musi zwrócić kontener zawierający niezmienione rozmiary pudełek (a, b, c) wraz z przyporządkowanymi im współrzędnymi środka pudełek (x, y, z) w pojemniku.

2. Algorytmy naiwne

a. Działanie algorytmu:

Krok 1)
Obróć wszystkie pudełka należące do zbioru A, tak aby ich współrzędna Z, czyli wysokość była minimalna: min (x, y, z).
Krok 2)
Przeszukaj wszystkie pudełka i znajdź 2 najdłuższe krawędzie x oraz y pudełek w całym zbiorze, tak aby każde pudełko zrzutowane ortogonalnie do 2D mogło się zmieścić obrębie prostokąta o bokach x i y. W celu zminimalizowaniu prostokąta można dokonać obrotu o pudełka o 90 w płaszczyźnie Z (zamiana x i y miejscami)

Krok 3)



Pojemnik: 43x22 Prostokąty: 10x10
Udało się zmieścić 8 prostokątów

Podziel powierzchnię podstawy na obliczone prostokąty w punkcie 2). Celem jest uzyskanie jak największej liczby k takich prostokątów.
Krok 4)
Wybieramy i-ty prostokąt.
Krok 5)
Wkładamy tam pierwsze z brzegu pudełko, sprawdzamy czy aktualna wysokość całego pojemnika została przekroczona;
Jeśli TAK: nadpisujemy wysokość pojemnika
Krok 6)
Usuwamy ze zbioru pudełek do rozstawienia postawione pudełko, jeśli zbiór jest pusty STOP, w przeciwnym wypadku inkrementuj i modulo k oraz przejdź do Krok 4)

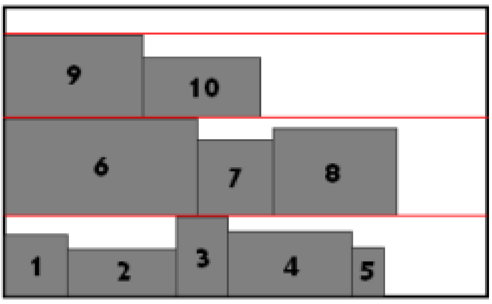
b. *Komentarz:*
Algorytm ten działa w czasie liniowym, nie stosujemy żadnego sortowania. Może znaleźć zastosowanie w przypadku, gdy potrzebujemy bardzo szybkiego algorytmu i wycinamy pudełka z taniego materiału lub przestrzeń w naszym pojemniku jest bardzo tania.

c. *Wariacja algorytmu:*
Drugi naiwny algorytm dodatkowo będzie wykorzystywać sortowanie prostokątów z kroku 3) po wysokościach, zamiast wybierania kolejnego prostokąta w ciemno, będziemy wybierać prostokąt o najmniejszej wysokości, co ograniczy puste, niewykorzystane przestrzenie między pudełkami, przy nieco większej złożoności n*log(k).

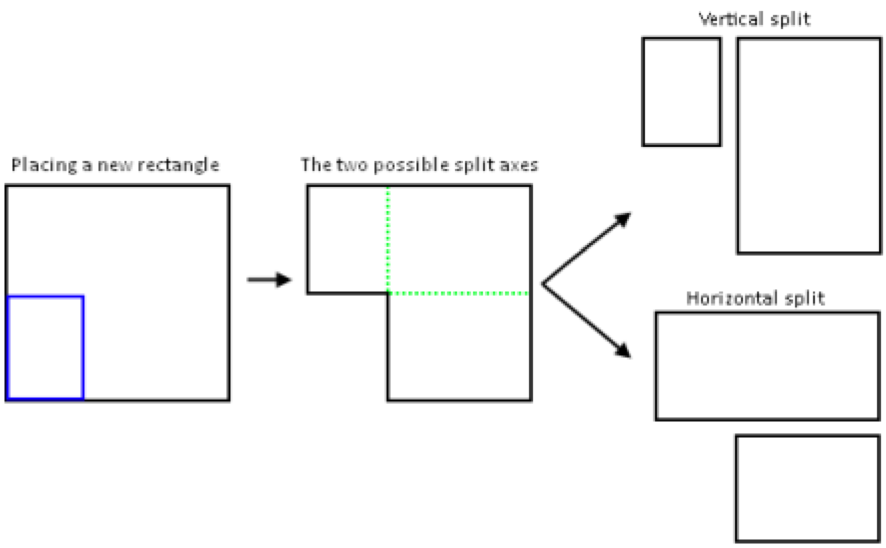
3. Algorytm półkowy (Shelf Algorithm)

a. *Działanie algorytmu:*
Krok 1)
Posortuj wszystkie pudełka w zbiorze nierosnąco według współrzędnej z.
Krok 2)
Tworzymy półkę w płaszczyźnie Z o wysokości najwyższego w zbiorze

pudełek nieprzydzielonych. Półkę definiujemy jako płaszczyznę prostopadłą do osi Z, dzielącą zbiór A na dwa niepuste podzbiory pudełek w taki sposób, że owa płaszczyzna nie przecina żadnego pudełka.
Krok 3)
Po utworzeniu półki na osi Z, rozwiązujemy problem pakowania w 2D w obrębie płaszczyzny XY, odpowiednio dzieląc półkę Z na półki Y.



Krok 4)
W obrębie półki Y przechowujemy wolne miejsca zdefiniowane dwoma wierzchołkami, początkowym i końcowym
Krok 5) Jeśli pudełko jest pierwsze na nowej półce, ustawmy je poziomo, aby ograniczyć wysokość półki - h_s. Jeśli pudełko mieści się pionowo, tzn., że jego współrzędna y jest mniejsza od h_s to kładziemy je pionowo, w przeciwnym przypadku pudełko jest kładzione poziomo. W przypadku, gdy nie ma miejsca na obecnej półce nowa półka Y jest tworzona. Jeśli brakuje miejsca na nowe półki w płaszczyźnie XY tworzona jest nowa półka na osi Z.
Należy podjąć decyzję, który typ algorytmu półkowego zostanie zaimplementowany. Wykorzystamy Shelf Best Area Fit(SHELF-BAF), który będzie wybierać tą półkę, po której położeniu pudełka pozostała na półce niewykorzystana przestrzeń zostanie zminimalizowana. Po znalezieniu odpowiedniego miejsca dokonujemy operacji Horizontal Split.



4. Przeszukiwanie systematyczne

a. *Działanie algorytmu:*

Krok 1)
Jeśli wszystkie permutacje kolejności wprowadzania pudełek zostały sprawdzone STOP. W przeciwnym razie przejdź do kolejnej permutacji.
Krok 2)
Jeśli wszystkie permutacje ułożenia klocków zostały wykorzystane przejdź do Krok 1). W przeciwny razie obróć klocek ustawienia klocków.
Krok 3)
BEGIN LOOP
Dla wszystkich pudełek:
Połóż pudełko w najniższym położeniu dostępnym jednocześnie sprawdzając kolizje.
END LOOP
Jeśli permutacja dała lepszy wynik niż do tej pory zapisz obecne ułożenie pudełek w pojemniku.
Krok 4)
Idź do Krok 2)

5. Generacja danych

Testy będą przeprowadzone na jednej, kwadratowej podstawie pojemników $x = 1000$ oraz $y = 1000$. Rozmiary pudełek będą losowane rozkładem jednostajnym ciągłym od $a = 1$ do b dla każdej krawędzi oddzielnie, gdyż zależy nam na jak największej różnorodności rozmiarów pudełek. Testy zostaną przeprowadzone dla różnych wartości b, przykładowo równej 25%,50%, 75% krawędzi podstawy pudełka.
Rozmiary zbiorów do testowania będą (w miarę możliwości) dobierane w taki sposób, aby pomiary trwały od kilkunastu sekund do kilku minut, aby wyniki były jak najbardziej miarodajne.

6. Testowanie

Każdy algorytm testowany będzie na identycznych wygenerowanych zbiorach danych. Otrzymane wyniki z pomiarów będą analizowane pod kątem:
a) Uzyskanej wysokości pojemnika (***precyzja***) – chcemy, aby wartość współrzędnej z pojemnika była jak najmniejsza. Im mniejsza wysokość, tym mniej wolnego, zmarnowanego miejsca pomiędzy pudełkami. Ponieważ każdy algorytm przyjmuje identyczny zbiór pudełek, posiadając uzyskaną wysokość możemy obliczyć miarodajny parametr określający jak dobrze algorytm się spisuje:

$$\frac{\sum_{i=1}^{n} a_i \cdot b_i \cdot c_i}{x \cdot y \cdot h} = \frac{\sum_{i=1}^{n} V_i}{V_p}$$

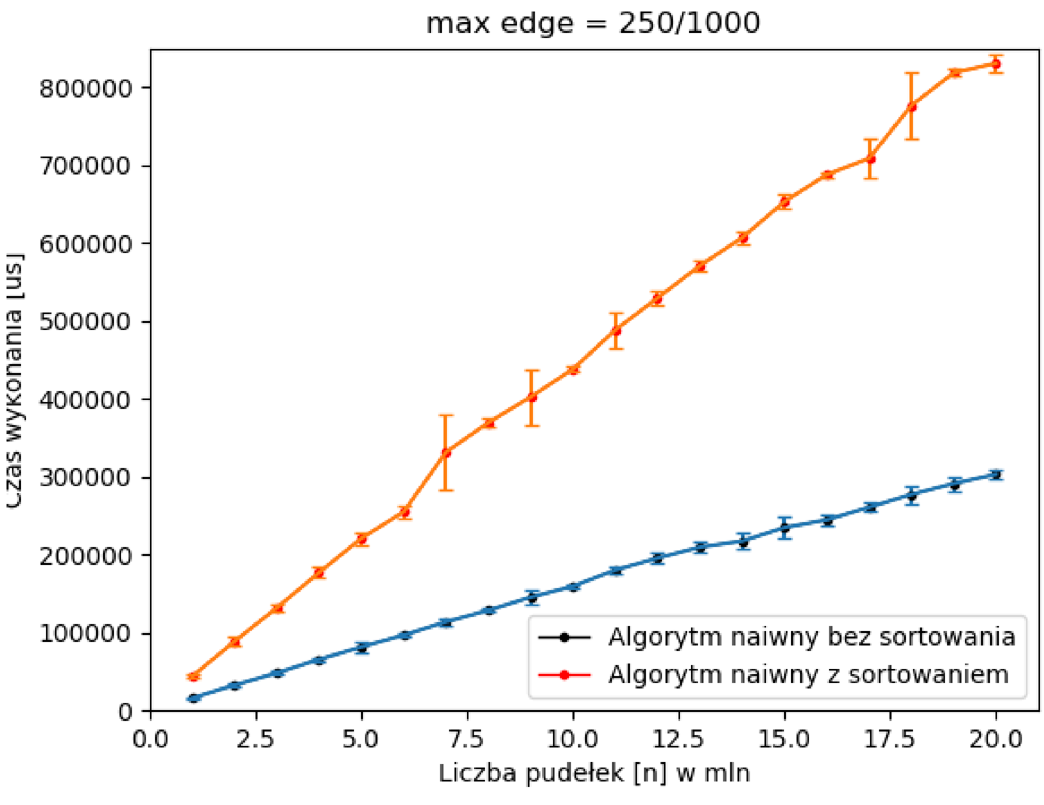
a_i, b_i, c_i - długości boków i-tego pudełka
V_i – objętość i-tego pudełka
n – całkowita liczba pudełek w zbiorze
x, y – wymiary podstawy pojemnika

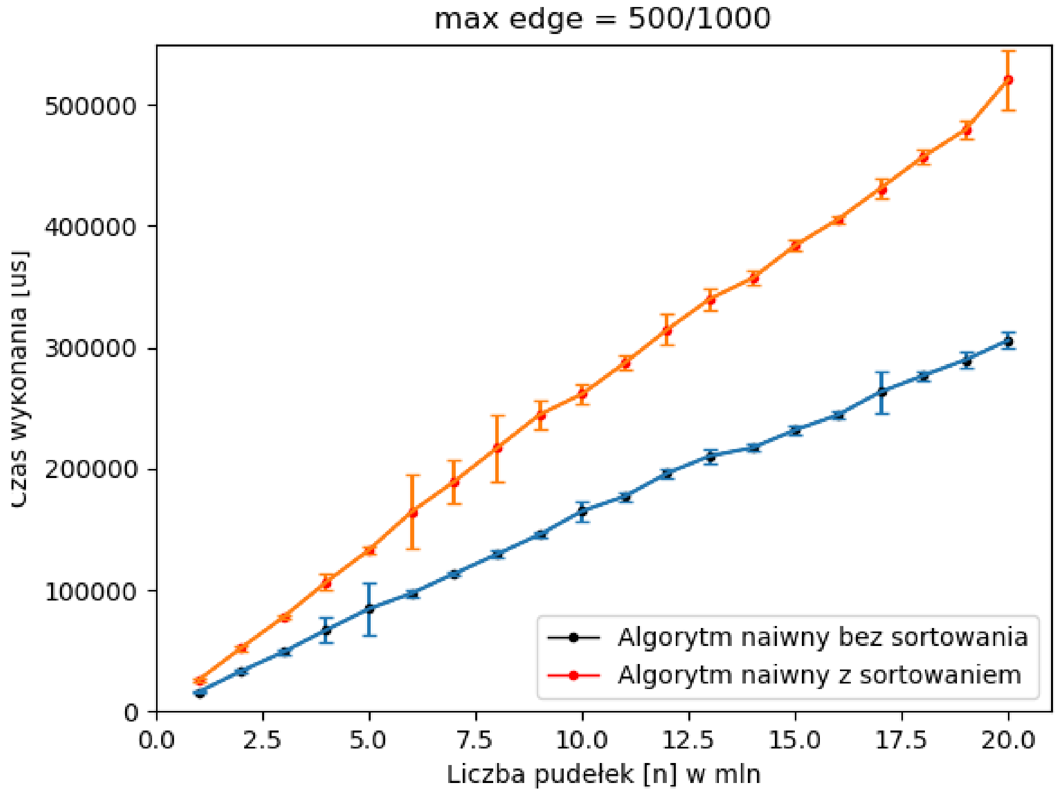
h – wynikowa wysokość pojemnika
V_p – wynikowa objętość pojemnika
Parametr określa jaki procent objętości pojemnika jest zapełniony pudełkami.
b) Czasu wykonania obliczeń – Im szybciej algorytm zakończy swoje działanie tym lepiej

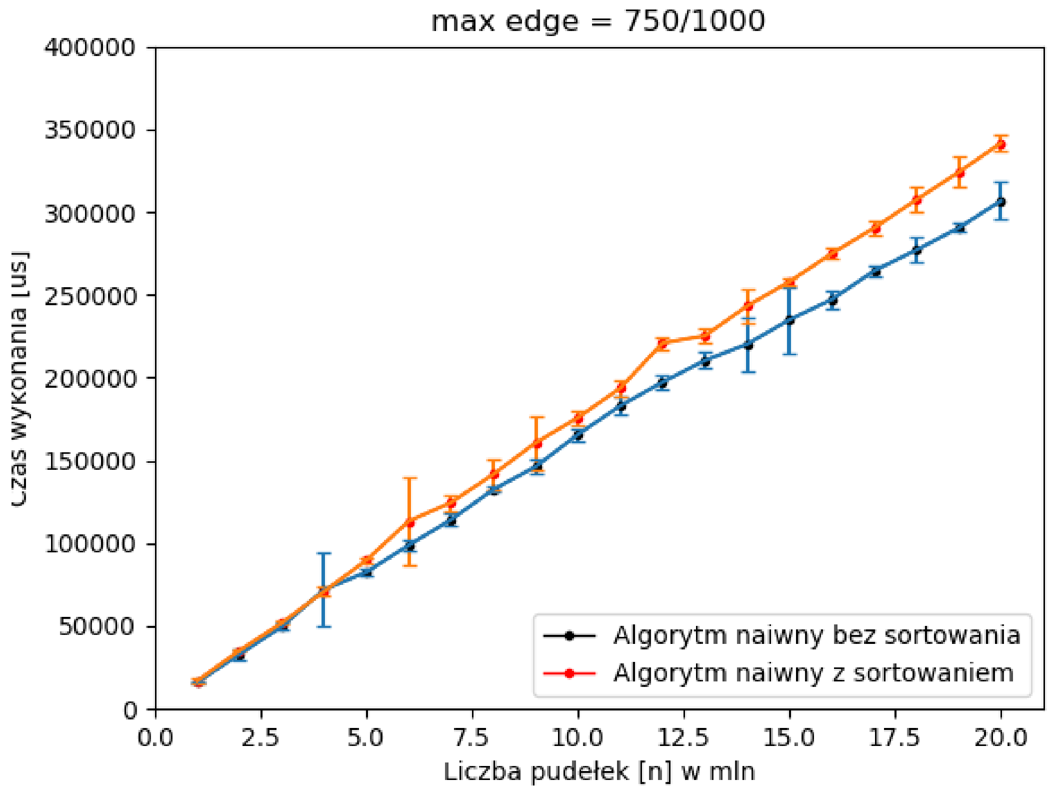
Z powodu możliwej nieprzewidywalności wyników wynikającej z natury JVM, do testowania wykorzystana będzie biblioteka JMH oraz wszelkie zalecenia znajdujące się w artykule Avoiding Benchmarking Pitfalls on the JVM.

Wyniki testów dla algorytmów naiwnych:

Porównanie czasowe dla trzech ustawień maksymalnych krawędzi pudełek.




max edge = 500/1000
Czas wykonania [us]
Liczba pudełek [n] w mln
Algorytm naiwny bez sortowania
Algorytm naiwny z sortowaniem
0
100000
200000
300000
400000
500000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


max edge = 750/1000
Czas wykonania [us]
Liczba pudełek [n] w mln
Algorytm naiwny bez sortowania
Algorytm naiwny z sortowaniem
0
50000
100000
150000
200000
250000
300000
350000
400000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Pomiary czasowe dla tych algorytmów były utrudnione ze względu na jego niską złożoność – bardzo krótki czas obliczeń. Przyjąłem następujące parametry do testów:
-Tryb testowania : single shot (1 iteracja = pomiar pojedynczego wykonania algorytmu)
-10 iteracji do rozgrzania maszyny
-5 iteracji właściwych, z obliczonym progiem błędu, które stanowią pojedynczy punkt pomiarowy

Na wykresie można zauważyć, że największy % błędu ukazuje się dla najmniejszej liczby pudełek co wynika z bardzo krótkiego czasu wykonania algorytmu, jednakże przy wszystkich pomiarach udało się uzyskać 15-18/20 punktów pomiarowych o błędzie <5%.
Niewrażliwość algorytmu na dane wynika z faktu, iż przyjęliśmy, że długości krawędzi pudełek są losowane po rozkładzie liniowym od 1 do <maksymalna długość krawędzi>, biorąc pod uwagę, że za każdym razem losujemy miliony pudełek przypadki testowe nie różnią się bardzo od siebie.

Do obliczeń q(n) w tabeli poniżej był uwzględniony najbardziej pesymistyczny przypadek z 5 iteracji – krawędź 250/1000, algorytm bez sortowania.

n	t(n)[us]	q(n)
1000000	15319	0.9670882346751962
2000000	29899	0.9437617053513183
3000000	45781	0.9633846089097452
4000000	63249	0.9982270995980724
5000000	74252	0.9375055238736558
6000000	94415	0.9934029166052903
7000000	108988	0.9829158236809591
8000000	126202	0.9958912901664527
9000000	136672	0.9586779178328177
10000000	157339	0.9932808653016562
11000000	175308	1.0061083042712218
12000000	188369	0.9909776730287663
13000000	203375	0.9876200481406601
14000000	207509	0.9357171415945431
15000000	221059	0.9303634182782349
16000000	238005	0.9390782496159593
17000000	254972	0.9468457907773281
18000000	265480	0.9310971289885875
19000000	282115	0.9373639798028351
20000000	297624	0.9394499274005177

Na pierwszy rzut oka możemy zauważyć, że potwierdziła się oczekiwana złożoność obliczeniowa = n. Każde kolejne pudełko kładzione jest w wydzielonym sektorze, a wraz z rosnącą liczbą pudełek zapotrzebowanie czasowe rośnie liniowo.
Kolejnym wnioskiem widocznym na wykresie jest to, że stała przy złożoności n w algorytmie ze znajdowaniem najniższego sektora jest zależna od ilości sektorów. Dla maksymalnej krawędzi = 250 mamy 16 sektorów, dla krawędzi = 500 mamy 4 sektory, a dla krawędzi = 750 tylko jeden sektor. Widać, że im większa liczba sektorów tym algorytm jest wolniejszy, ze względu na potrzebę znajdowania tego najniższego.

Pomiary precyzji algorytmu:
Z powodu bardzo wysokiej liczby pudełek przy testowaniu wyniki działania algorytmu są niemalże jednakowe dla wszystkich punktów pomiarowych:

Algorytm – maksymalna krawędź	Precyzja
Naiwny – 1/4	50.1% ± 0.1%
Naiwny – 2/4	50.1% ± 0.1%
Naiwny – 3/4	28.2% ± 0.1%
Naiwny z sortowaniem – 1/4	50.2% ± 0.1%
Naiwny z sortowaniem – 2/4	50.1% ± 0.1%
Naiwny z sortowaniem – 3/4	28.2% ± 0.1%

Widać, że kluczowe znaczenie w algorytmach ma dopasowanie sektorów. Dla krawędzi 250 i 500 dopasowanie jest idealne – sektory pokrywają 100% podstawy pojemnika. W przypadku krawędzi 750 widać wyraźnie niższą precyzje, wynikającą z tego, że sektor pokrywa jedynie 9/16 powierzchni podstawy. Można więc uznać, że precyzja algorytmu jest wprost proporcjonalna do pokrycia podstawy pojemnika przez sektory. Kolejną obserwacją jest fakt, iż wybór najniższego sektora nie przynosi żadnych korzyści przy tak dużej ilości pudełek.

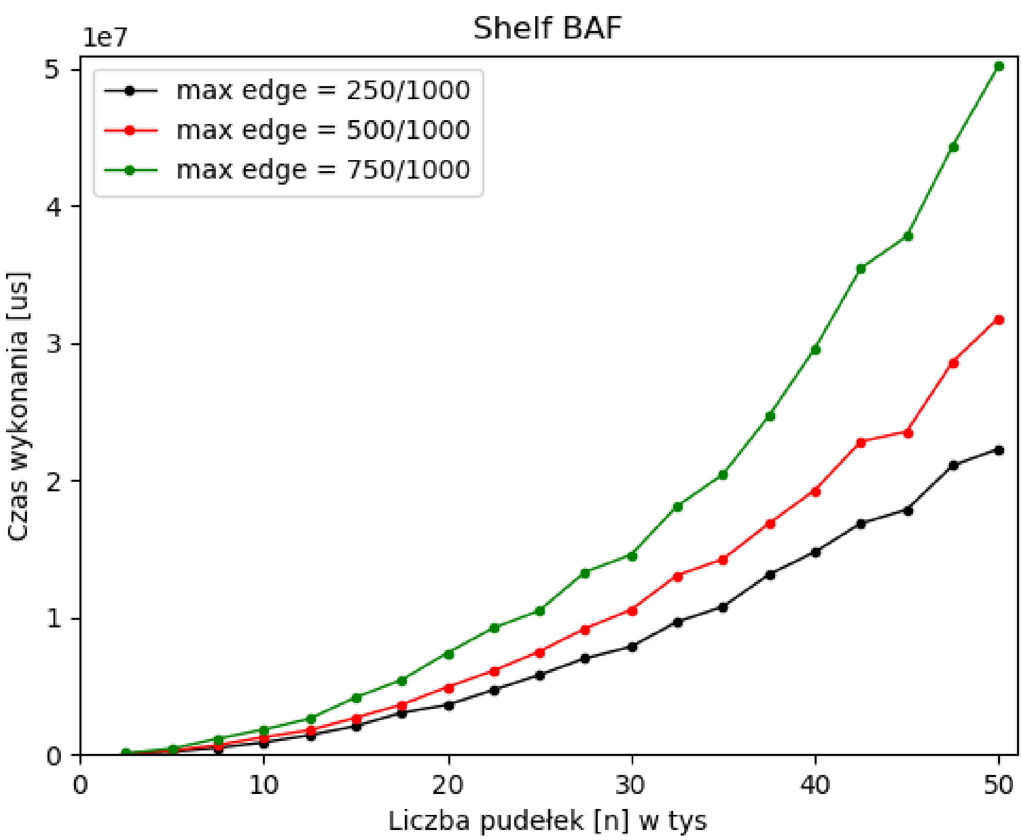
Testy algorytmu półkowego:

Niewrażliwość algorytmu na dane:

Przeprowadziłem szybki test w trybie single shot (1 iteracja = pomiar pojedynczego wykonania algorytmu) z 2 iteracjami rozgrzewkowymi oraz 10 właściwymi, losując nowe pudełka z każdą kolejną iteracją. Otrzymane wyniki:

```
Iteration   1: Shelf BAF solution---ACCURACY : 90.6267054734776---
3098195,543 us/op
Iteration   2: Shelf BAF solution---ACCURACY : 90.78382740488735---
3092996,325 us/op
Iteration   3: Shelf BAF solution---ACCURACY : 90.6447589199155---
2988372,653 us/op
Iteration   4: Shelf BAF solution---ACCURACY : 90.23383360808224---
3053703,224 us/op
Iteration   5: Shelf BAF solution---ACCURACY : 90.47191623128774---
3041812,515 us/op
Iteration   6: Shelf BAF solution---ACCURACY : 90.72411969826906---
3030679,491 us/op
Iteration   7: Shelf BAF solution---ACCURACY : 91.05868449663605---
3040172,362 us/op
Iteration   8: Shelf BAF solution---ACCURACY : 90.84518616406125---
3037714,538 us/op
Iteration   9: Shelf BAF solution---ACCURACY : 90.32995950636797---
3096410,717 us/op
Iteration  10: Shelf BAF solution---ACCURACY : 90.55996802855753---
3047456,977 us/op
```

Pomiary wskazują, że różnice zarówno w pomiarze jak i w precyzji algorytmu są minimalne, więc postanowiłem założyć niewrażliwość algorytmu na dane i w dalszej fazie testować algorytm pojedynczymi iteracjami.



Przy pomiarach tego algorytmu ze względu na jego większą złożoność zostały przyjęte następujące parametry:
-Tryb testowania: single shot (1 iteracja = pomiar pojedynczego wykonania algorytmu)
-2 iteracje do rozgrzania maszyny
-1 iteracja właściwa stanowiąca pojedynczy punkt pomiarowy

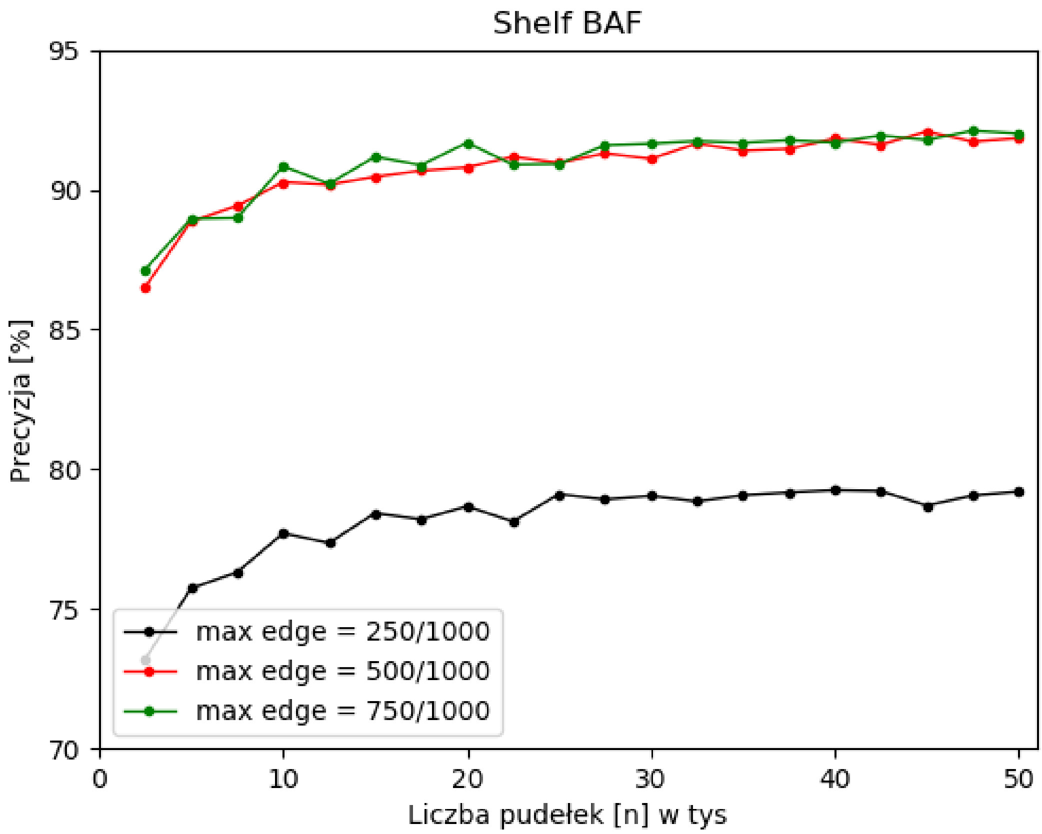
n	t(n)[ms]	q(n)
2500	44.678	0.7705428061344455
5000	212.657	0.9169016155833564
7500	487.094	0.9334118646792396
10000	885.908	0.9549303296602274
12500	1413.026	0.9747947707425856
15000	2075.863	0.9944872825618055
17500	3041.612	1.0705594250615986
20000	3621.059	0.9757951910890108
22500	4704.861	1.0017626255727576
25000	5795.544	0.9995332684622519
27500	7018.589	1.0003857285435935
30000	7862.364	0.9416590845436051
32500	9676.595	0.9875040977141292
35000	10783.176	0.9488414941565548
37500	13125.018	1.0060520166808529
40000	14766.88	0.994836765390214
42500	16852.986	1.0057315254502013
45000	17851.107	0.9502178607242718

2 pierwsze punkty pomiarowe znacząco odbiegają od q(n) = 1, jednak możemy założyć, że wynika to z niedokładności pomiaru, wynikającej z krótkiego czasu wykonania algorytmu dla tej liczby pudełek.
Przewidywana złożoność algorytmu została potwierdzona = n^2. Wynika ona z faktu, że przy każdym położeniu pudełka pojawiają się 2 nowe miejsca, które są sprawdzane przy położeniu następnego pudełka. Szacowana liczba sprawdzeń dla n pudełek wynosi:

$$\frac{(1+n)}{2} \cdot n$$

Należy także zauważyć, że dla krótszych krawędzi stała przy złożoności n^2 jest mniejsza, co prawdopodobnie wynika z faktu, że w pojedynczej półce średnio zmieścimy więcej klocków i rzadziej jesteśmy zmuszeni wykonywać operację tworzenia kolejnych półek.
Złożoność mogłaby być znacząco obniżona, jeśli zdecydowalibyśmy się usuwać wolne miejsca poniżej określonego rozmiaru (nie przeglądać ich).

Wykres pomiarów precyzji algorytmu półkowego:



Precyzja jest zdecydowanie wyższa niż w przypadku algorytmów naiwnych, lecz odbywa się ona kosztem zdecydowanie większej złożoności. Zwiększona precyzja w przypadku pudełek o dłuższych krawędziach jest spowodowana mniejszą ilością operacji *split* wewnątrz półek Y. Operacja *split* powoduje, że dopasowanie kolejnych pudełek staje się mocno utrudnione, ponieważ nie rozpatrujemy pozostałego miejsca w całości, ale jako 2 oddzielne części. Ponadto, widać niewielki wzrost precyzji wraz ze wzrostem liczby pudełek, co jest spowodowane faktem, że przy większej liczbie pudełek prawdopodobieństwo, że niewielkie wolne przestrzenie między pudełkami zostaną wypełnione jest większe, ponieważ przeglądamy wszystkie dostępne wolne miejsca przy kładzeniu pudełka wewnątrz pojemnika, zanim otworzymy kolejną półkę.

7. Testy przeszukiwania systematycznego:

W przypadku tego algorytmu mierzymy się z wyjątkowo wysoką złożonością:

$$n!\,6^n$$

Niestety złożoność algorytmu pozwala mi na zbadanie go jedynie dla wartości n do n = 6. Lepsze rezultaty możliwe by były do uzyskania w przypadku zoptymalizowania funkcji *next_orientation()*, która wyznacza kolejne możliwe ułożenie pudełka – pomimo tego że n! jest dominujący w asymptotyce, dla tak niskich n zdecydowanie większy wpływ będzie miał człon 6^n, który jest liczbą wszystkich możliwych ułożeń pudełek dla pojedynczej permutacji kolejności.

Przyjęte parametry testowania:
-Tryb single shot
-2 iteracja rozgrzewkowa
-3 iteracje właściwe
-losowanie nowych pudełek za każdą kolejną iteracją

Do pomiaru czasu wybierany jest czas najwolniejszy z trzech przeprowadzonych pomiarów.

Ze względu na bardzo małą liczbę pudełek pomiary są przeprowadzane dla maksymalnych krawędzi = 800 oraz 1000.

n	max=800 t(n)[us]	q(n)
3	907	1.4106550437427487
4	18432	1.1944686588692734
5	459933	0.9935177113710242
6	20788187	1.2473696030963495

n	max=1000 t(n)[us]	q(n)
3	970	1.4627324282801422
4	16718	1.0504278666661262
5	476660	0.9983190711111292
6	20702266	1.2044146866282226

Uwzględniając jak mało jest punktów pomiarowych wyniki można uznać za satysfakcjonujące. Złożoność jest na poziomie $\sim n!\,6^n$ dla badanych n.

8. Porównanie precyzji algorytmów:

Na koniec przeprowadziłem proste porównanie działania algorytmów, które pozwoliłoby w lepszy sposób zobrazować skuteczność przeszukiwania systematycznego.
10-krotny pomiar precyzji, za każdym razem wykonany dla innych danych przez 3 algorytmy: naiwny, półkowy oraz systematycznego przeszukiwania dla pudełek n = 5. Maksymalna krawędź pudełka wynosi 1000/1000, tak aby pomiary jak najlepiej oddały działanie algorytmu przeszukiwania systematycznego.

Algorytm	Średnia precyzja dla 10 pomiarów (n=5)
Naiwny	44.86%
Półkowy	58.54%
Systematyczne przeszukiwanie	67.60%

Przy każdym pomiarze algorytm systematycznego przeszukiwania uzyskiwał lepszy bądź taki sam rezultat jak pozostałe algorytmy.

9. Podsumowanie wyników:

Najlepsze rezultaty pod względem precyzji, zapewni nam przeszukiwanie systematyczne, jednak przeszukiwanie wszystkich możliwości bez zastosowania żadnej heurystyki jest zbyt czasochłonne, żeby mogło mieć jakiekolwiek zastosowanie w świecie rzeczywistym. Algorytmy naiwne, natomiast przy odpowiednio dopasowanych sektorach są w stanie zapewnić precyzję w granicach 50%, natomiast wynik otrzymujemy natychmiastowo. Mogłoby to mieć zastosowanie w przypadku wycinki z bardzo taniego materiału, gdzie nie mamy czasu na obliczenia. Algorytm półkowy w wersji ze znajdowaniem najlepszych miejsc wydaje się najbardziej użytecznym algorytmem, ponieważ zapewnia wysoką precyzję przy umiarkowanej złożoności (kilkanaście tysięcy pudełek w kilka sekund).

Należy pamiętać, że zaimplementowane algorytmy i przeprowadzone testy są tylko kroplą w morzu problemu plecakowego oraz wyciągnięte tutaj wnioski są właściwe jedynie dla początkowo przyjętych założeń.

10. Wizualizacja danych

Prosta wizualizacja z wykorzystaniem biblioteki JavaFX. Możliwość obejrzenia jak poszczególne pudełka zostały rozmieszczone w pojemniku za pomocą ruchomej kamery.