INSTYTUT INFORMATYKI
WYDZIAŁ INFORMATYKI
POLITECHNIKA POZNAŃSKA

PRACA DYPLOMOWA MAGISTERSKA

PROBLEM PAKOWANIA

- BIBLIOTEKA ALGORYTMÓW

inż. Marcin ROBASZYŃSKI

Promotor:

dr hab. inż. Małgorzata STERNA

Poznań, 2011 r.

KARTA PRACY DYPLOMOWEJ

SPIS TREŚCI

1. WSTĘP

We współczesnych czasach w praktycznie wszystkich gałęziach gospodarki poszukuje się sposobów na zwiększenie zysków (i minimalizację strat). W dawnych czasach, gdy towary i usługi nie były łatwo dostępne, wystarczyło zwiększyć produkcję danego towaru czy też zasięg świadczonych usług. Taką możliwość dawały głównie inwestycje w nowe technologie, które pozwalały produkować szybciej, taniej i wydajniej. Powstały wyspecjalizowane linie produkcyjne, rozwinął się też transport. Często jednak trudno znacząco zwiększać produkcję; poza tym rynek w wielu dziedzinach jest nasycony i osiągnięcie wyższej sprzedaży produktów lub usług jest bardzo trudne.

Zamiast tego, ludzie i firmy coraz częściej skupiają się na oszczędzaniu oraz jak najlepszym wykorzystaniu dostępnych zasobów. Jest to szczególnie istotne w dzisiejszych czasach, gdy coraz więcej mówi się o ekologii, efektywniejszej gospodarce coraz trudniej dostępnymi złożami surowców naturalnych i problemach finansowych kolejnych krajów.

Okazuje się jednak, że rozwój technologii często nie nadąża za rzeczywistymi potrzebami. Poza tym, nowe rozwiązania są z reguły drogie i trudne do wykorzystania na większą skalę. Alternatywą jest poprawa wykorzystywanych już sposobów działania (produkcji lub realizacji usług).

Dobrym przykładem takiego działania jest np. ładowanie w przedsiębiorstwach transportowych takiej samej ilości towarów do mniejszej liczby pojazdów, skrzynek czy kontenerów. Znacząco obniża to koszty transportu, zwiększając w ten sposób końcowy zysk i konkurencyjność firmy. Inny przykład stanowią linie produkcyjne na których wycina się pewne części z większej ilości materiału (np. z płatów blachy). W takim wypadku wykorzystanie pozostałych fragmentów surowca często jest drogie lub wręcz niemożliwe ze względu na utratę podczas obróbki właściwości wymaganych przez proces technologiczny.

Podobnych przykładów można by wymienić bardzo wiele. Większość z nich można opisać jako wypełnianie pewnych pojemników elementami. Oczywiście istnieją modele formalne i algorytmy, rozwiązujące podobne problemy praktyczne. W zależności od tego, co jest głównym celem, wyróżnić można m.in. problemy

plecakowe (wymagające umieszczenia w plecaku o określonej pojemności elementów o jak największej sumarycznej wartości), problemy pokrycia (polegające na wypełnieniu - do pełna - jak największej liczby pojemników dostępnymi elementami, przy czym dopuszczalne jest przekroczenie pojemności pojemnika) czy problemy pakowania (związane z umieszczeniem dostępnych elementów w jak najmniejszej liczbie pojemników).

Niestety rozwiązanie tych problemów w większości przypadków nie jest trywialne. Z tego powodu obecnie wykorzystuje się narzędzia współczesnej informatyki, które pozwalają znajdować nowe efektywne metody, wykorzystywać i poprawiać te znane już od dawna oraz porównywać ze sobą jedne i drugie.

1.1. Cel i zakres pracy

Celem pracy było stworzenie biblioteki algorytmów rozwiązujących jednowymiarowy problem pakowania oraz porównanie ich z jej wykorzystaniem. Z tego względu system musiał posiadać moduł eksperymentu obliczeniowego, umożliwiającego testy dla znacznej liczby większych instancji. Miał on też umożliwiać porównanie wybranych algorytmów ze sobą i ich prostą analizę za pomocą wykresów.

Poza przeprowadzaniem eksperymentów obliczeniowych system ma umożliwiać prezentowanie zasady działania poszczególnych algorytmów. Z tego względu musi on posiadać moduł wizualizacji, prezentujący poszczególne kroki w sposób graficzny, oraz możliwość wprowadzania własnych instancji problemu (np. w celu pokazania zachowania algorytmu w szczególnym przypadku). Należało również umożliwić odczyt najpopularniejszych formatów plików, zawierających instancje problemu pakowania.

Niniejsza praca zawiera prezentację problemu pakowania oraz opis podstawowych algorytmów go rozwiązujących, uzupełniony eksperymentami obliczeniowymi wykonanymi z użyciem stworzonego w ramach pracy systemu o nazwie *Bin Packing*. Ponadto przedstawia projekt koncepcyjny i techniczny aplikacji oraz opis implementacji systemu.

Struktura pracy jest następująca. W rozdziale 2. sformułowano problem pakowania w sposób formalny oraz opisano jego różne warianty i zastosowania

praktyczne wraz z przykładami. Oprócz tego przytoczono również dowód NP-trudności problemu i opisano jego konsekwencje.

W rozdziale 3. opisano poszczególne algorytmy rozwiązujące problem pakowania zaimplementowane w prezentowanej bibliotece *Bin Packing*: *Next-Fit*, *First-Fit*, *Best-Fit*, *First-Fit Decreasing*, *Best-Fit Decreasing*, *Random-Fit*, algorytm dokładny, algorytm redukcji, asymptotyczny schemat aproksymacyjny oraz algorytm *Next-Fit* z dodatkową optymalizacją. Dla każdego z nich przedstawiono ideę działania oraz wady i zalety stosowanego podejścia. Podano również złożoność obliczeniową oraz oszacowanie jakości (jeżeli jest znane). Dodatkowo zilustrowano zasadę działania poszczególnych metod dla instancji testowej.

Na opisie samego systemu skupiono się w rozdziale 4. Zawarto w nim podstawowe informacje na temat biblioteki algorytmów *Bin Packing*. Opisano również przeznaczenie oraz sposób korzystania z poszczególnych elementów interfejsu. Poza tym przedstawiono aspekty techniczne, w tym architekturę systemu oraz wymagania, które przed nim postawiono. Również w tym rozdziale znajduje się opis wspieranych typów plików oraz sposobu reprezentacji elementów systemu w pamięci komputera.

Eksperyment obliczeniowy przeprowadzony z użyciem biblioteki *Bin Packing* wraz z wynikami i ich analizą znajdują się w rozdziale 5.

W rozdziale 6. ujęto zgromadzone podczas pracy nad systemem doświadczenia, opis tego, co udało się osiągnąć (a czego nie) oraz wnioski z przeprowadzonych badań.

2. PROBLEM PAKOWANIA

2.1. Sformułowanie problemu pakowania

Istnieje wiele wariantów problemu pakowania. W niniejszej pracy skupiono się na najprostszej jednowymiarowej wersji problemu. Oznacza to, że pod uwagę bierze się tylko 1 wymiar (z reguły jest to wysokość) elementów. Dodatkowo założono, że wartość określająca rozmiary pudełek oraz wartości opisujące rozmiary elementów są liczbami całkowitymi dodatnimi. Nie powoduje to utraty ogólności – wystarczy przeskalować poszczególne wartości aby uzyskać równoważny problem oparty o wartości rzeczywiste dodatnie.

Definicja (jednowymiarowego) problemu pakowania:

Dany jest zbiór N składający się z n elementów o (dodatnich) rozmiarach w_i $(i = 1, \dots, n)$ oraz m pudełek o ustalonej pojemności c. Należy zapakować elementy do pudełek nie przekraczając ich pojemności w taki sposób, aby liczba wykorzystanych pudełek była minimalna.

Przytoczony powyżej problem pakowania można sformułować w postaci zadania programowania liniowego całkowitoliczbowego:

$$\min \quad z = \sum_{j=1}^{m} y_j$$

przy ograniczeniach:

$$\sum_{i=1}^{n} w_j x_{ij} \leq c, \qquad j = 1, \dots, m$$

$$\sum_{j=1}^{m} x_{ij} = 1, \qquad i = 1, \dots, n$$

$$y_j \geq x_{ij}, \qquad i = 1, \dots, n \quad j = 1, \dots, m$$

$$y_j \in \{0,1\}, \qquad j = 1, \dots, m$$

$$x_{ij} \in \{0,1\}, \qquad i = 1, \dots, n \quad j = 1, \dots, m$$

$$c \in \mathbb{Z}^+,$$

$$w_i \in \mathbb{Z}^+, \qquad i = 1, \dots, n$$
$$w_i \leq c, \qquad i = 1, \dots, n$$

gdzie

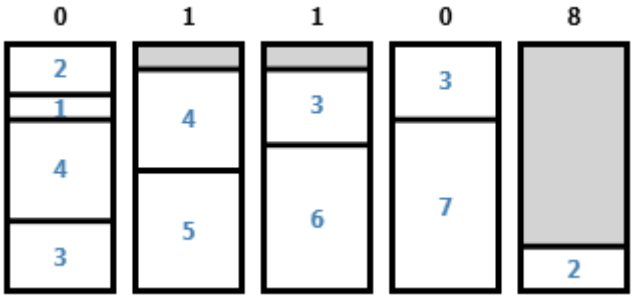
$$y_j = \begin{cases} 1, & jeśli\ pudełko\ j\ zostało\ wykorzystane \\ 0, & w\ przeciwnym\ wypadku \end{cases}$$
$$x_{ij} = \begin{cases} 1, & jeśli\ element\ i\ został\ umieszczony\ w\ pudełku\ j \\ 0, & w\ przeciwnym\ wypadku \end{cases}$$

Istnieją również inne warianty problemu pakowania, obejmujące więcej wymiarów. Najczęściej spotykany jest wariant dwuwymiarowy, w którym należy umieścić prostokątne elementy o różnych rozmiarach w prostokątnych pudełkach.

Dosyć oczywiste jest, że minimalizując liczbę wykorzystanych pudełek pośrednio minimalizuje się również pozostałą w nich wolną przestrzeń. Można więc spojrzeć na problem inaczej – należy wyciąć z fragmentów materiału o zadanej wielkości określone elementy, wykorzystując jak najmniej materiału.

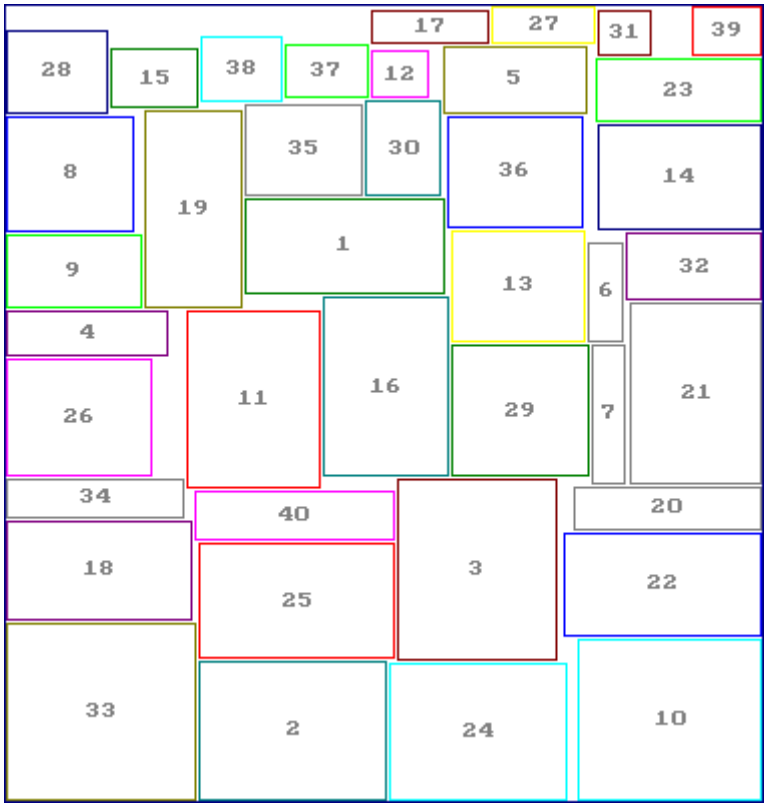
Tak opisany problem to przykład tzw. problemu rozkroju (ang. *cutting problem*). W zależności od wariantu problemu rozkroju jego celem może być minimalizacja niewykorzystanego materiału bądź też maksymalizacja liczby wykonanych (wyciętych) elementów.

Na poniższych rysunkach przedstawiono przykładowe rozwiązania jedno-, dwu- oraz trójwymiarowego problemu pakowania. Pierwszy z nich (rys. 2.1.) przedstawia rozwiązanie 1-wymiarowego problemu pakowania, składającego się 11 elementów. Rozmiar pudełek to 10. Poszczególnym elementom odpowiadają białe prostokąty; niebieskie liczby oznaczają natomiast ich rozmiar. Prostokąty, w których umieszczono elementy odpowiadają pudełkom. Szarym kolorem oznaczono pozostałą wolną w pudełkach przestrzeń. Jej wartość liczbową reprezentują czarne liczby znajdujące się nad każdym z pudełek.



Rys. 2.1. Przykładowe rozwiązanie 1-wymiarowego problemu pakowania

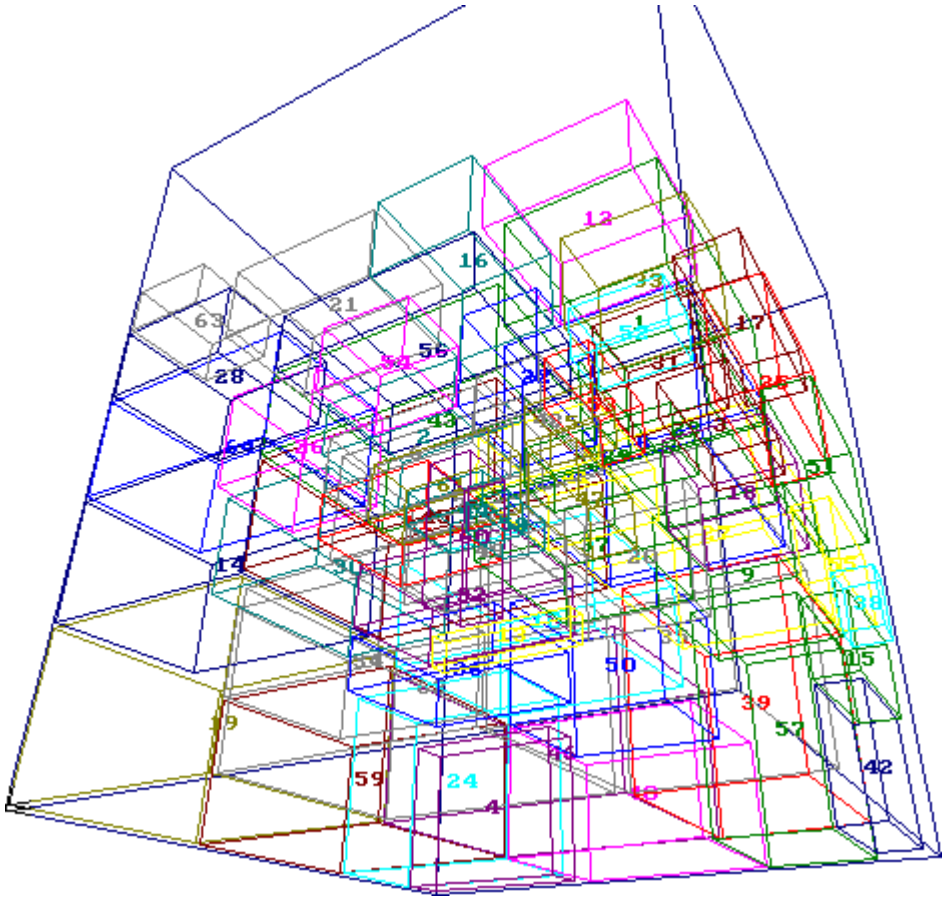
Rys. 2.2. przedstawia przykładowe rozwiązanie dwuwymiarowego problemu pakowania. W tym przypadku nie podano rozmiarów elementów ani pudełek. Poszczególne liczby określają numer elementu (w sumie jest ich 40). Wszystkie elementy umieszczono w jednym pudełku.



Rys. 2.2. Przykładowe rozwiązanie 2-wymiarowego problemu pakowania

(źródło: http://www.astrokettle.com)

Ostatni rysunek (2.3), prezentuje przykładowe rozwiązanie trójwymiarowego problemu pakowania. W tym przypadku poszczególne liczby również reprezentują numery konkretnych elementów. Wymiary pudełek i elementów nie zostały podane.



Rys. 2.3. Przykładowe rozwiązanie 3-wymiarowego problemu pakowania

(źródło: http://www.astrokettle.com)

Problem pakowania znajduje zastosowanie wielu dziedzinach, np. w transporcie – często istnieje potrzeba załadowania towarów (przedmiotów) o określonych rozmiarach do jak najmniejszej liczby kontenerów lub samochodów dostawczych. Wiele zastosowań problem pakowania ma również w informatyce. Dobrym przykładem jest zagadnienie przydziału pamięci w systemach stosujących stronicowanie pamięci lub transmisja danych. Umieszczenie porcji danych do wysłania w mniejszej liczbie pakietów pozwala skrócić czas transmisji, jak również obniżyć jej koszt gdy jest ona płatna. Inne zastosowanie problemu pakowania to, często spotykany w przemyśle problem cięcia materiału – polega on na efektywnym cięciu materiału (bądź wycinaniu

z niego mniejszych części), dzięki czemu zmniejsza się ilość materiału niewykorzystanego.

2.2. Dowód NP-trudności

Problem pakowania jest problemem trudnym do rozwiązania. Jest to spowodowane tym, że umieszczając kolejne elementy instancji problemu pakowania w pudełkach w zdecydowanej większości przypadków nie można jednoznacznie stwierdzić, w którym pudełku najlepiej umieścić rozważany element. Niełatwe jest również uzyskanie poprawy istniejącego już rozwiązania nie będącego rozwiązaniem optymalnym – wymaga to przeniesienia wszystkich elementów jednego (lub kilku) pudełek do pozostałych pudełek, co wiąże się najczęściej z koniecznością przestawiania elementów również pomiędzy nimi.

Okazuje się, że problem pakowania w wersji decyzyjnej jest silnie NP-zupełny [GARE 1979].

DOWÓD:

Jednym ze znanych problemów silnie NP-zupełnych jest tzw. problem trójpodziału (ang. *3-partition problem*) [GARE 1979], [PAPA 1982]:

Dany jest zbiór elementów $A = \{a_1, a_2, \dots, a_{3n}\}$ o rozmiarach $s(a_i)$, dla $i = 1, \dots, 3n$ oraz ograniczenie B. Ponadto, dla każdego i zachodzi: $\frac{1}{4}B < s(a_i) < \frac{1}{2}B$ oraz: $\sum_{i=1}^{3n} s(a_i) = nB$.

Czy istnieje podział zbioru A na n rozłącznych podzbiorów $S_1, S_2, \dots, S_n$ taki, że $\sum_{a_i \in S_j} s(a_i) = B$ dla $j = 1, \dots, n$?

Wystarczy zauważyć, że problem trójpodziału jest szczególnym przypadkiem problemu pakowania, w którym elementom a_i o rozmiarze $s(a_i)$ w tym problemie odpowiadają elementy a_i w problemie pakowania o rozmiarze:

$$s'(a_i) = \frac{ns(a_i)}{\sum_{a_i \in A} s(a_i)}, \quad i = 1, \dots, 3n$$

Zbiory $S_1, S_2, \dots, S_n$ wyznaczają natomiast zawartość n pudełek, każde o łącznym rozmiarze wynoszącym 1. W tym przypadku pytamy o istnienie rozwiązania problemu pakowania o liczbie pudełek nie przekraczającej n. Jeżeli takie rozwiązanie istnieje, to będzie się składało z n całkowicie wypełnionych pudełek.

Powyższe spostrzeżenie udowadnia silną NP-zupełność wersji decyzyjnej problemu pakowania, a tym samym silną NP-trudność problemu pakowania. Oznacza to, że nie istnieje algorytm wielomianowy, znajdujący rozwiązanie optymalne w ogólnym przypadku. Z tego powodu dla problemu pakowania stosuje się heurystyki, znajdujące rozwiązanie przybliżone. Ponadto zaproponowano również asymptotyczne schematy aproksymacyjne, które gwarantują znalezienie rozwiązania o zadanej dokładności. Wraz ze wzrostem dokładności zwiększa się również czas obliczeń. Zaletą takiego podejścia jest możliwość ustalenia kompromisu pomiędzy dokładnością a czasem obliczeń.

2.3. Przegląd literatury

Problem pakowania pojawił się w literaturze kilka dekad temu (lata siedemdziesiąte). Od tego czasu powstało wiele publikacji na ten temat, przedstawiających różne sposoby rozwiązania problemu pakowania.

Jeden z pierwszych algorytmów dokładnych został zaproponowany przez Eilona i Christofidesa [EILO 1971]. Inny algorytm dokładny zaproponowali Martello i Toth [MART 1990]. Poza algorytmem dokładnym zaproponowali oni również nowe dolne ograniczenia oraz procedurę redukcji, które wykorzystywał do zmniejszenia przestrzeni poszukiwań.

Inny algorytm dokładny został zaprezentowany przez Fukanagę i Korfa [FUKA 2007]. W odróżnieniu od poprzednich algorytmów, autorzy proponują rozwiązanie przeszukujące przestrzeń możliwych sposobów wypełnienia pudełek (a nie miejsc, w których można umieścić pojedynczy element).

W 1981 Fernandez de la Vega i Lueker w swojej pracy przedstawili wielomianowy schemat aproksymacji (ang. *polynomial-time approximation scheme*,

w skrócie PTAS) [FERN 1981]. Powstało również wiele metod heurystycznych. Najpopularniejsze z nich to algorytmy listowe, często wykorzystywane w pracach.

3. ALGORYTMY

3.1. Wstęp

W poniższym rozdziale przedstawiono algorytmy zaimplementowane w bibliotece *Bin Packing*. Najliczniejszą grupę stanowią heurystyki listowe. Ich zasada działania opiera się na pobieraniu z wejścia pojedynczych elementów i dokładaniu ich (w kolejności pobrania z wejścia) do aktualnego rozwiązania wg ściśle określonych zasad. Ich główną zaletą jest prostota.

Poza najbardziej znanymi algorytmami listowymi zaimplementowano również algorytm dokładny, oparty na metodzie podziału i ograniczeń (ang. *branch and bound*). Jego zaletą jest znajdowanie rozwiązań optymalnych, wadą natomiast bardzo długi czas działania, wykluczający praktyczne zastosowanie tej metody w większości przypadków.

Kolejnym z algorytmów zaimplementowanych w bibliotece *Bin Packing* jest asymptotyczny schemat aproksymacyjny (ang. *asymptotic approximation scheme*, *AAS*), wykorzystujący programowanie liniowe. Metoda ta wyróżnia się spośród innych możliwością określenia dokładności uzyskanego rozwiązania za pomocą dodatkowego parametru. Wzrost dokładności powoduje jednak wydłużenie czasu obliczeń.

Większość znanych algorytmów skupia się na umieszczaniu pojedynczych elementów w pudełkach w optymalny sposób. Jest to działanie zorientowane na element (ang. *item-oriented*). Istnieje również, znacznie mniej liczna, grupa algorytmów zorientowanych na pudełko (ang. *bin-oriented*). Ich zasada działania opiera się na znajdowaniu jak najlepszych sposobów wypełnienia pudełka dostępnymi elementami i dodawaniu do rozwiązania wypełnionych już w dany sposób pudełek. Wadą tych rozwiązań jest duża złożoność obliczeniowa związana z wyszukiwaniem sposobów zapakowania pudełek, przez co często rozważa się tylko sposoby zapakowania zawierające tylko niewielką liczbę elementów. Biblioteka *Bin Packing* zawiera jeden z algorytmów tego typu – tzw. algorytm redukcji.

Ostatnią metodą, którą zaimplementowano jest algorytm następnego dopasowania z dodatkową optymalizacją, którego zasada działania opiera się na próbie poprawy rozwiązania uzyskanego przez najszybszy z algorytmów listowych poprzez wykorzystanie algorytmu redukcji.

3.2. Dolne ograniczenia

Ze względu na to, że w większości przypadków optymalne rozwiązanie problemu pakowania nie jest znane, w celu oszacowania jakości uzyskiwanych rozwiązań stosuje się dolne ograniczenia. Poza tym są one wykorzystywane również w niektórych algorytmach, np. w metodzie podziału i ograniczeń (ang. *Branch & Bound*).

W omawianym systemie zaimplementowano 2 dolne ograniczenia wartości funkcji kryterialnej problemu pakowania, znane z literatury [MART 1990]. Pierwsze z nich, L_1 jest bardzo proste – jego wartość stanowi cecha górna (lub sufit) sumy rozmiarów wszystkich elementów, podzielonej przez wielkość pudełka:

$$L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$$

Niestety ograniczenie to nie sprawdza się w wielu przypadkach – najlepiej sobie radzi gdy instancja problemu zawiera głównie małe elementy. W przypadku większych elementów obliczony wynik może być znacznie niższy od rzeczywistej wartości funkcji kryterialnej.

Drugie dolne ograniczenie, L_2, zdefiniowane jest jako:

$$L_2 = max\left\{L(\alpha): 0 \le \alpha \le \frac{c}{2}, \alpha \in \mathbb{Z}\right\}$$

gdzie $L(\alpha)$ oblicza się na podstawie zbiorów J_1, J_2, J_3:

$$J_1 = \{j \in N: w_j > c - \alpha\},$$
$$J_2 = \left\{j \in N: c - \alpha \ge w_j > \frac{c}{2}\right\},$$
$$J_3 = \left\{j \in N: \frac{c}{2} \ge w_j \ge \alpha\right\}$$

oraz parametru α jako:

$$L(\alpha) = |J_1| + |J_2| + max\left(0, \left\lceil \frac{\sum_{j \in J_3} w_j - \left(|J_2| c - \sum_{j \in J_2} w_j\right)}{c} \right\rceil\right)$$

Martello i Toth wykazali również, że w celu wyznaczenia wartości L_2 wystarczy obliczać kolejno $L(\alpha)$ tylko dla wartości α równych unikalnym wartościom elementów $w_j \leq c/2 \quad (j = 1, \dots, n)$, posortowanych malejąco. Ponadto, często nie trzeba przeprowadzać obliczeń dla wszystkich wartości w_j spełniających podaną nierówność. Obliczenia można przerwać gdy spełniona będzie nierówność:

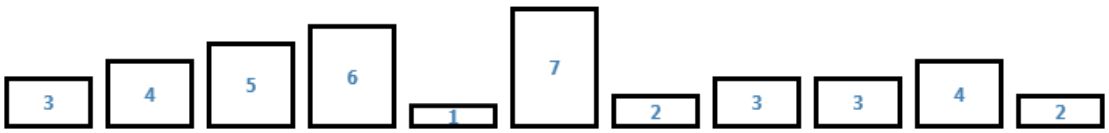
$$L' = |J_1| + |J_2| + \left\lceil \frac{\sum_{j=j^*}^{n} w_j - \left(|J_2|c - \sum_{j \in J_2} w_j\right)}{c} \right\rceil \leq L_2^*$$

gdzie L_2^* oznacza największą wartość $L(w_j)$ obliczoną do tej pory, a j^* zdefiniowane jest jako:

$$j^* = min\left\{j \in N : w_j \leq \frac{c}{2}\right\}$$

Przykład:

Poniżej przedstawiono obliczenia dolnych ograniczeń dla przykładowej instancji problemu pakowania. Składa się ona z 11 elementów; wielkość pudełka – 10. Przedstawiono ją na rysunku 3.1. Poszczególne prostokąty reprezentują pojedyncze elementy. Liczby wewnątrz prostokątów odpowiadają rozmiarom elementów.



Rys. 3.1 Przykładowa instancja problemu

W celu skrócenia zapisu przedstawiono tylko główne obliczenia, pomijając wyznaczanie zbiorów J_1, J_2, J_3, itp.

Po posortowaniu elementów otrzymujemy zbiór danych wejściowych:

$n = 11, c = 10, N = \{7,6,5,4,4,3,3,3,2,2,1\}, \alpha \in \{5,4,3,2,1\}$

$$L_2^* = L_1 = \left\lceil \frac{40}{10} \right\rceil = 4$$

$$L(5) = 2 + 0 + max\left(0, \left\lceil \frac{5-0}{10} \right\rceil\right) = 3$$

$$L' = 2 + 0 + \left\lceil \frac{27 - 0}{10} \right\rceil = 5 > L_2^*$$

$$L(4) = 1 + 1 + \max\left(0, \left\lceil \frac{13 - (10 - 6)}{10} \right\rceil\right) = 3$$

$$L' = 1 + 1 + \left\lceil \frac{27 - (10 - 6)}{10} \right\rceil = 5 > L_2^*$$

$$L(3) = 0 + 2 + \max\left(0, \left\lceil \frac{22 - (20 - 13)}{10} \right\rceil\right) = 4$$

$$L_2^* = L(3) = 4$$

$$L' = 0 + 1 + \left\lceil \frac{27 - (20 - 13)}{10} \right\rceil = 3 \leq L_2^*$$

W związku z tym, że obliczona wartość $L' \leq L_2^*$ można zakończyć obliczenia i przyjmując $L_2 = L_2^* = 4$.

3.3. Algorytm dokładny

Zaimplementowany w bibliotece *Bin Packing* algorytm dokładny opiera się na metodzie podziału i ograniczeń [BŁAŻ 1982] i stanowi zmodyfikowaną wersję algorytmu dokładnego opisanego w [MART 1990]. W pierwszym kroku elementy są sortowane wg malejących rozmiarów. Przebieg obliczeń można zilustrować za pomocą drzewa przeszukiwań problemu. W poszczególnych krokach następuje podział problemu reprezentowany przez węzły drzewa. W każdym węźle generowane są nowe rozwiązania poprzez umieszczenie aktualnie rozważanego elementu kolejno we wszystkich otwartych już pudełkach, w których jest wystarczająca ilość miejsca aby go pomieścić oraz w jednym, nowym pudełku. Liście drzewa przeszukiwań problemu reprezentują rozwiązania dopuszczalne.

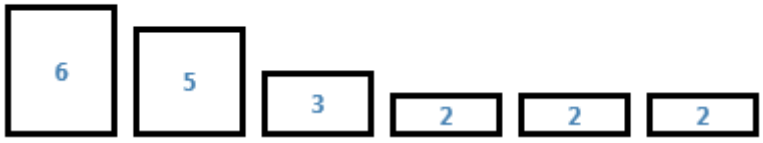
W celu ograniczenia liczby węzłów w drzewie stosuje się tzw. odcięcia – polegają one na odrzuceniu gałęzi, w których na pewno nie zostaną znalezione rozwiązania lepsze od najlepszego znalezionego do tej pory rozwiązania. Liczba elementów tego rozwiązania wyznacza górne ograniczenie $\bar{z}$. W omawianym algorytmie stosuje się 2 rodzaje odcięć. Pierwsze z nich pomija gałęzie umieszczające element w nowym pudełku, dla których

aktualna liczba pudełek wynosi $\bar{z}$ lub $\bar{z} - 1$ (w tym przypadku dodanie nowego pudełka może doprowadzić do rozwiązania co najwyżej tak samo dobrego jak już znalezione). W każdym węźle obliczana jest też wartość dolnych ograniczeń L_2 i L_3 [MART 1990]. Jeżeli wartość któregoś z nich jest większa od najmniejszej obliczonej dotychczas, to dana gałąź nie jest przeglądana. Oznacza to bowiem, że przeszukując daną gałąź możemy znaleźć rozwiązanie co najwyżej tak samo dobre jak przeszukując gałąź, dla której obliczona wartość dolnego ograniczenia była niższa. Jest to drugie stosowane odcięcie. Za każdym razem, gdy obliczone dolne ograniczenia są mniejsze od aktualnie najmniejszych, stają się one nowymi najlepszymi ograniczeniami.

Dodatkowo, gdy wartość nowo znalezionego rozwiązania dopuszczalnego odpowiada wartości dolnego ograniczenia, to obliczenia algorytmu zostają zakończone, gdyż oznacza to znalezienie rozwiązania optymalnego.

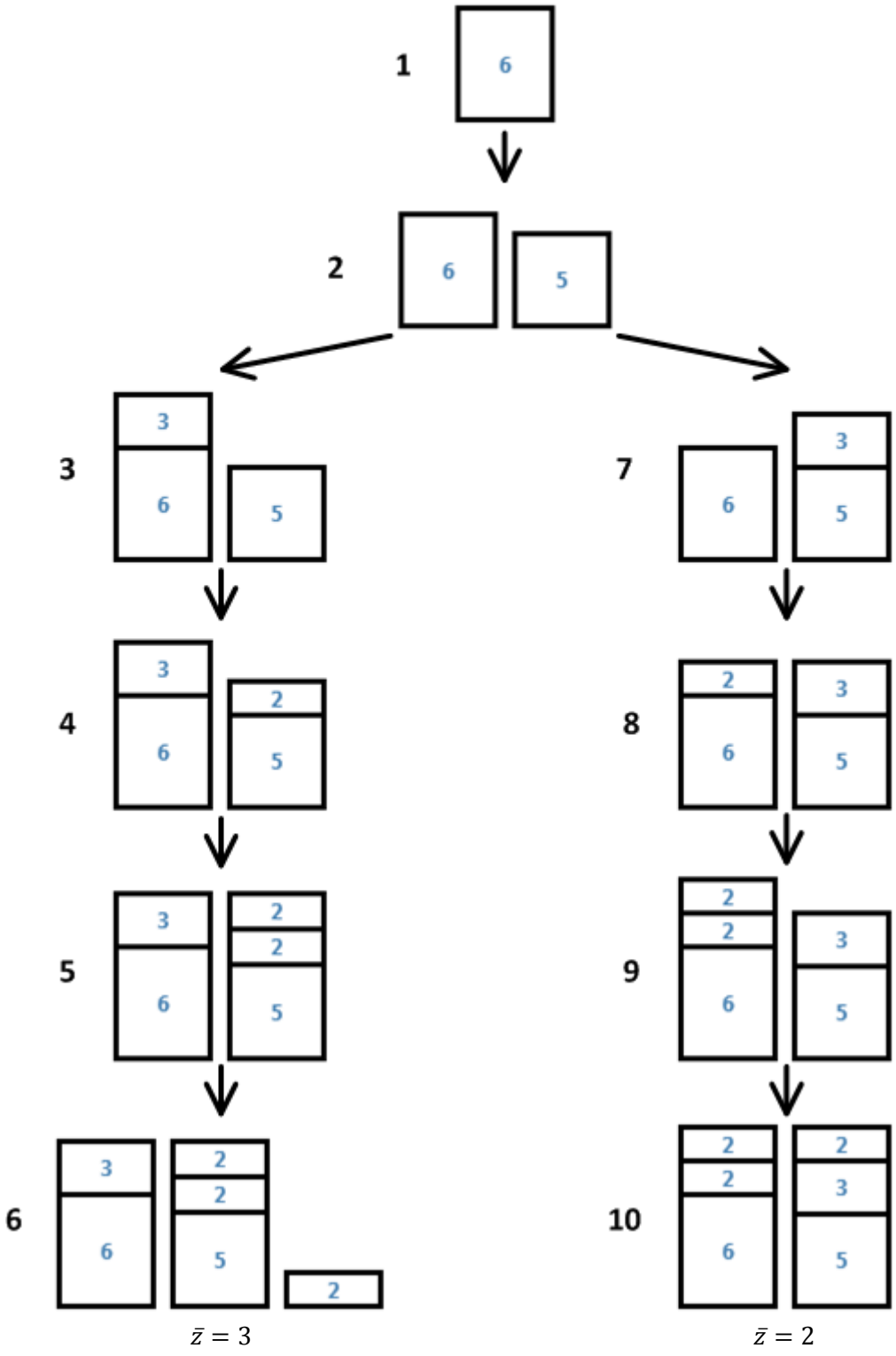
Modyfikacja w stosunku do oryginalnego algorytmu przedstawionego przez Martello i Totha polega na pominięciu wykorzystania tzw. kryterium dominacji.

Na rysunku 3.3. przedstawiono drzewo przeszukiwań problemu dla przykładowej instancji z poniższego rysunku (rozmiar pudełka to 10).



Rys. 3.2. Przykładowa instancja problemu pakowania dla algorytmu dokładnego

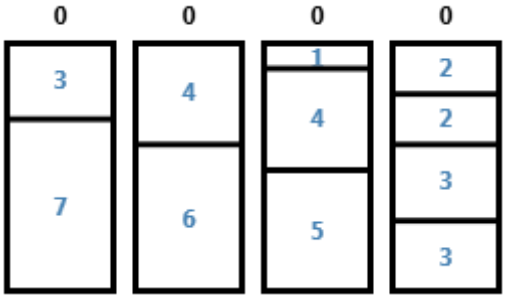
Liczby obok poszczególnych węzłów przedstawiają kolejność ich generowania. Wartości dolnych ograniczeń zostały pominięte, ponieważ drzewo przeszukiwań problemu nie zawiera żadnego węzła, dla którego znaleziono wartość dolnego ograniczenia niższą niż obliczona na początku działania algorytmu.



Rys. 3.3. Drzewo rozwiązań dla instancji z rys. 3.2.

W krokach 1-6 algorytm znajduje pierwsze poprawne rozwiązanie, odpowiadające rozwiązaniu problemu metodą *First-Fit Decreasing*. Przeszukiwanie jest kontynuowane ze względu na wartość dolnych ograniczeń niższą od liczby pudełek w uzyskanym rozwiązaniu. Dla węzłów 3-5 drugi węzeł (umieszczający dodawany element w osobnym pudełku) nie jest generowany, ponieważ uzyskane w ten sposób rozwiązanie nie mogło by być lepsze od aktualnego (składałoby się z przynajmniej 3 pudełek). W kolejnych krokach (7-10) uzyskiwane jest drugie rozwiązanie. W tym przypadku liczba pudełek odpowiada wartości najmniejszego dolnego ograniczenia, więc algorytm jest przerywany. W związku z tym kolejne węzły nie są generowane.

Dla instancji testowej z rys. 3.1. algorytm daje (oczywiście) rozwiązanie optymalne, składające się z 4 pudełek:



Rys. 3.4. Wynik działania algorytmu dokładnego dla instancji testowej

3.4. Algorytmy listowe

Algorytmy listowe stanowią najbardziej znaną i często spotykaną w literaturze grupę algorytmów, rozwiązujących problem pakowania. Pobierają one z wejścia pojedyncze elementy i dokładają je w kolejności pobrania z wejścia do aktualnego (częściowego) rozwiązania problemu.

Większość z algorytmów listowych rozważa elementy wejściowe w takiej kolejności, w jakiej umieszczono je „na wejściu”. W związku z tym nie wymagają one znajomości wszystkich elementów *a priori*. Metody takie nazywane są metodami typu *on-line*. Ich największa wadą jest duża zależność jakości osiąganych rozwiązań od kolejności elementów. Istnieją również inne algorytmy listowe, nazywane metodami typu *off-line*, które wymagają znajomości całej listy elementów *a priori*.

Ocena jakości uzyskiwanych rozwiązań

Do oceny jakości rozwiązań uzyskiwanych przez algorytmy listowe wykorzystuje się często tzw. asymptotyczne oszacowanie najgorszego przypadku (ang. *asymptotic worst-case performance ratio*) [MART 1990]. Dla algorytmu aproksymacyjnego A definiuje się go jako minimalną liczbę rzeczywistą $r^{\infty}(A)$ taką, że dla pewnej całkowitej dodatniej liczby k:

$$\frac{A(I)}{z(I)} \leq r^{\infty}(A)$$

dla wszystkich instancji I, spełniających warunek $z(I) \geq k$, gdzie $z(I)$ oznacza liczbę pudełek w rozwiązaniu optymalnym, a $A(I)$ liczbę pudełek w rozwiązaniu uzyskanym przez dany algorytm.

3.4.1. Algorytm następnego dopasowania (ang. *Next-Fit*)

Najprostszym (i zarazem najszybszym) z zaimplementowanych algorytmów jest algorytm następnego dopasowania (ang. *Next-Fit*). Zasada jego działania opiera się na umieszczaniu kolejnych elementów w aktualnym pudełku, dopóki pozwala na to ilość wolnego miejsca. W przypadku gdy w pudełku nie ma już miejsca pozwalającego umieścić w niej aktualny element, dodawana jest nowe pudełko i aktualny, oraz w miarę możliwości, kolejne elementy są umieszczane w nowym pudełku. Cały proces jest powtarzany aż do wykorzystania wszystkich elementów.

Algorytm *Next-Fit*:

k - numer aktualnie wybranego pudełka
s - suma elementów znajdujących się w aktualnie wybranym (k-tym) pudełku

1. $k = 1, s = 0$

2. dla każdego i od 1 do n wykonaj kroki 3 – 5

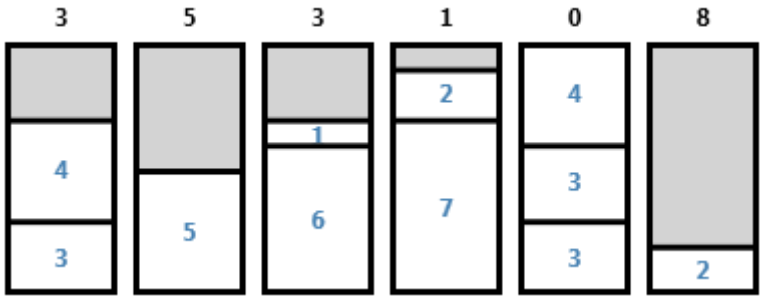
3. jeżeli $s + w_i > c$ (element nie mieści się w aktualnym pudełku) to zwiększ wartość k o 1 i przypisz $s = 0$

4. umieść i-ty element w k-tym pudełku, tzn. $x_{ik} = 1$, $y_k = 1$

5. $s = s + w_i$

Główne zalety tego algorytmu to prostota i szybkość działania – jego złożoność obliczeniowa to zaledwie $O(n)$. Nie wymaga on też znajomości wszystkich elementów *a priori* – elementy są pobierane w kolejności, w której znajdują się na wejściu, można więc dokładać nowe elementy w trakcie działania algorytmu (na zasadzie kolejki). Główną wadą metody jest jednak nie wykorzystywanie wolnego miejsca w pudełkach innych niż aktualne. Często prowadzi to do dokładania nowych pudełek dla kolejnych elementów, podczas gdy w istniejących pudełkach jest jeszcze dla nich miejsce. Przekłada się to oczywiście na niską jakość uzyskiwanych rozwiązań. Wartość r^{∞} dla tego algorytmu wynosi 2.

Dla przykładowej instancji zdefiniowanej na rys. 3.1. *Next-Fit* uzyskuje rozwiązanie z 6 pudełkami. Przedstawia to poniższy rysunek (liczby nad pudełkami reprezentują pozostałe wolne miejsce):



Rys.3.5. Wynik działania algorytmu Next-Fit

3.4.2. Algorytm pierwszego dopasowania (ang. *First-Fit*)

Kolejny algorytm to algorytm pierwszego dopasowania (ang. *First-Fit*). Tak jak *Next-Fit*, pobiera on elementy w kolejności ich występowania w instancji wejściowej. Wybrany element jest umieszczany w pierwszym pudełku, w którym znajduje się

wystarczająca ilość wolnego miejsca (pudełka są numerowane w kolejności ich dodania do rozwiązania). Nowe pudełko jest dodawane tylko wtedy, gdy w żadnym istniejącym pudełku nie ma miejsca dla aktualnie rozważanego elementu.

Algorytm *First-Fit*:

s_j - suma elementów w j-tym pudełku, czyli $s_j = \sum_{h \in N, h:x_{hj}=1} w_h$

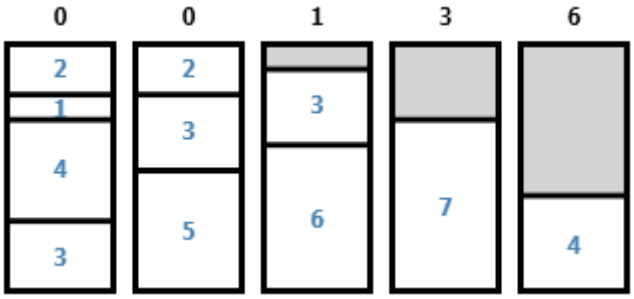
1. dla wartości i od 1 do n wykonaj kroki 2 – 3

2. dla wartości j od 1 do m wykonaj krok 3

3. oblicz wartość s_j. Jeżeli $s_j + w_j \leq c$, to umieść i-ty element w j-tym pudełku, tzn. $x_{ij} = 1,\ y_j = 1$ i przejdź do następnej iteracji pętli w kroku 1. W przeciwnym wypadku przejdź do następnej iteracji pętli w kroku 2.

Algorytm eliminuje główną wadę swojego poprzednika – wykorzystuje wolną przestrzeń w już otwartych pudełkach. Dzięki temu osiągane wyniki są znacznie lepsze: $r^{\infty} = \frac{17}{10}$. Konieczność poszukiwania wolnego miejsca wśród istniejących pudełek powoduje oczywiście zwiększenie złożoności obliczeniowej algorytmu. Przy zastosowaniu struktur tzw. 2-3 drzew (ang. *2-3 tree*) możliwe jest uzyskanie złożoności rzędu $O(n \log n)$ [MART 1990]. W omawianym systemie zdecydowano się jednak na prostszą („klasyczną") implementację algorytmu o złożoności $O(n^2)$, przeszukującą dodane pudełka wg kolejności ich dodania (w czasie liniowym).

Wynik osiągany przez *First-Fit* dla instancji testowej z rys. 3.1. to 5 pudełek. Przedstawiono to na poniższym rysunku:



Rys. 3.6. Wynik działania algorytmu First-Fit

3.4.3. Algorytm najlepszego dopasowania (ang. *Best-Fit*)

Algorytm najlepszego dopasowania (ang. *Best-Fit*) działa na zasadzie podobnej do *First-Fit*. Różnica polega na tym, że zamiast umieszczać element w pierwszym pudełku, w którym element się mieści, umieszcza się go w pudełku, w którym pozostanie po jego dodaniu najmniej wolnego miejsca. Rozwiązanie to bazuje na założeniu, że należy minimalizować pozostałą w poszczególnych pudełkach przestrzeń (wykorzystując je w maksymalnym stopniu) – w ten sposób minimalizuje się całkowitą wolną przestrzeń, a w konsekwencji zmniejsza się liczbę pudełek potrzebnych do umieszczenia wszystkich elementów. Oczywiście istnieją przypadki, gdy takie podejście nie jest efektywne.

Algorytm *Best-Fit*:

s_j - suma elementów w j-tym pudełku, czyli $s_j = \sum_{h \in N, h: x_{hj}=1} w_h$

1. dla wartości i od 1 do n wykonaj kroki 2 – 3

2. dla wartości j od 1 do m wykonaj krok 3

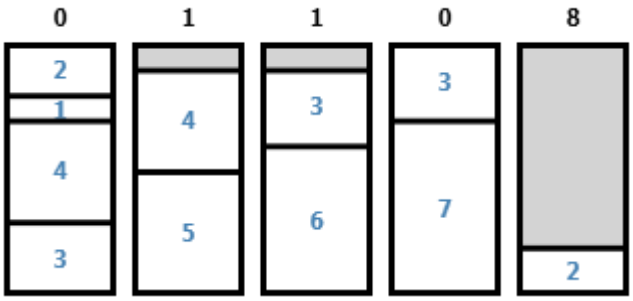
3. oblicz wartość s_j

4. oblicz wartość k, gdzie: $k = min\left\{j: s_j + w_i \leq c, s_j = max\{s_h: s_h + w_i \leq c\}\right\}$, $j = 1, \dots, m$ oraz $h = 1, \dots, m$

5. umieść i-ty element w k-tym pudełku, tzn. $x_{ik} = 1$, $y_k = 1$ i przejdź do następnej iteracji pętli w kroku 1

Wartość r^{∞} dla algorytmu *Best-Fit* jest taka sama jak dla *First-Fit* i wynosi $\frac{17}{10}$. Złożoność czasowa również jest identyczna – $O(n \log n)$ przy zastosowaniu 2-3 drzew ([MART 1990]) i $O(n^2)$ bez wykorzystania tych struktur, którą to wersję algorytmu zamieszczono w bibliotece *Bin Packing*.

Rozwiązanie dla instancji testowej (rys. 3.1.) przedstawiono poniżej. Uzyskany wynik składa się z 5 pudełek.



Rys. 3.7. Wynik działania algorytmu Best-Fit

3.4.4. Algorytm pierwszego dopasowania z sortowaniem (ang. *First-Fit Decreasing*)

Wszystkie opisane do tej pory algorytmy posiadają wspólną cechę – elementy są umieszczane w pudełkach wg kolejności ich występowania w instancji wejściowej. Zaletą takiego podejścia jest brak konieczności znajomości wszystkich elementów *a priori*. Wadą natomiast duża zależność jakości uzyskiwanych rozwiązań od kolejności elementów. Próbę rozwiązania tego problemu stanowią 2 kolejne algorytmy.

Pierwszym z nich jest algorytm *First-Fit Decreasing*. Łatwo zauważyć, że poprzednie algorytmy lepiej radzą sobie w przypadkach, gdy elementy są umieszczane kolejno wg malejących rozmiarów (wag). Obserwację tę wykorzystuje opisywany algorytm. Pierwszy krok algorytmu to posortowanie elementów wg malejących rozmiarów. Następnie tak uzyskaną instancję problemu rozwiązuje się za pomocą algorytmu *First-Fit*. Oczywiście wymaga to znajomości wszystkich elementów już na początku działania algorytmu.

Algorytm *First-Fit Decreasing*:

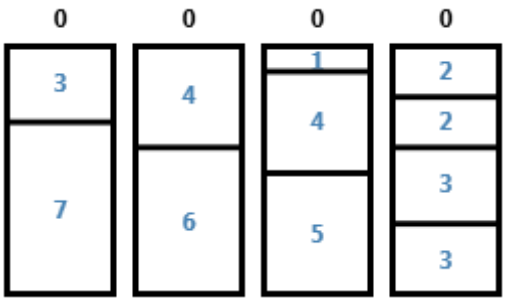
1. posortuj elementy wg malejących rozmiarów

2. rozwiąż uzyskaną instancję problemu algorytmem *First-Fit*

W związku z tym, że algorytmy sortowania posiadają złożoność obliczeniową takiego samego lub niższego rzędu niż złożoność algorytmu *First-Fit*, złożoność czasowa algorytmu *First-Fit Decreasing* jest taka sama co algorytmu *First-Fit* i wynosi $O(n \log n)$ lub $O(n^2)$ w zależności od tego czy wykorzystano 2-3 drzewa czy też nie. Rzeczywisty czas działania jest oczywiście wydłużony o czas sortowania elementów.

W ogólności wyniki osiągane przez algorytm są o wiele lepsze od tych uzyskiwanych przez poprzednie algorytmy. Oszacowanie asymptotyczne r^{∞} wynosi $\frac{11}{9} = 1{,}222\ldots$

Dla instancji problemu pakowania z rys. 3.1. *FFD* daje rozwiązanie optymalne, odpowiadające 4 pudełkom:



Rys. 3.8. Wynik działania algorytmu First-Fit Decreasing

3.4.5. Algorytm najlepszego dopasowania z sortowaniem (ang. *Best-Fit Decreasing*)

Idea algorytmu *Best-Fit Decreasing* jest taka sama jak *FFD* – w pierwszym kroku sortuje on elementy wg malejących rozmiarów. Różnica polega na algorytmie wykorzystywanym w drugim kroku – w tym przypadku jest to *Best-Fit*.

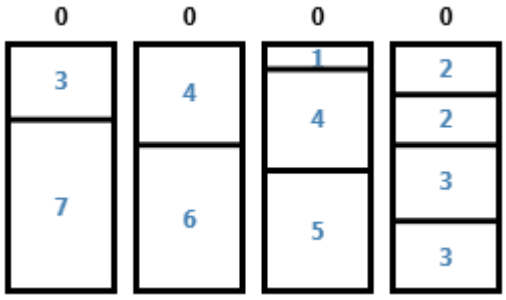
Algorytm *Best-Fit Decreasing*:

1. posortuj elementy wg malejących rozmiarów

2. rozwiąż uzyskaną instancję problemu algorytmem *Best-Fit*

Złożoność czasowa algorytmu *Best-Fit Decreasing* jest identyczna jak w przypadku algorytmu *BF* ($O(n \log n)$ przy zastosowaniu 2-3 drzew i $O(n^2)$ w przeciwnym wypadku). Wartość r^{∞} wynosi $\frac{11}{9}$.

Również w tym przypadku dla instancji z rys. 3.1. otrzymamy rozwiązanie optymalne (4):



Rys. 3.9. Wynik działania algorytmu Best-Fit Decreasing

3.4.6. Algorytm losowego dopasowania (ang. *Random-Fit*)

W przeciwieństwie do poprzednich algorytmów, które pobierały elementy z instancji w ściśle określonej kolejności, algorytm losowego dopasowania pobiera je w kolejności losowej. Kolejnym krokiem jest ponowne losowanie – tym razem losowane jest jedno z pudełek, w których element się zmieści (lista takich pudełek jest uprzednio tworzona). Element zostaje umieszczony w wylosowanym pudełku (lub w nowym – jeżeli w żadnej się nie mieści). Dzięki tworzeniu listy pudełek, algorytm zawsze wykorzystuje wolne miejsce, gdy jest ono dostępne.

Algorytm *Random-Fit*:

1. dopóki istnieje i takie, że $i \in N, \bigwedge_{j=1,\ldots,m} x_{ij} = 0$ powtarzaj kroki 2 – 7

2. wylosuj i takie, że $i \in N, \bigwedge_{j=1,\dots,m} x_{ij} = 0$

3. $C = \{j: s_j + w_i \leq c, y_j = 1\}, j = 1, \dots, m$

4. jeżeli $C \neq \emptyset$ to przejdź do korku 5, w przeciwnym razie do kroku 6.

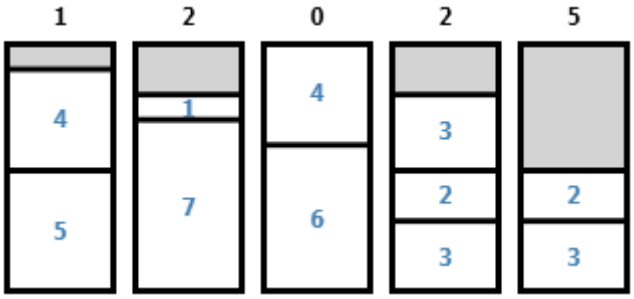
5. wylosuj $k, k \in C$ i przejdź do kroku 7.

6. wybierz $k = min\{j: y_j = 0\}, j = 1, \dots, m$ i przejdź do następnego kroku

7. umieść i-ty element w k-tym pudełku, tzn. $x_{ik} = 1, y_k = 1$ i przejdź do kroku 1.

Złożoność czasowa algorytmu jest taka sama jak w przypadku algorytmu *Best-Fit*, ponieważ algorytm również przegląda wszystkie już dodane pudełka.

Ze względu na element losowości metoda może przy wielokrotnym uruchomieniu zwracać różne wyniki. Jedno z rozwiązań (składające się z 5 pudełek) uzyskanych dla instancji testowej (rys. 3.1.) przedstawiono na poniższym rysunku:



Rys. 3.10. Jeden z wyników działania algorytmu Random-Fit

3.5. Algorytm redukcji

Kolejnym z wykorzystanych algorytmów jest algorytm redukcji. Zastosowane w nim podejście znacznie różni się od sposobu, w jaki działa większość przedstawionych metod. Opisywane do tej pory algorytmy pobierały z instancji elementy pojedynczo, koncentrując się na ich optymalnym umieszczeniu w pudełku. Jest to działanie zorientowane na element (ang. *item-oriented*).

Działanie algorytmu redukcji jest zorientowane na pudełko (ang. *bin-oriented*). Opiera się ono na procedurze redukcji, zaproponowanej przez Martello i Totha [MART 1990]. Procedura ta poszukuje dopuszczalnego wypełnienia pudełka, złożonego z co najwyżej 3 elementów, takiego, że dominuje ono wszystkie pozostałe (co najwyżej 3-elementowe) wypełnienia. Relacja dominacji pozwala porównywać ze sobą 2 dane wypełnienia, tj. określić, które z nich jest wyższej jakości. Relację tę można zdefiniować następująco [FUKA 2007]:

Relacja dominacji: *Dla dwóch danych dopuszczalnych (poprawnych) sposobów wypełnienia pudełek F_1 oraz F_2 mówimy, że F_1* ***dominuje*** *F_2 jeżeli wartość optymalnego rozwiązania które można uzyskać wykorzystując wypełnienie pudełka w sposób F_1 jest* ***nie gorsza*** *od wartości optymalnego rozwiązania, które można uzyskać stosując wypełnienie F_2 dla tego samego pudełka.*

W swojej pracy Fukunaga i Korf opisują kilka kryteriów dominacji dla problemu pakowania, m.in. zaproponowane przez Martello i Totha [MART 1990] (i stosowane w procedurze redukcji):

Kryterium dominacji: *Niech A i B będą poprawnymi wypełnieniami pudełek. A dominuje B jeżeli B można podzielić na i podzbiorów $B_1, \dots, B_i$ takich, że każdemu podzbiorowi B_k przyporządkowany jest element $a_k \in A$ taki, że suma wag (rozmiarów) elementów zbioru B_k jest mniejsza lub równa niż waga (rozmiar) elementu a_k.*

Innymi słowy, jeżeli wszystkie elementy ze zbioru B można zapakować do pudełek, których pojemność stanowią elementy zbioru A, to zbiór A dominuje B.

Przykład:

$A = \{10,20,35\}, B = \{5,5,5,10,15,15\}$

A dominuje B, ponieważ B można podzielić na podzbiory $\{5,5\}$, $\{5,10\}$ oraz $\{15,15\}$, z których każdy można by zmieścić w pudełku odpowiadającemu wielkości pojedynczego elementu A. W tym wypadku wymienione zbiory można przyporządkować kolejno do elementów 10, 20 i 35. Oczywiście istnieją również inne przyporządkowania.

Wynikiem działania procedury redukcji jest wektor $\bar{b}$, złożony z n elementów. Dla każdego $i = 1, \dots, n$ element b_i zawiera numer pudełka, w którym należy umieścić i-ty element.

Algorytm redukcji znajduje dla wszystkich pozostałych elementów wypełnienia dominujące wszystkie pozostałe, a następnie dodaje je do aktualnego rozwiązania. Kolejny krok to usunięcie wykorzystanych w ten sposób elementów z listy tych, które pozostały. W celu ponownego wykorzystania procedury redukcji dodatkowo usuwany jest także najmniejszy element (jest on zapamiętywany na osobnej liście) – pozwala to na kontynuowanie działania w przypadku, gdy procedura redukcji nie znajdzie wypełnień, dominujących pozostałe. Cała procedura jest powtarzana aż do zużycia wszystkich elementów. Ostatnim krokiem jest wypełnienie, w miarę możliwości, wolnych miejsc w uzyskanym rozwiązaniu elementami, które były odrzucane jako najmniejsze w poszczególnych krokach algorytmu. Pozostałe elementy (jeżeli nie wszystkie udało się umieścić w wolnych miejscach) są pakowane algorytmem *Next-Fit* – jest on szybki i w przypadku małych elementów radzi sobie całkiem dobrze.

Algorytm redukcji:

P – lista elementów, jeszcze nie wykorzystanych przez algorytm

L – lista elementów usuwanych jako najmniejsze po każdym wywołaniu procedury redukcji

1. $P = N, L = \emptyset$

2. dopóki $P \neq \emptyset$ powtarzaj kroki 3 – 5

3. stosując procedurę redukcji znajdź wypełnienia pudełek, złożone z elementów listy P, dominujące inne wypełnienia (w postaci wektora $\bar{b}$)

4. dla każdego b_i wykonaj $x_{ib_i} = 1, y_{b_i} = 1, P = P \backslash \{w_i\}, i = 1, \dots, n$

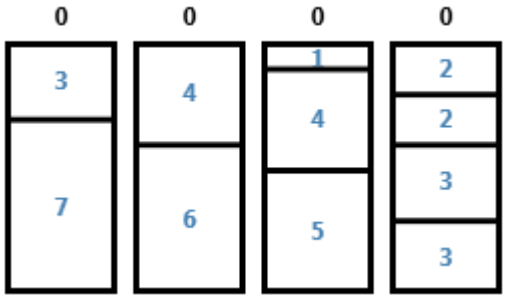
5. jeżeli $P \neq \emptyset$, to wybierz element $k = min\{h: h \in P\}$, a następnie dodaj go do listy L i usuń z listy P: $L = L \cup k$, $P = P \backslash k$.

6. dopóki to możliwe umieszczaj kolejne elementy listy L w wolnych miejscach w aktualnym rozwiązaniu, usuwając je z listy L

7. pozostałe elementy listy L zapakuj do nowych pudełek, stosując algorytm *Next-Fit*

Złożoność czasowa przedstawionego algorytmu to $O(n^2)$ [MART 1990]. Zaimplementowana wersja ma złożoność $O(n^3)$ - ze względu na zastosowanie prostszej implementacji jednej z wewnętrznych fragmentów algorytmu.

Dla przykładowej instancji problemu z rys. 3.1. algorytm znajduje rozwiązanie optymalne, składające się z 4 pudełek.



Rys. 3.11. Wynik działania algorytmu redukcji

3.6. Asymptotyczny schemat aproksymacyjny

Kolejnym z algorytmów zaimplementowanych w bibliotece *Bin Packing* jest asymptotyczny schemat aproksymacyjny (ang. *asymptotic approximation scheme*, *AAS*), zaproponowany przez Fernandez de la Vegę i Luekera [FERN 1981] oraz opisany także przez Korte'a i Vygena [KORT 2000]. Pozwala on na uzyskanie rozwiązania o zadanej dokładności. Oczywiście zwiększanie dokładności powoduje wzrost czasu obliczeń.

Idea metody polega na podzieleniu elementów na 3 główne grupy. Pierwsza z nich zawiera największe elementy, które są pakowane po jednym do osobnych pudełek. Trzecia składa się z elementów najmniejszych. Druga, złożona z elementów

o średniej wielkości jest dzielona na równoliczne zbiory elementów. Rozmiary elementów należących do poszczególnych zbiorów zostają zaokrąglone do rozmiaru największego elementu zbioru. Dzięki temu liczba różnych rozmiarów elementów analizowanych w trakcie konstrukcji rozwiązania zostaje znacznie zmniejszona. Elementy o pośrednim rozmiarze pakowane są do pudełek w oparciu o rozwiązanie zadania programowania liniowego [PAPA 1982], które jest następnie transformowane do rozwiązania dopuszczalnego (całkowitoliczbowego). Ostatnim krokiem jest wypełnienie w miarę możliwości pozostałych miejsc najmniejszymi elementami (z trzeciej grupy). Te elementy, których nie uda się pomieścić są pakowane algorytmem *Next-Fit*.

Asymptotyczny schemat aproksymacyjny:

1. dla podanej dokładności ε oblicz wartości parametrów pomocniczych: $\gamma = \frac{\varepsilon}{\varepsilon+1}$, $h = \lceil \varepsilon SUM(I) \rceil$, gdzie $SUM(I)$ oznacza sumę rozmiarów wszystkich elementów wejściowej instancji problemu pakowania I.

2. stwórz listę L, zawierającą elementy spełniające warunek: $w_i < \gamma, i = 1, \dots, n$

3. oblicz pomocniczy parametr $m = \left\lfloor \frac{|I|-|L|}{h} \right\rfloor$, gdzie $|I|$ oznacza liczbę elementów instancji wejściowej I (i wynosi n). m określa liczbę zbiorów (list), na które zostanie podzielona lista elementów o średnich rozmiarach.

4. Stwórz listy M i R:

$$M = K_0, y_1, K_1, y_2, \dots, K_{m-1}, y_m$$
$$K_0 = \{w_i : \gamma \leq w_i \leq y_1\}, i = 1, \dots, n$$
$$K_i = \{w_i : y_j \leq w_i \leq y_{j+1}\}, i = 1, \dots, n;\ j = 1, \dots, m-1$$
$$R = \{w_i : y_m \leq w_i\}$$
$$|K_1| = \cdots = |K_{m-1}| = |R| = h - 1$$
$$|K_0| \leq h - 1$$

gdzie:

$K_0, \dots, K_{m-1}$ – listy

y_j – elementy których rozmiary będą wykorzystywane w zadaniu programowania liniowego. Elementy y_j należy wyznaczyć jako $(|I| + 1 - (m - j + 1)h)$-ty najmniejszy element w instancji I, $j = 1, \dots, m$.

5. zapakuj elementy listy R, umieszczając każdy jej element w osobnym pudełku

6. na podstawie listy M stwórz nową listę elementów Q. Lista Q składa się z elementów o rozmiarach $y_1, \dots, y_m$, z których każdy występuje h razy.

7. wygeneruj wszystkie możliwe sposoby zapakowania pojedynczego pudełka za pomocą elementów nowej listy. Oznaczmy liczbę uzyskanych sposobów zapakowania przez P. Otrzymane sposoby pakowania można zapisać jako: $T_p = (t_{p1}, \dots, t_{pm})$, $p = 1, \dots, P$. Pozycja t_{pj} oznacza liczbę wystąpień elementu o rozmiarze y_m w pakowaniu T_p, $p = 1, \dots, P$ a $j = 1, \dots, m$.

8. rozwiąż zadanie programowania liniowego postaci:

$$\begin{aligned} & min && \sum_{p=1}^{P} x_p \\ & przy\ ogr. && \sum_{p=1}^{P} t_{pj} x_p \geq h_j \quad j = 1, \dots, m \\ & && x_p \geq 0 \qquad p = 1, \dots, P \end{aligned}$$

gdzie:

x_p – liczba pudełek zapakowanych w sposób T_p

h_j – liczba elementów o rozmiarze y_j. Ze względu na to, że liczba elementów w liście K_0 może być mniejsza niż w listach $K_1, \dots, K_{m-1}$, nie można podać jako ograniczenia liczby h.

9. przekształć otrzymane rozwiązanie w rozwiązanie całkowitoliczbowe, zaokrąglając każdą otrzymaną wartość x_p w dół, tzn. zastępując ją wartością $\lfloor x_p \rfloor$, $p = 1, \dots, P$.

10. przekształć uzyskane w ten sposób rozwiązanie (zapakowanie listy Q) w zapakowanie listy M poprzez zastąpienie wartości poszczególnych elementów o rozmiarach $y_1, \dots, y_m$ oryginalnymi rozmiarami elementów z listy M.

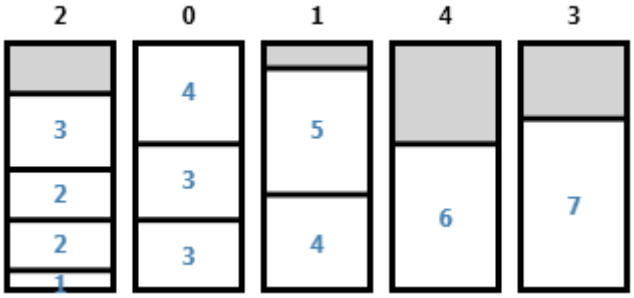
11. zapakuj elementy listy M do nowych pudełek, stosując otrzymane rozwiązanie.

12. jeżeli $M \neq \emptyset$, zapakuj elementy listy M za pomocą algorytmu *Next-Fit*

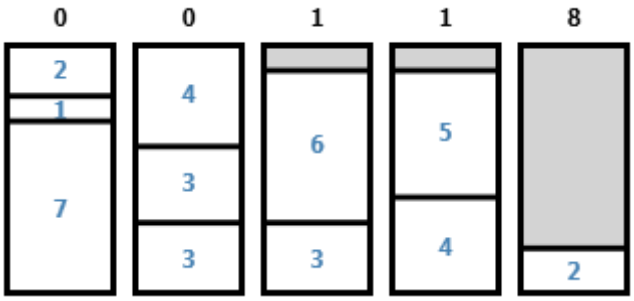
13. dopóki to możliwe, umieszczaj kolejne elementy listy L w wolnych miejscach w aktualnym rozwiązaniu

14. jeżeli $L \neq \emptyset$, zapakuj pozostałe elementy listy L do nowych pudełek, stosując algorytm *Next-Fit*

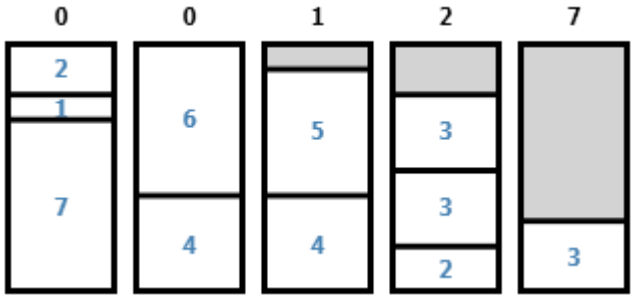
Dla dowolnie określonej dokładności $\varepsilon > 0$ złożoność czasowa aproksymacyjnego schematu obliczeń wynosi $O\left(n\frac{1}{\varepsilon^2}\right)$. Metoda gwarantuje uzyskanie rozwiązania o liczbie pudełek $(1+\varepsilon)OPT(I) + \frac{1}{\varepsilon^2}$, gdzie $OPT(I)$ oznacza liczbę pudełek w rozwiązaniu optymalnym. Poniżej przedstawiono rezultaty uzyskane dla przykładowej instancji testowej z rys. 3.1. dla różnych wartości parametru ε.



Rys. 3.12. Wynik działania asymptotycznego schematu aproksymacji dla $\varepsilon = 0{,}1$



Rys. 3.13. Wynik działania asymptotycznego schematu aproksymacji dla $\varepsilon = 0{,}3333$



Rys. 3.14. Wynik działania asymptotycznego schematu aproksymacji dla $\varepsilon = 0{,}5$

3.7. Algorytm następnego dopasowania z dodatkową optymalizacją

Poza implementacją znanych z literatury algorytmów rozwiązujących problem pakowania, zdecydowano się również na stworzenie własnej metody heurystycznej. Opracowany algorytm jest połączeniem dwóch wcześniej opisywanych rozwiązań – algorytmu *Next-Fit* oraz algorytmu redukcji. Nazwano go *PBI* (ang. *Probably Best Improvement*) – przypuszczalnie najlepsza poprawa. W pierwszym etapie metody elementy są pakowane algorytmem następnego dopasowania. Następnie podejmowane są próby poprawienia uzyskanego rozwiązania. Procedura poprawy jest wywoływana maksymalnie $\frac{m}{2}$ razy, gdzie m oznacza liczbę pudełek rozwiązania uzyskanego w pierwszym etapie. Wynika to z tego, że algorytm *Next-Fit* w najgorszym wypadku zwróci rozwiązanie 2 razy gorsze od optymalnego – nie można więc zmniejszyć liczby pudełek więcej niż dwukrotnie. Poza tym jeżeli w 3 kolejnych próbach nie uzyskano poprawy, algorytm jest przerywany.

W poszczególnych próbach poprawy wybierane są pewne pudełka (co najwyżej 9), których upakowanie próbuje się poprawić. W pierwszej kolejności wybierane są 3 pudełka z największą ilością wolnego miejsca – teoretycznie ich zawartość najłatwiej będzie umieścić w innych pudełkach. Następnie wybiera się 3 pudełka z największą liczbą elementów – największa liczba elementów oznacza, w przeciętnym przypadku, najmniejsze elementy – dają one najwięcej możliwości przenoszenia elementów poprzez nieznaczne zwiększenie wolnej przestrzeni. Na końcu wybiera się 3 losowe pudełka. W przypadku, gdy liczba pudełek jest mniejsza niż 9, wybierane są wszystkie pudełka. Elementy z tak wybranych (co najwyżej 9) pudełek są pakowane redukcji.

W najgorszym przypadku rozwiązanie uzyskane przez *Next-Fit* będzie odpowiadało liczbie elementów, czyli n. W takim wypadku znalezienie pudełek do poprawy będzie wymagało sprawdzenia n pudełek. Liczba prób poprawienia wyniku może wynieść maksymalnie $\frac{n}{2}$, co daje złożoność $O\left(\frac{n^2}{2}\right)$. Każda próba poprawy wymaga jednak uruchomienia procedury redukcji, której czas obliczeń zależy od liczby elementów w pudełkach, których upakowanie próbuje się poprawić. W najgorszym przypadku będzie ich n (wszystkie elementy), więc złożoność obliczeniowa całego algorytmu wyniesie $O(n^5)$. Warto jednak zauważyć, że przypadek ten zachodzi tylko dla bardzo małych instancji.

Ze względu na sposób działania dla instancji z rysunku 3.1. po pierwszym kroku do poprawy wybrane zostaną wszystkie pudełka. Działanie algorytmu będzie więc równoważne z uruchomieniem algorytmu redukcji dla tej instancji. Otrzymane rozwiązanie jest więc optymalne i składa się z 4 pudełek.

4. IMPLEMENTACJA

4.1. Opis systemu

Aplikację *Bin Packing* zdecydowano się napisać jako standardową aplikację desktopową. Została ona napisana w całości w języku programowania C#, z wykorzystaniem platformy .NET w wersji 3.5. Interfejs oparto na technologii WPF (ang. *Windows Presentation Foundation*). Składa się on z głównego okna, udostępniającego większość funkcji oraz dodatkowych okienek pomocniczych, pozwalających na wprowadzenie dodatkowych informacji przez użytkownika (np. podczas otwierania pliku) i wyświetlających postęp oraz wynik obliczeń (prezentowanie sposobu działania algorytmów, wyświetlanie wyników eksperymentu obliczeniowego, itp.).

Wiele elementów systemu zdecydowano się zrealizować za pomocą własnych kontrolek. Dotyczy to m.in. wyświetlania wykresów, przedstawiających wyniki eksperymentu obliczeniowego, wyświetlania podglądu elementów i pudełek oraz kontrolek pozwalających na wybór koloru (np. wypełnienia pudełek). Główne przyczyny podjętej decyzji to brak dostępności odpowiednich kontrolek w bibliotece .NET. Kolejny powód, dotyczący w szczególności rysowania wykresów, to niewielka liczba skończonych (lub nadal rozwijanych), darmowych rozwiązań, oferujących potrzebną funkcjonalność oraz konieczność nauki wykorzystania wybranego rozwiązania, przy częstym braku przykładów i ubogiej dokumentacji technicznej.

4.1.1. Architektura

System składa się z dwóch głównych części. Pierwsza z nich to biblioteka DLL (ang. *Dynamic-Link Library*) zawierająca właściwą funkcjonalność związaną z algorytmami rozwiązującymi problem pakowania. Główne elementy biblioteki to:

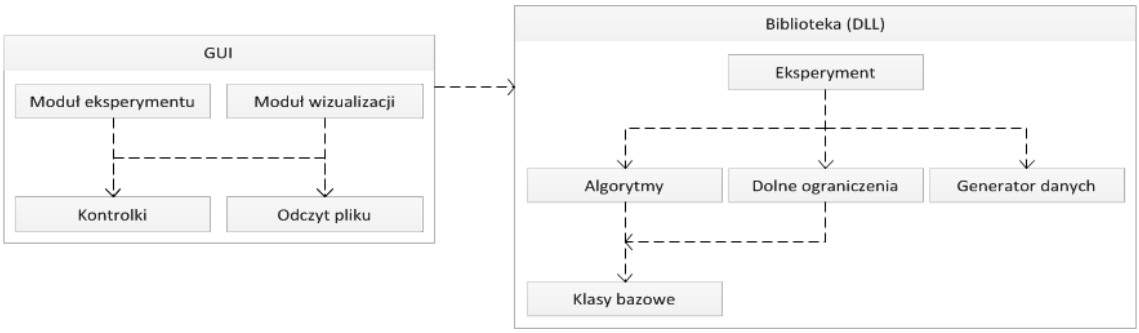
- klasy bazowe (ang. *base*) – podstawowe klasy, reprezentujące pudełko, instancję problemu oraz algorytm (właściwości i metody występujące we wszystkich algorytmach, np. jak nazwa) i algorytm listowy (właściwości i metody występujące we wszystkich algorytmach, których sposób działania można prezentować, tj. w algorytmach listowych)

- dolne ograniczenia – implementacja dolnych ograniczeń dla zadanej instancji
- algorytmy – implementacje poszczególnych algorytmów
- eksperyment – klasy odpowiedzialne za przeprowadzanie eksperymentu obliczeniowego oraz klasy reprezentujące m.in. aktualny stan eksperymentu, jego parametry wejściowe, oraz wyjściowe (próbki danych), itp.
- generator danych – klasy, umożliwiające generowanie losowych elementów, spełniających wymagania co do zakresu rozmiarów elementów i rozkładu ich wartości

Drugą część systemu stanowi graficzny interfejs użytkownika – *GUI* (ang. *Graphical User Interface*). Składają się na niego:

- moduł wizualizacji – widok głównego okna programu oraz okna prezentujące algorytmy
- moduł eksperymentu – widok ustawień eksperymentu obliczeniowego w głównym oknie programu oraz okna wyświetlające informacje o postępie eksperymentu i jego wynik
- kontrolki – własne kontrolki, reprezentujące pojedynczy element, pudełko oraz wykres przedstawiający wyniki eksperymentu; korzystają z nich 2 poprzednie moduły
- okno odczytu pliku – okno umożliwiające wybór typu otwieranego pliku (zawierającego jedną lub wiele instancji problemu pakowania)

Architekturę systemu *Bin Packing* przedstawiono na rysunku 4.1.



Rys. 4.1. Architektura systemu

4.1.2. Wymagania funkcjonalne i pozafunkcjonalne

Przed systemem *Bin Packing* postawiono kilka wymagań dotyczących funkcjonalności. Najważniejsze z nich to implementacja wybranych algorytmów, w celu umożliwienia rozwiązywania instancji problemu pakowania oraz przeprowadzania eksperymentów obliczeniowych. Szczególnie ważna była implementacja heurystyk listowych. Kolejne z postawionych wymagań to implementacja dolnych ograniczeń umożliwiających oszacowanie jakości rezultatów osiąganych przez poszczególne algorytmy. System *Bin Packing* miał też umożliwiać prezentację zasady działania algorytmów listowych oraz wizualizację rozwiązań uzyskiwanych przez poszczególne algorytmy. Należało również umożliwić odczyt oraz zapis instancji problemu pakowania do pliku. Ważne było również udostępnienie możliwości przeprowadzenia eksperymentu obliczeniowego, polegającego na wykonywaniu obliczeń dla dużych zbiorów danych i porównaniu efektywności algorytmów oraz przedstawienie wyników eksperymentu w postaci wykresu – wraz z możliwością wyboru wyświetlanych parametrów i dodatkowych funkcji. Aby umożliwić powtórzenie eksperymentu obliczeniowego dla tych samych parametrów, należało umożliwić zapis oraz odczyt ustawień generatora danych dla eksperymentu z pliku. Dalszą analizę wyników miał umożliwić zapis wyników eksperymentu do pliku w formacie, który można by otworzyć np. w programie *Excel*.

Poza wymaganiami funkcjonalnymi zdefiniowano również wymagania pozafunkcjonalne. Biblioteka algorytmów *Bin Packing* miała być w założeniu prosta i intuicyjna w obsłudze. Wprowadzanie przykładowych instancji problemu pakowania

powinno być proste i szybkie. Po wprowadzeniu danych użytkownik powinien mieć możliwość „natychmiastowego” wyświetlenia wyniku działania wybranego algorytmu. Przeprowadzenie eksperymentu obliczeniowego, w tym wprowadzenie jego parametrów, nie powinno sprawiać trudności również mniej doświadczonym uzytkownikom.

4.1.3. Wymagania sprzętowe i systemowe

W celu zapewnienia poprawności działania aplikacji, komputer użytkownika powinien spełniać następujące wymagania:

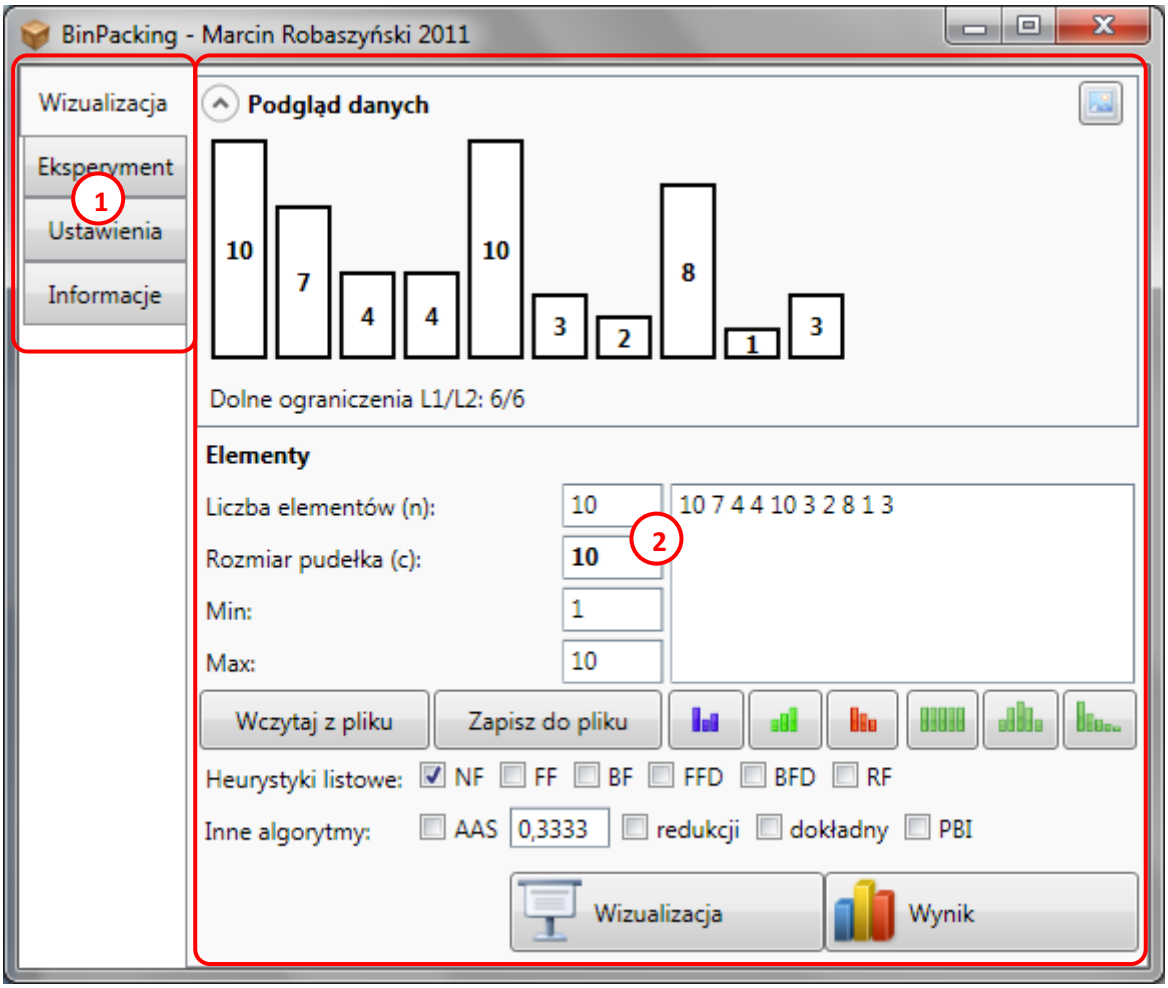
- komputer klasy PC
- system operacyjny Microsoft Windows XP lub nowszy
- przynajmniej 192 MB pamięci operacyjnej RAM
- karta graficzna z obsługą akceleracji sprzętowej.

W systemie musi być również zainstalowany komponent .NET framework w wersji 3.5.

4.2. Dokumentacja użytkownika

4.2.1. Główne okno aplikacji

Po uruchomieniu aplikacji użytkownikowi systemu *Bin Packing* prezentowane jest jej główne okno. Oparto je na jednej ze standardowych architektur interfejsu, często stosowanej w programach antywirusowych. Składa się ono z 2 głównych części. Pierwszą z nich stanowi menu (rys. 4.2. poz. 1.), udostępniające główne opcje systemu. Druga natomiast (rys. 4.2. poz. 2.) służy do wyświetlania szczegółowych opcji/ustawień dla aktualnie wybranej pozycji z menu głównego.

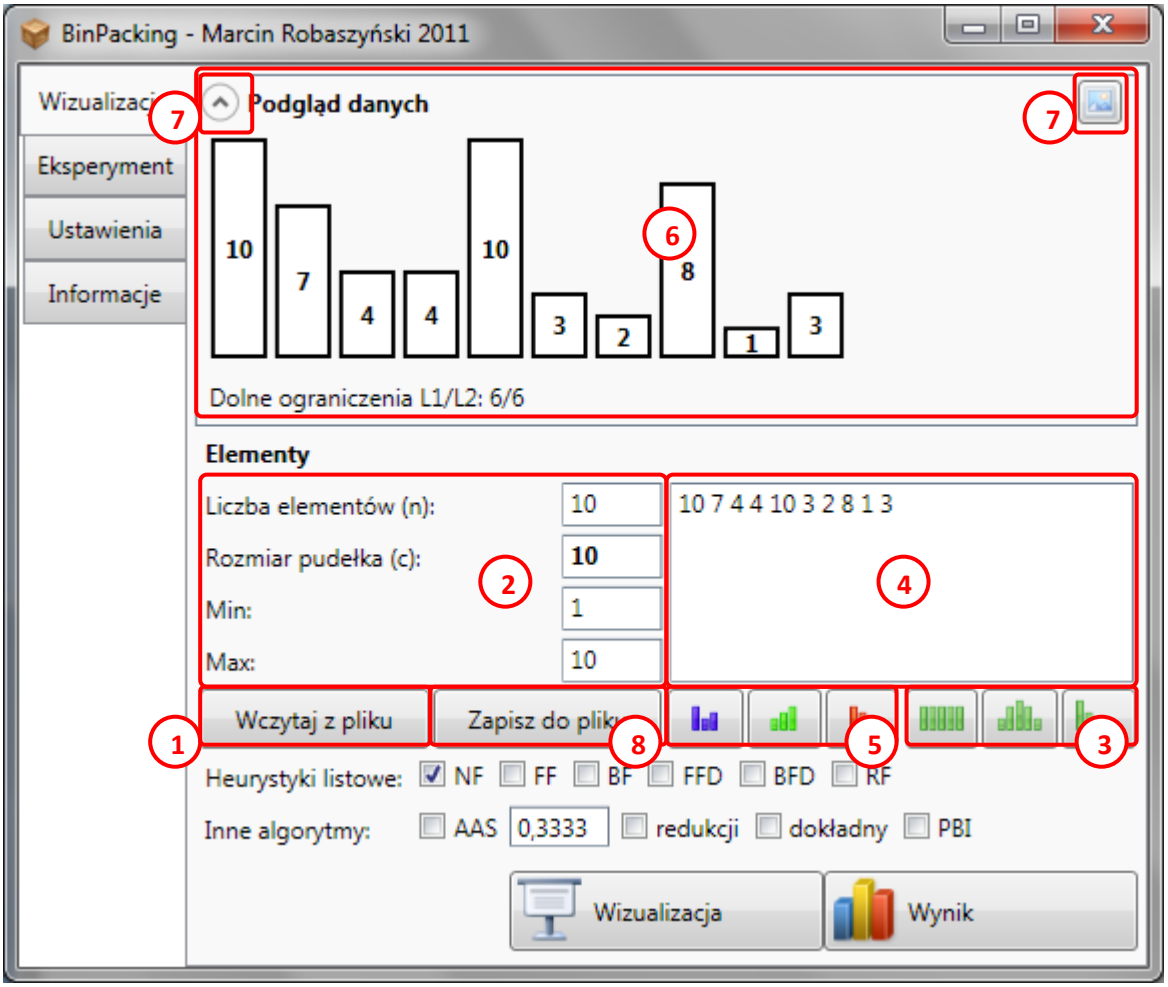


Rys. 4.2. Główne okno programu

Menu główne składa się z 4 pozycji. Pierwsza z nich („Wizualizacja”) udostępnia moduł wizualizacji. Druga („Ekspryment”) powoduje przejście do modułu eksperymentu obliczeniowego. Trzecia pozycja odpowiada za ustawienia parametrów konfiguracyjnych systemu, natomiast ostatnia – wyświetla podstawowe informacje o autorze programu.

4.2.2. Moduł wizualizacji

Moduł wizualizacji umożliwia prezentację działania algorytmów listowych krok po kroku oraz wyświetlenie wyników działania wszystkich algorytmów dostępnych w systemie dla pojedynczych instancji problemu pakowania. Na rysunku 4.3. przedstawiono widok modułu wizualizacji.



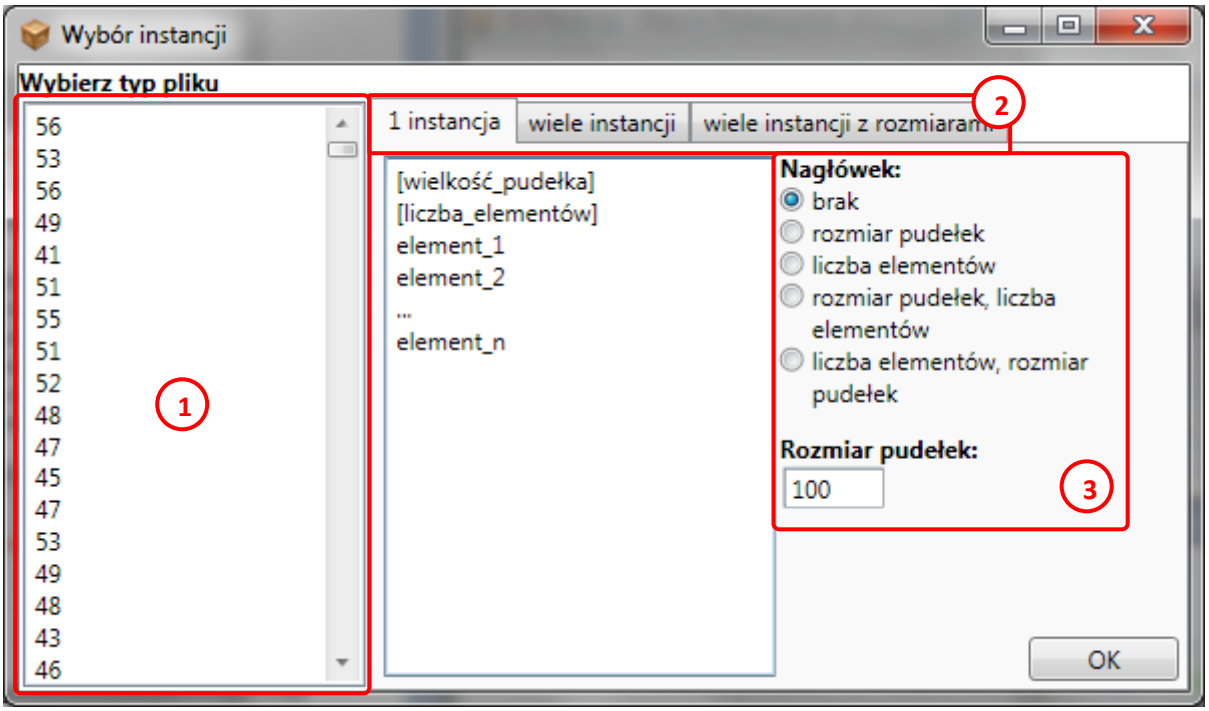
Rys. 4.3. Widok modułu wizualizacji

Wprowadzanie danych

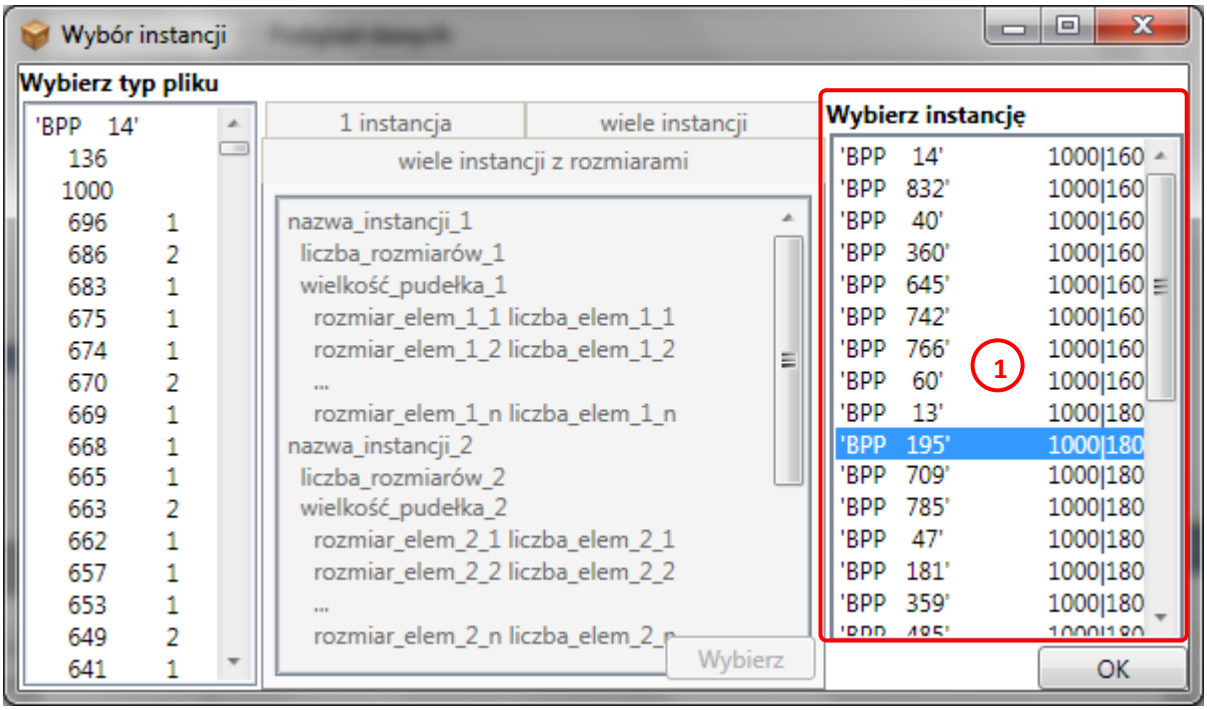
W celu wyświetlenia wyniku działania lub prezentacji sposobu działania algorytmu dla instancji problemu pakowania należy tę instancję wprowadzić do programu. Można to zrobić na 3 sposoby. Pierwszym z nich jest pobranie instancji z pliku, drugim automatyczne wygenerowanie instancji, trzecim natomiast – ręczne wprowadzenie wielkości poszczególnych elementów i pudełka.

Aby pobrać instancję problemu pakowania z pliku należy kliknąć odpowiedni przycisk (rys. 4.3. poz. 1.) i wybrać plik za pomocą standardowego okna otwierania pliku. Następnie wyświetlone zostanie okno wyboru typu pliku (rys. 4.4.). Po jego lewej stronie (poz. 1.) wyświetlana jest zawartość pliku. W prawej części okna należy wybrać typ pliku za pomocą zakładek (poz. 2.) – obsługiwane są pliki zawierające pojedynczą

instancję problemu pakowania, wiele instancji problemu lub też wiele instancji wraz z opisem grup. Poszczególne typy plików zostały opisane bardziej szczegółowo w podrozdziale 4.3.5. W przypadku pliku z pojedynczą instancją problemu pakowania należy podać kilka dodatkowych informacji nt. nagłówka (zawartości pierwszych dwóch linii pliku) – czy zawiera on liczbę elementów i/lub informację o rozmiarze pudełek. W przypadku braku tej drugiej informacji należy podać ją ręcznie w polu znajdującym się w dolnej części poz. 3. na rys. 4.4. Dla plików z wieloma instancjami należy kliknąć przycisk „Wybierz” i wybrać pojedynczą instancję z listy (rys. 4.5. poz. 1). Po kliknięciu przycisku „OK” okno zostanie zamknięte a wybrana instancja problemu załadowana i wyświetlona.



Rys. 4.4. Okno wyboru typu pliku



Rys. 4.5. Wybór pojedynczej instancji z pliku zawierającego wiele instancji problemu

Drugi sposób wprowadzenia danych do systemu *Bin Packing* to ich automatyczne wygenerowanie. Parametry generowanych danych należy wpisać do pól z rys. 4.3. poz. 2. a następnie wygenerować losowe dane (spełniające parametry) za pomocą jednego z 3 przycisków (rys. 4.3. poz. 3.). Każdy z nich generuje dane wg innego rozkładu (od lewej): jednostajnego, normalnego (Gaussa) oraz wykładniczego. Wygenerowane dane zostaną wyświetlone w polu rys. 4.3. poz. 4. Przyciski (rys. 4.3. poz. 3. i 5.) umożliwiają uzyskanie konkretnej kolejności elementów (od lewej): losowej (przetasowanie elementów), rosnącej oraz malejącej.

Ostatni, trzeci sposób utworzenia instancji problemu to ręczne wprowadzenie danych w polu 4.4.. Wprowadzone elementy muszą być liczbami całkowitymi – elementy nie spełniające tych warunków zostaną usunięte. Możliwa jest też edycja elementów uprzednio wygenerowanych bądź odczytanych z pliku. Umożliwia to zmianę rozmiarów pojedynczych elementów, itp.

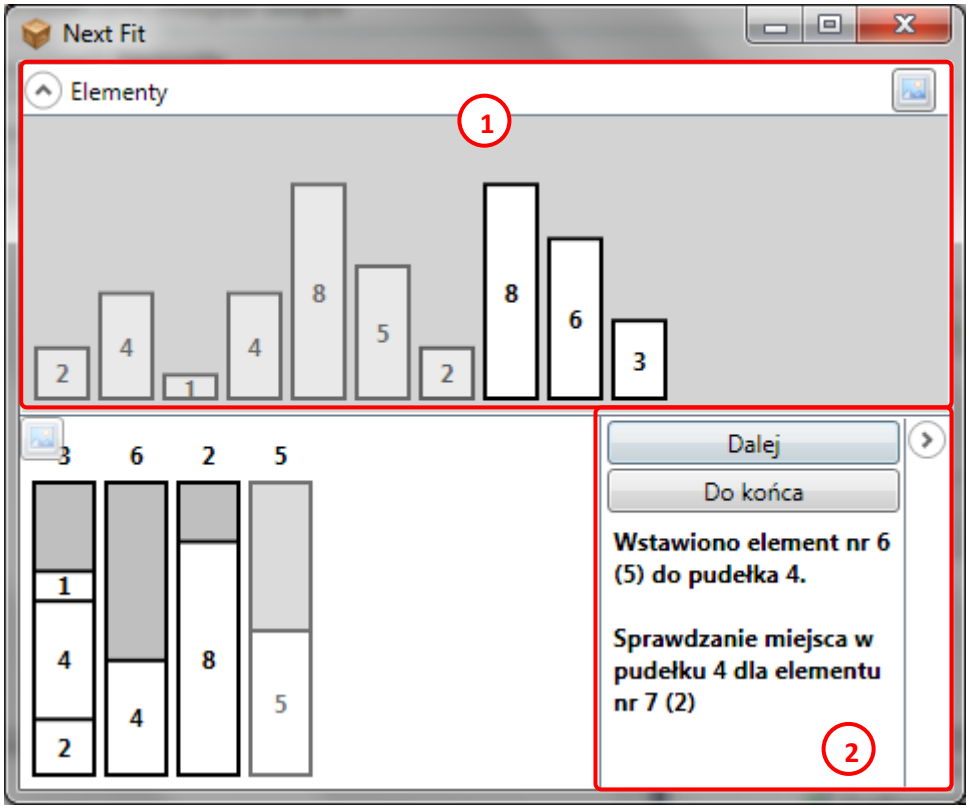
Niezależnie od wybranej metody wprowadzenia danych, każda zmiana rozmiarów elementów spowoduje wyświetlenie podglądu uzyskanej w ten sposób instancji rys. 4.3. poz. 6., wraz z wartościami obliczonych dolnych ograniczeń L_1 i L_2. Za pomocą przycisków rys. 4.3. poz. 7. można zwinąć podgląd lub zapisać instancję do

pliku graficznego. Aby zapisać instancję do pliku, należy kliknąć przycisk z rys. 4.3. poz. 8. Spowoduje to wyświetlenie standardowego okna zapisu pliku, w którym należy wskazać lokalizację oraz nazwę pliku wyjściowego.

Przed rozpoczęciem prezentacji zasady działania algorytmów należy wybrać metody, które zostaną zaprezentowane. Prezentacja działania jest możliwa tylko dla algorytmów listowych (górny rząd w oknie na rys. 4.3.). Możliwy jest wybór kilku algorytmów – dla każdego z nich zostanie utworzone osobne okno. Aby rozpocząć prezentację (lub wyświetlić wynik działania algorytmu) należy kliknąć odpowiednio przycisk „Wizualizacja" lub „Wynik".

Prezentacja sposobu działania algorytmu

W przypadku wybrania prezentacji sposobu działania algorytmu wyświetlone zostanie okno zaprezentowane na poniższym rysunku:



Rys. 4.6. Okno prezentacji działania algorytmu

W górnej części (rys. 4.6. poz. 1.) wyświetlany jest podgląd wszystkich elementów składających się na instancję problemu pakowania wraz z ich rozmiarami. Elementy, które już zostały wykorzystane są wyblakłe, natomiast aktualnie wybrany element pulsuje. Za pomocą przycisków na górnej belce możliwe jest zwinięcie podglądu lub zapisanie go do pliku graficznego.

W środkowej części umieszczono podgląd bieżącego stanu algorytmu. Aktualnie wybrane pudełko pulsuje. Przycisk w lewym górnym rogu umożliwia zapis podglądu do pliku graficznego.

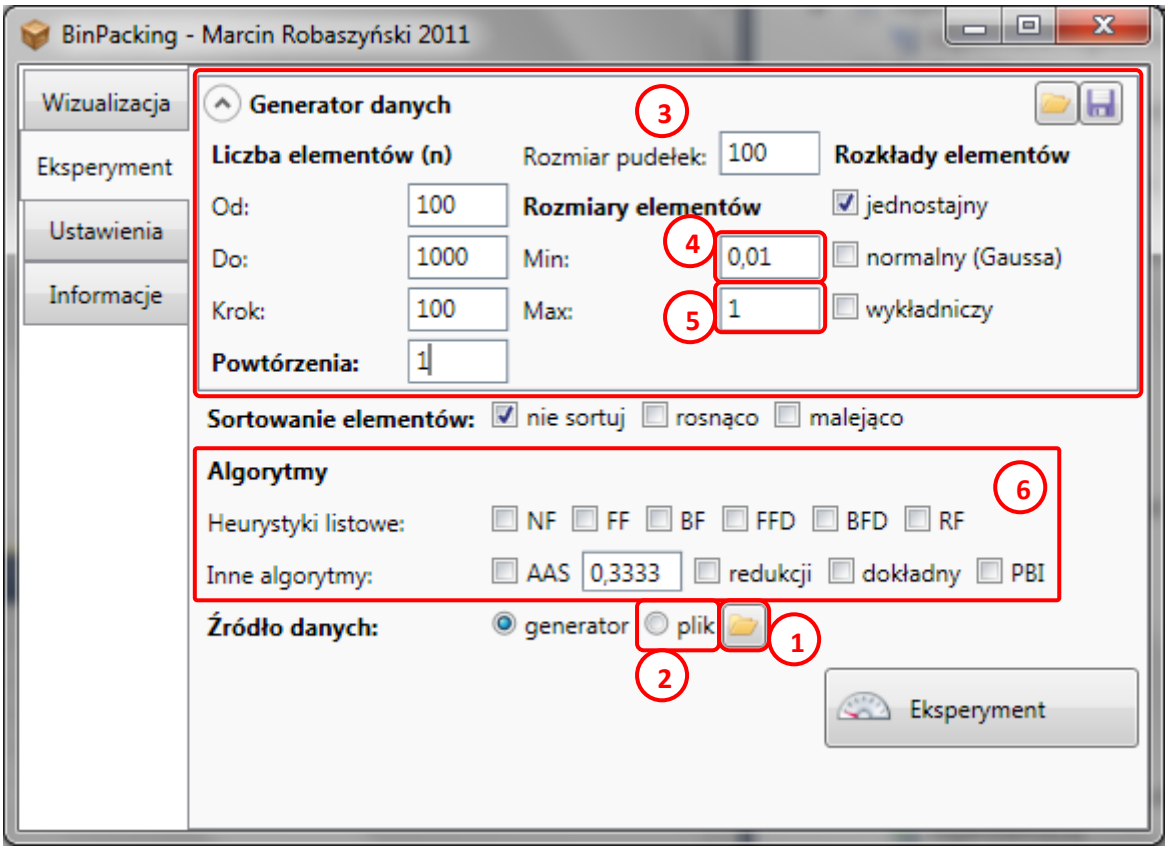
Do sterowania przebiegiem prezentacji służy panel znajdujący się w prawej części okna (rys. 4.6. poz. 2.). W jego górnej części umieszczono przyciski pozwalające wykonać następny krok algorytmu (przycisk „Dalej") lub też przejść do końca algorytmu (przycisk „Do końca"). Poniżej przycisków wyświetlane są informacje dotyczące wykonywanych kroków.

Po zakończeniu algorytmu wyświetlone zostaną podstawowe statystyki dotyczące efektywności jego działania.

W przypadku wybrania wyświetlenia wyniku działania algorytmu (bez prezentacji sposobu działania) zostanie wyświetlone to samo okno (co w przypadku prezentacji) wraz ze statystykami, w tym dodatkową informacją nt. czasu działania algorytmu.

4.2.3. Moduł eksperymentu obliczeniowego

Moduł eksperymentu obliczeniowego służy do porównywania efektywności algorytmów wchodzących w skład biblioteki (lub badania pojedynczego algorytmu) dla zbiorów danych, o różnej charakterystyce. Możliwe jest również sprawdzenie działania algorytmów dla pojedynczej konkretnej instancji załadowanej z pliku. Widok modułu eksperymentu obliczeniowego przedstawiono na poniższym rysunku:



Rys. 4.7. Widok modułu eksperymentu obliczeniowego

Wprowadzanie parametrów eksperymentu

W przypadku testowania pojedynczej instancji załadowanej z pliku należy kliknąć przycisk rys. 4.7. poz. 1. i załadować plik w sposób analogiczny do ładowania instancji podczas prezentacji zasady działania algorytmów (patrz: punkt 4.2.2.1.). Następnie należy zaznaczyć opcję rys. 4.7. poz. 2..

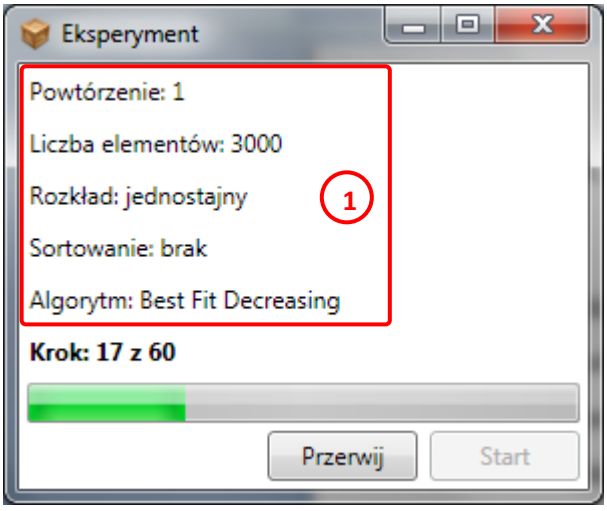
Możliwe jest też generowanie danych testowych „na bieżąco”, w trakcie trwania eksperymentu. Parametry generatora należy podać w górnej części okna (rys. 4.7. poz. 3.). Pierwsza kolumna odpowiada za rozmiary generowanych instancji i liczbę powtórzeń obliczeń. Druga kolumna ustawień generatora odpowiada za rozmiar pudełka i elementów. Będą one miały rozmiary z przedziału *A*-*B*, gdzie *A* odpowiada rozmiarowi pudełka pomnożonemu przez wartość wpisaną w polu 4., a *B* rozmiarowi pudełka pomnożonemu przez wartość wpisaną w polu 5. Najmniejsza wartość, jaką można wprowadzić do pola minimalnej wartości to 0,00001, natomiast największa wartość pola maksymalnej wartości to 1. Ostatnia kolumna odpowiada za sposób

losowania elementów (różne rozkłady prawdopodobieństwa). Dostępne rozkłady prawdopodobieństwa to rozkład jednostajny, normalny oraz wykładniczy. Możliwe jest wybranie kilku rozkładów jednocześnie – spowoduje to powtórzenie obliczeń dla każdego wybranego sposobu losowania rozmiarów elementów.

Niezależnie od wyspecyfikowanego źródła danych, przed rozpoczęciem eksperymentu należy jeszcze dokonać wyboru testowanych algorytmów za pomocą pól wyboru (rys. 4.7. poz. 6.), oraz ustalić czy elementy mają być sortowane wg rozmiarów czy nie. Możliwe jest nie sortowanie elementów (w takim wypadku ich kolejność pozostaje bez zmian) lub też sortowanie ich wg malejących lub rosnących rozmiarów. Tak jak w przypadku różnych rozkładów rozmiarów elementów możliwy jest wybór kilku opcji jednocześnie. Kliknięcie przycisku „Eksperyment" spowoduje wyświetlenie okna postępu przebiegu eksperymentu, które opisano w kolejnym punkcie.

Śledzenie przebiegu eksperymentu

Po uruchomieniu eksperymentu obliczeniowego program wyświetli okno postępu przebiegu eksperymentu (rys. 4.8.). Wyświetla ono informacje o aktualnie uruchomionym algorytmie i danych, na których działa (rys. 4.8. poz. 1.) oraz pasek, przedstawiający całkowity postęp przebiegu eksperymentu.

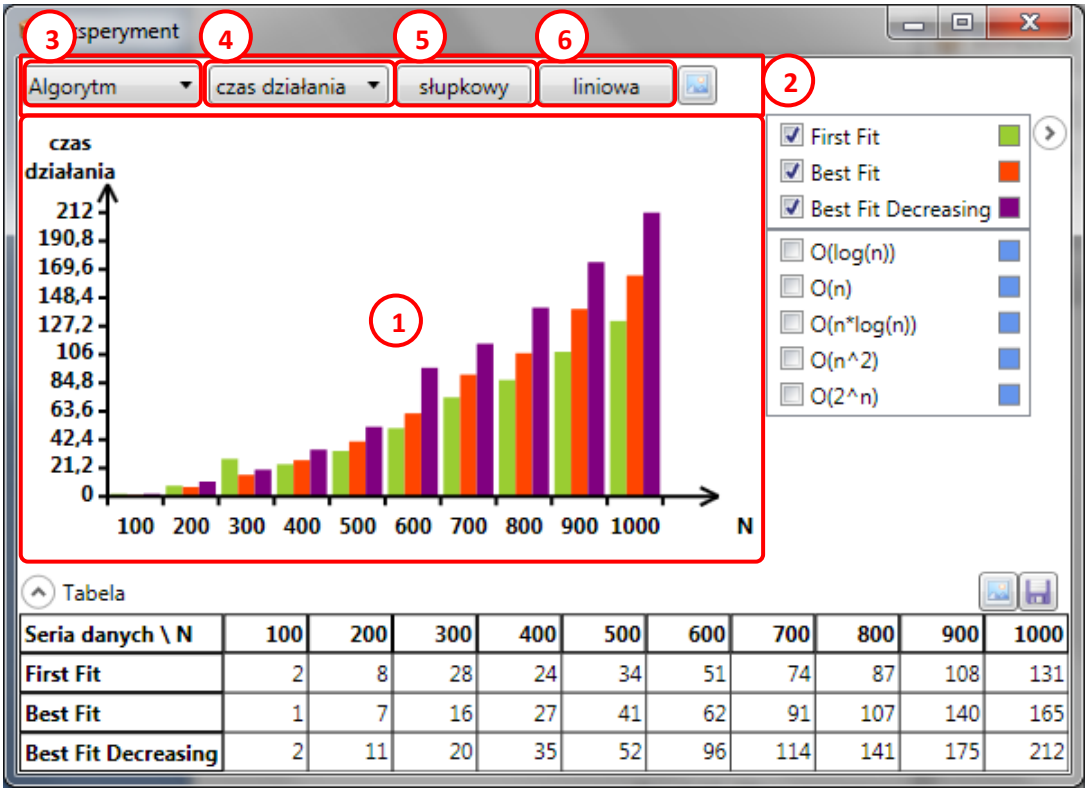


Rys. 4.8. Okno przebiegu eksperymentu

W celu rozpoczęcia obliczeń należy kliknąć przycisk „Start”. Kliknięcie przycisku „Przerwij” w trakcie eksperymentu spowoduje jego przerwanie i powrót do głównego okna programu.

Wyświetlanie wyników działania eksperymentu

Po zakończeniu eksperymentu wyświetlane jest okno prezentujące wyniki:



Seria danych \ N	100	200	300	400	500	600	700	800	900	1000
First Fit	2	8	28	24	34	51	74	87	108	131
Best Fit	1	7	16	27	41	62	91	107	140	165
Best Fit Decreasing	2	11	20	35	52	96	114	141	175	212

Rys. 4.9. Okno prezentujące wyniki eksperymentu

Jest ono podzielone na 4 części. W centralnej części okna (rys. 4.9. poz. 1.) wyświetlany jest wykres, przedstawiający wyniki dla wybranych parametrów. Jest on automatycznie skalowany w zależności od dostępnej przestrzeni. Konkretne wartości aktualnie prezentowanych danych są wyświetlane w postaci etykiet – w przypadku odpowiedniej ilości miejsca. Możliwe jest też sprawdzenie wartości poprzez najechanie kursorem myszy na wybrany słupek (na wykresie słupkowym) bądź też punkt (na wykresie liniowym i punktowym).

Wyboru parametrów wykresu dokonuje się za pomocą list wyboru oraz przycisków w górnej części okna (rys. 4.9. poz. 2.). Pierwsza lista (poz. 3.) pozwala na wybór wyników pogrupowanych wg algorytmów, rozkładów danych czy sposobu sortowania danych. Możliwy jest też wybór grupowania wg par: algorytm algorytm/rozkład, algorytm/sortowanie, rozkład/sortowanie. Pozwala to na zbadanie wpływy danych wejściowych na wyniki uzyskiwane przez zbiór algorytmów, itp. Druga lista wyboru (poz. 4.) umożliwia wybór konkretnego parametru oceny:

- czas działania algorytmu
- jakość rozwiązania wyrażona liczbą otwartych pudełek
- oszacowanie jakości – obliczane jako: $\frac{wynik\_algorytmu}{L_2}$
- oszacowanie błędu – obliczane jako: $\frac{(wynik\_algorytmu - L_2)}{L_2} * 100\%$

Za pomocą przycisku (rys. 4.9. poz. 5.) możliwa jest zmiana typu wykresu na słupkowy, liniowy lub punktowy. Przycisk (poz. 6.) służy do zmiany skali (oś Y) na liniową lub logarytmiczną. Ostatni przycisk umożliwia zapis wykresu do pliku graficznego w jednym z popularnych formatów.

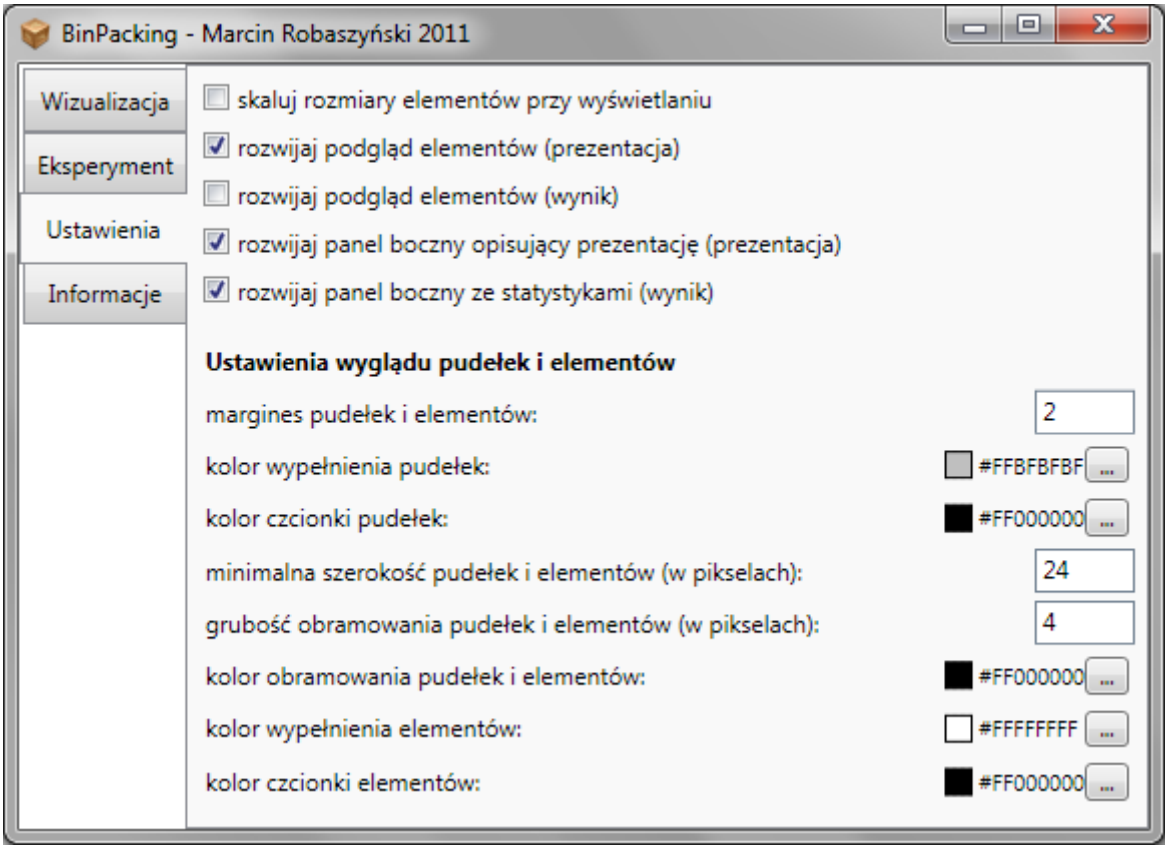
W prawej części okna wyświetlana jest legenda. Umożliwia ona wybór serii danych wyświetlanych na wykresie i w tabeli. W zależności od wybranego parametru oceny (rys. 4.9. poz. 4.) możliwy jest również wybór kilku podstawowych funkcji obrazujących złożoność lub linii obrazujących poziom błędów heurystyk. W prawej górnej części legendy umieszczono również przycisk służący do jej zwijania.

W dolnej części okna znajduje się tabela prezentująca wyniki w postaci liczbowej. Podobnie jak w przypadku legendy, możliwe jest jej zwinięcie. Przyciski w prawej części służą do zapisu tabeli do pliku graficznego bądź też pliku przecinkowego (rozszerzenie .csv), który można otworzyć np. w programie Excel.

4.2.4. Konfiguracja programu

W oknie ustawień (rys. 4.10.) możliwe jest wybranie kilku podstawowych opcji programu:

- *skaluj rozmiary elementów przy wyświetlaniu* – w przypadku wybrania tej opcji rozmiary elementów przy wyświetlaniu będą skalowane (tak aby wielkość pudełka odpowiadała wartości 1)
- *rozwijaj podgląd elementów (prezentacja)* – opcja określa czy po uruchomieniu prezentacji podgląd elementów ma być domyślnie rozwinięty
- *rozwijaj podgląd elementów (wynik)* – opcja o działaniu analogicznym do poprzedniej, jednak dla trybu wyświetlania wyniku
- *rozwijaj panel boczny opisujący prezentację (prezentacja)* – określa czy panel boczny opisujący aktualne kroki algorytmu (w trybie prezentacji) ma być domyślnie rozwinięty
- *rozwijaj panel boczny ze statystykami (wynik)* – opcja o działaniu analogicznym do poprzedniej, dotycząca jednak panelu bocznego w trybie wyświetlania wyniku



Rys. 4.10. Główne okno programu – ustawienia

Ponadto, w dolnej części okna umieszczono grupę kontrolek (*Ustawienia wyglądu pudełek i elementów*), pozwalającą dostosować wygląd pudełek i elementów. Możliwa jest zmiana kolorów wypełnienia i obramowania oraz rozmiarów (szerokości) pudełek i elementów.

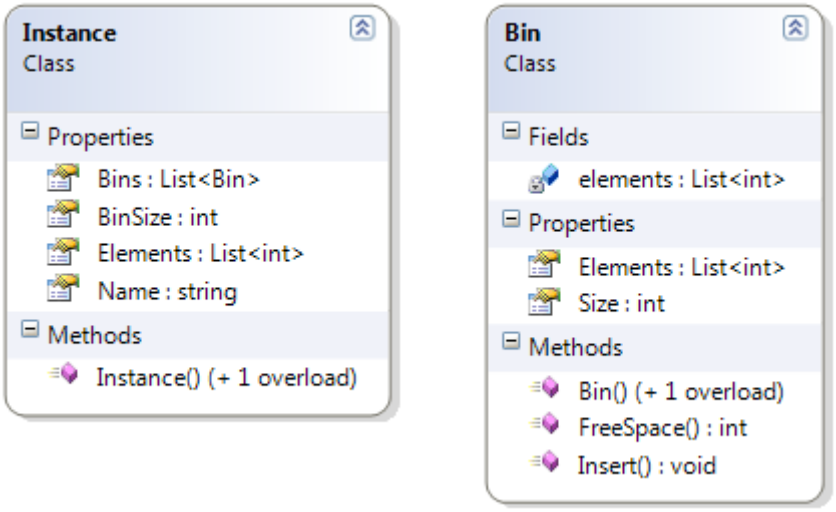
4.3. Dokumentacja techniczna

W tym podrozdziale skupiono się na opisie struktury klas głównych elementów biblioteki *Bin Packing*. Są to kolejno: klasy bazowe (opisujące podstawowe obiekty w problemie pakowania), klasy związane z algorytmami (oraz możliwością ich prezentacji), a także klasy powiązane z generatorem danych i eksperymentem. Na końcu opisano obsługiwane typy plików.

4.3.1. Struktura klas – instancja problemu, pudełko, element

Do opisu problemu pakowania potrzebne są 2 główne obiekty: element oraz pudełko. W stworzonym systemie każdy element jest reprezentowany jako pojedyncza (dodatnia) liczba całkowita. Pudełko jest natomiast reprezentowane przez osobną klasę. Klasa *Bin*, reprezentująca pudełko zawiera listę elementów, które w niej umieszczono oraz rozmiar pudełka (właściwość *Size*). Utworzono również 2 metody pomocnicze: dodającą element do pudełka – *Insert()* i obliczającą pozostałe wolne miejsce w pudełku – *FreeSpace()*.

W trakcie działania algorytmu otrzymywane są rozwiązania częściowe. W celu ich przechowywania utworzono klasę *Instance*. Poza przechowywaniem aktualnego rozwiązania zawiera ona również informacje nt. instancji problemu. Opis instancji składa się z nazwy, rozmiaru pudełka oraz listy elementów. Aktualne rozwiązanie jest reprezentowane przez listę obiektów typu *Bin*. Na początku działania algorytmu lista ta jest pusta. Klasy *Bin* i *Instance* przedstawiono na poniższym rysunku:



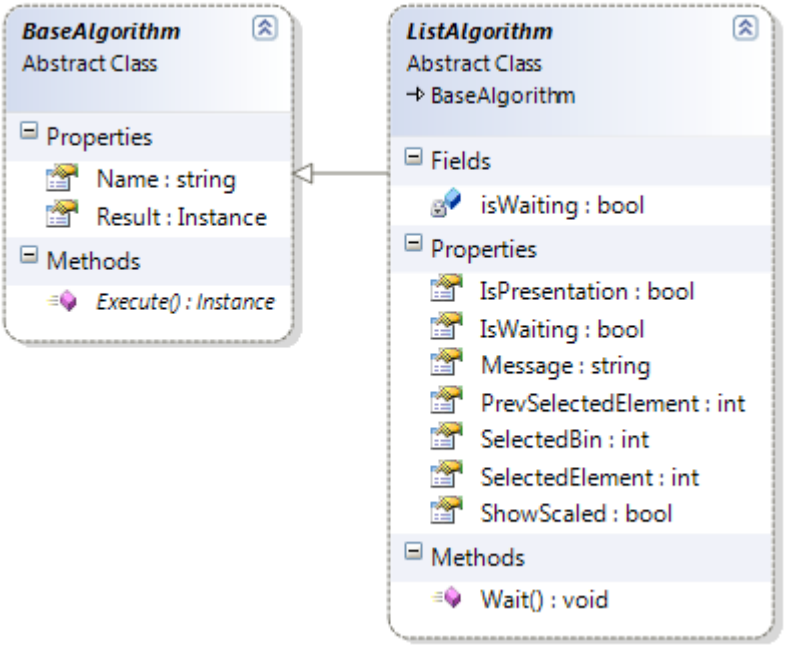
Rys. 4.11. Klasy bazowe

4.3.2. Struktura klas – algorytmy

Algorytmy są reprezentowane poprzez klasy dziedziczące po jednej z dwóch abstrakcyjnych klas – *BaseAlgorithm* lub *ListAlgorithm*.

Pierwsza z nich reprezentuje każdy algorytm i zawiera nazwę algorytmu, wynik działania oraz metodę *Execute()*, wykonującą właściwe obliczenia.

Druga klasa dziedziczy po pierwszej i reprezentuje algorytmy, dla których można przeprowadzać prezentację sposobu działania. Zdecydowano się na rozwiązanie wykorzystujące po jednej implementacji każdego algorytmu (zamiast tworzenia osobnych wersji kodu dla prezentacji sposobu działania i obliczeń podczas eksperymentu). Z tego względu klasa *ListAlgorithm* została rozbudowana o pola *IsPresentation* oraz *IsWaiting* – określają one, odpowiednio, czy algorytm działa w trybie prezentacji i czy znajduje się w trybie oczekiwania. Do zatrzymywania działania algorytmu służy metoda *Wait()*. Czeka ona na zmianę *IsWaiting* (np. na skutek kliknięcia przycisku „Dalej” przez użytkownika). Pola *Message, PrevSelectedBin, SelectedBin, SelectedElement* odpowiadają za wyświetlaną podczas prezentacji sposobu działania algorytmu wiadomość (informację) oraz za element i pudełko, które należy zaznaczyć jako aktualnie rozważane przez algorytm. Pole *ShowScaled* określa czy wartości elementów są skalowane (ma to wpływ na wyświetlane komunikaty). Rys. 4.12. przedstawia klasy *BaseAlgorithm* oraz *ListAlgorithm*.



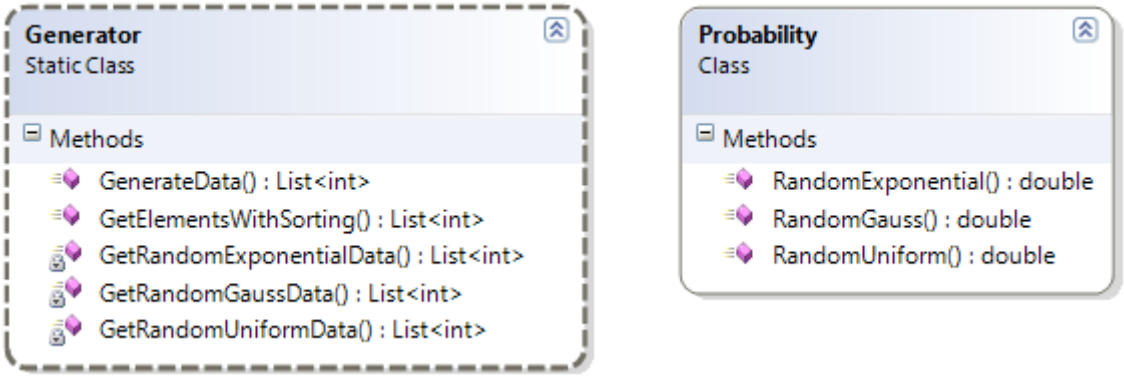
Rys. 4.12. Klasy reprezentujące algorytmy

4.3.3. Struktura klas – generator

Do generowania danych wykorzystywane są 2 klasy, które przedstawiono na rys. 4.14.

Klasa *Probability* zawiera metody zwracające pojedyncze liczby zmiennoprzecinkowe (typ *double*), o różnych rozkładach losowych.

Generowanie list elementów odbywa się za pomocą metod klasy *Generator*. Metoda *GenerateData()* zwraca listę elementów o podanej liczebności, wartościach mieszczących się w zadanym zakresie oraz o zadanym rozkładzie. Metoda *GetElementsWithSorting()* pozwala uzyskać elementy posortowane w różny sposób.



Rys.4.13.Klasy generujące dane

4.3.4. Struktura klas – eksperyment

Największa liczba klas powiązana jest z eksperymentem – przedstawiono je na rys. 4.15.

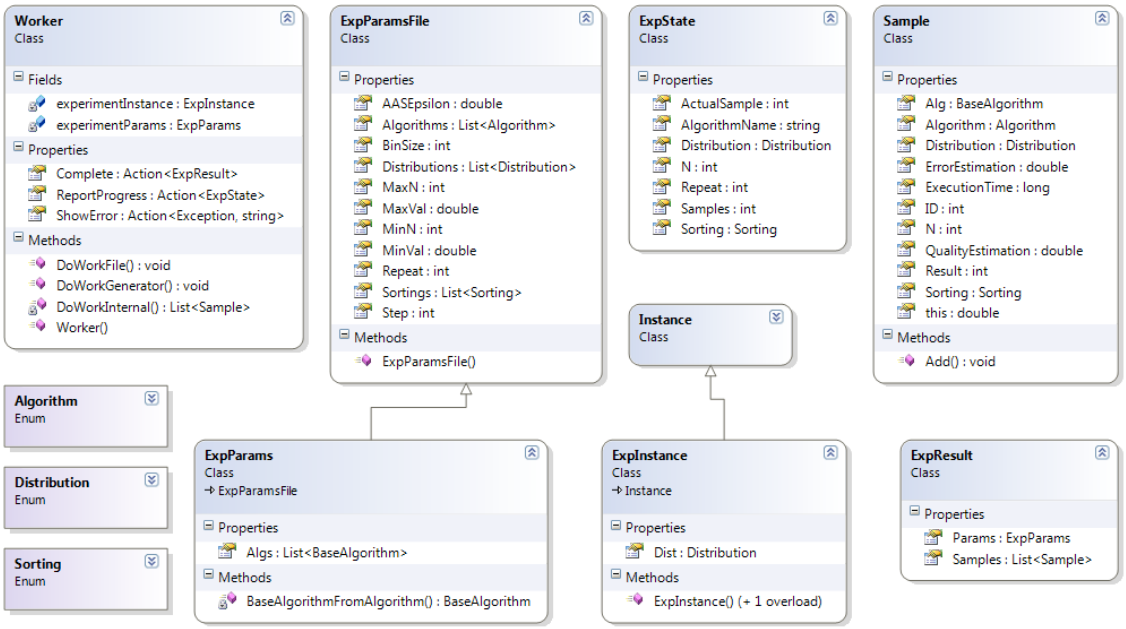
Za obliczenia odpowiadają metody *DoWorkFile()* i *DoWorkGenerator()* klasy *Worker*. Pierwsza obsługuje eksperyment z jedną instancją załadowaną z pliku; druga eksperyment z danymi z generatora (generowanymi podczas eksperymentu).

Niezależnie od źródła danych, do eksperymentu zostają przekazane jego parametry w postaci obiektu typu *ExpParams*. Klasa *ExpParamsFile* odpowiada za ustawienia generatora zapisywane w pliku. *ExpParams*, która po niej dziedziczy, dodaje tylko algorytmy w postaci referencji (pole *Algs*) na podstawie pola *Algorithms*.

Klasa *ExpInstance* dodaje do instancji informację o rozkładzie elementów i jest wykorzystywana wewnątrz eksperymentu i przy ładowaniu plików.

Klasa *ExpState* opisuje aktualny stan eksperymentu czyli podaje informacje o aktualnie uruchomionym algorytmie, sortowaniu, liczbie elementów, itp. Zawiera też numer próbki, co pozwala na określenie postępu eksperymentu oraz na późniejsze obliczanie uśrednionych wyników.

Wyniki eksperymentu są reprezentowane poprzez klasę *ExpResult*, zawierającą parametry eksperymentu oraz zebrane próbki w postaci obiektów klasy *Sample*. Każda próbka przechowuj informacje nt. stanu eksperymentu oraz właściwe dane takie jak czas obliczeń czy oszacowanie jakości rozwiązania.



Rys. 4.14. Struktura klas wykorzystywanych w eksperymencie

4.3.5. Obsługiwane formaty plików

System *Bin Packing* umożliwia odczyt kilku najpopularniejszych typów plików, zawierających instancje problemu pakowania. Poniżej opisano każdy z nich. Na końcu znajduje się informacja o formacie zapisywanych plików.

Prosty plik z pojedynczą instancją

Pierwszy i zarazem najprostszy plik składa się tylko z pojedynczych liczb znajdujących się w kolejnych linijkach. Pierwsze dwie linie mogą (ale nie muszą zawierać wielkość pudełka i/lub liczbę elementów). W przypadku nie podania wielkości pudełka użytkownik wprowadza ją ręcznie. Brak informacji o liczbie elementów powoduje odczytanie pliku do końca (wczytanie wszystkich elementów). Pozostałe linie zawierają rozmiary poszczególnych elementów. Poniżej przedstawiono strukturę pliku (elementy w nawiasach kwadratowych są opcjonalne):

```
[wielkość_pudełka]
[liczba_elementów]
element_1
element_2
...
element_n
```

`gdzie` n oznacza liczbę elementów.

Plik z wieloma instancjami

Drugi obsługiwany typ pliku może zawierać wiele instancji problemu pakowania. W pierwszej linii znajduje się liczba określająca liczbę instancji. W dalszej części pliku zapisane są kolejne instancje.

Pierwsza linia opisu instancji zawiera jej nazwę. W drugiej znajdują się wartości (rozdzielone dowolną liczbą spacji lub tabulatorów) określające: wielkość pudełka, liczbę elementów, wartość najlepszego znanego rozwiązania (ta wartość jest ignorowana). Kolejne linie zawierają rozmiary elementów (po jednym w każdej linii). Przedstawiono to poniżej (wcięcia dodano w celu zwiększenia czytelności):

```
liczba_instancji
  nazwa_instancji_1
  wielkość_pudełka_1 liczba_elementów_1 [najlepsze_znane_rozw_1]
    element_1_1
    element_1_2
    ...
    element_1_n
  nazwa_instancji_2
  wielkość_pudełka_2 liczba_elementów_2 [najlepsze_znane_rozw_2]
    element_2_1
    element_2_2
    ...
    element_2_n
  ...
  nazwa_instancji_m
  wielkość_pudełka_m liczba_elementów_m [najlepsze_znane_rozw_m]
    element_m_1
    element_m_2
    ...
    element_m_n
```

gdzie m oznacza liczbę instancji, natomiast n - liczbę elementów w instancji.

Plik z wieloma instancjami z opisem grup elementów

Ostatni typ pliku również może zawierać wiele instancji. Nie zawiera on informacji o ich liczbie – składa się tylko z kolejnych opisów pojedynczych problemów. Każdy z nich zawiera w pierwszej linii nazwę. Druga linia określa liczbę rozmiarów

elementów (nie muszą to być różne rozmiary). Następne linie zawierają po 2 wartości, oddzielone dowolną liczbą spacji bądź tabulatorów, określające kolejno: rozmiar elementu oraz liczbę elementów o podanym rozmiarze. Schemat pliku znajduje się poniżej:

```
nazwa_instancji_1
  liczba_rozmiarów_1
  wielkość_pudełka_1
    rozmiar_elem_1_1 liczba_elem_1_1
    rozmiar_elem_1_2 liczba_elem_1_2
    ...
    rozmiar_elem_1_n liczba_elem_1_n
nazwa_instancji_2
  liczba_rozmiarów_2
  wielkość_pudełka_2
    rozmiar_elem_2_1 liczba_elem_2_1
    rozmiar_elem_2_2 liczba_elem_2_2
    ...
    rozmiar_elem_2_n liczba_elem_2_n
...
nazwa_instancji_m
  liczba_rozmiarów_m
  wielkość_pudełka_m
    rozmiar_elem_m_1 liczba_elem_m_1
    rozmiar_elem_m_2 liczba_elem_m_2
    ...
    rozmiar_elem_m_n liczba_elem_m_n
```

W tym przypadku m ponownie określa liczbę instancji problemu pakowania, natomiast n liczbę par (rozmiar elementu – liczba elementów o podanym rozmiarze) w instancji.

Format zapisywanych plików

Poza odczytem danych możliwy jest również zapis instancji w pierwszym z opisanych powyżej formatów. W pierwszej linii zapisywana jest wielkość pudełka, w drugiej liczba elementów a w następnych kolejne elementy.

5. EKSPERYMENT OBLICZENIOWY

Eksperyment obliczeniowy został podzielony na 3 główne części. W pierwszej testowane były algorytmy listowe. W osobnym podrozdziale pokazano również wpływ charakterystyki danych wejściowych na wyniki. Druga część skupia się na asymptotycznym schemacie aproksymacyjnym – czasie jego działania i jakości uzyskiwanych rozwiązań, w zależności od parametru ε. W kolejnym podrozdziale porównano algorytmy „niestandardowe" (asymptotyczny schemat aproksymacyjny, algorytm redukcji i *PBI*) z dwoma algorytmami listowymi: dającym najlepsze rezultaty – *Best-Fit Decreasing* i najszybszym – *Next-Fit*.

Ze względu na złożoność w eksperymentach pominięto algorytm dokładny. Przeprowadzone testy wykazały, że możliwe jest rozwiązywanie niektórych instancji liczących nawet kilkaset elementów. Z drugiej strony jednak w niektórych przypadkach kilkadziesiąt elementów wystarczy, aby wydłużyć czas obliczeń do kilku godzin co stawia pod znakiem zapytania stosowanie tego algorytmu w ogóle (oprócz sytuacji, gdy rozwiązanie optymalne jest wymagane).

5.1. Heurystyki listowe

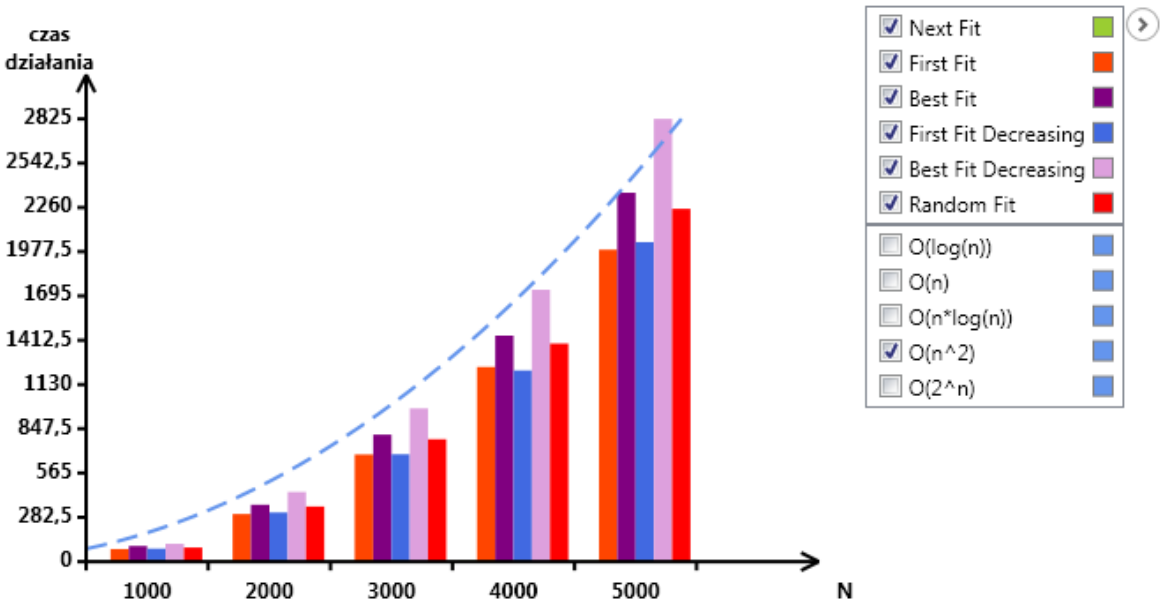
Algorytmy listowe testowano dla następujących parametrów:

od-do (krok)	**rozkłady danych**	**sortowanie elementów**	**rozmiar pudełka**	**rozmiar elementów**	**powtórzenia**
1000-5000 (1000)	normalny jednostajny wykładniczy	brak rosnąco malejąco	100	1-100	4

Tab. 5.1. Parametry eksperymentu testującego heurystyki listowe

Na pierwszym wykresie (rys. 5.2.) przedstawiono średni czas działania poszczególnych algorytmów. W celu ułatwienia analizy wyświetlono również linię obrazującą złożoność obliczeniową $O(n^2)$. Łatwo zauważyć, że czasy obliczeń dla każdego z algorytmów nie rosną szybciej niż ta linia – potwierdza to ich złożoność czasową, przedstawioną w rozdziale 3. Wykres pozwala również zauważyć, że czasy

obliczeń dla algorytmu *Next-Fit* są bardzo małe – z tego powodu nie są widoczne na wykresie.

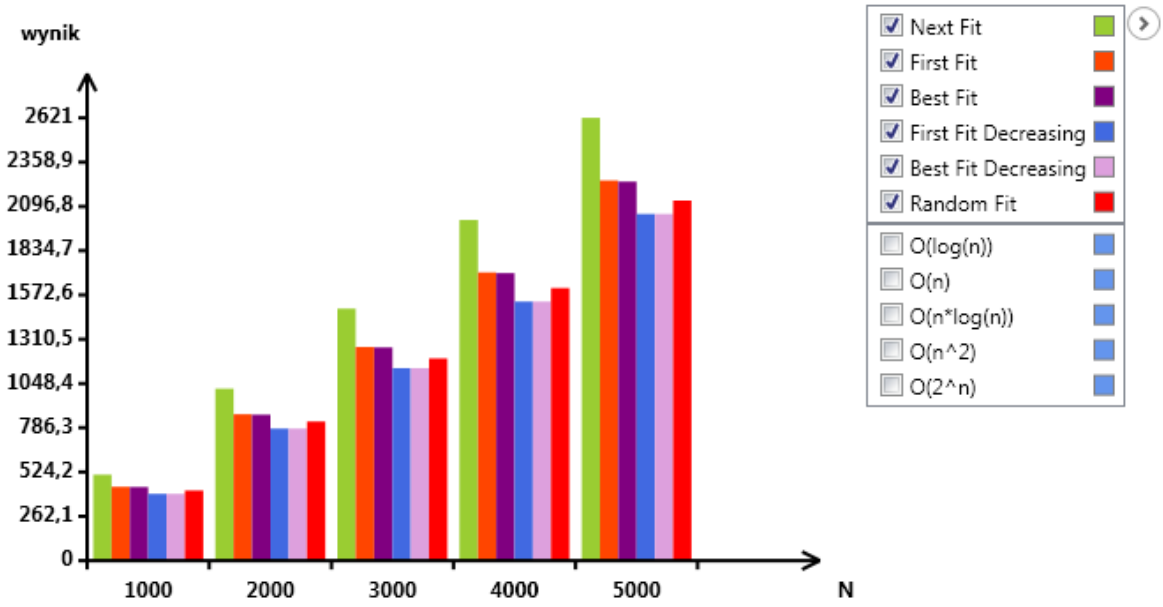


Rys. 5.2. Czas działania algorytmów listowych (w ms)

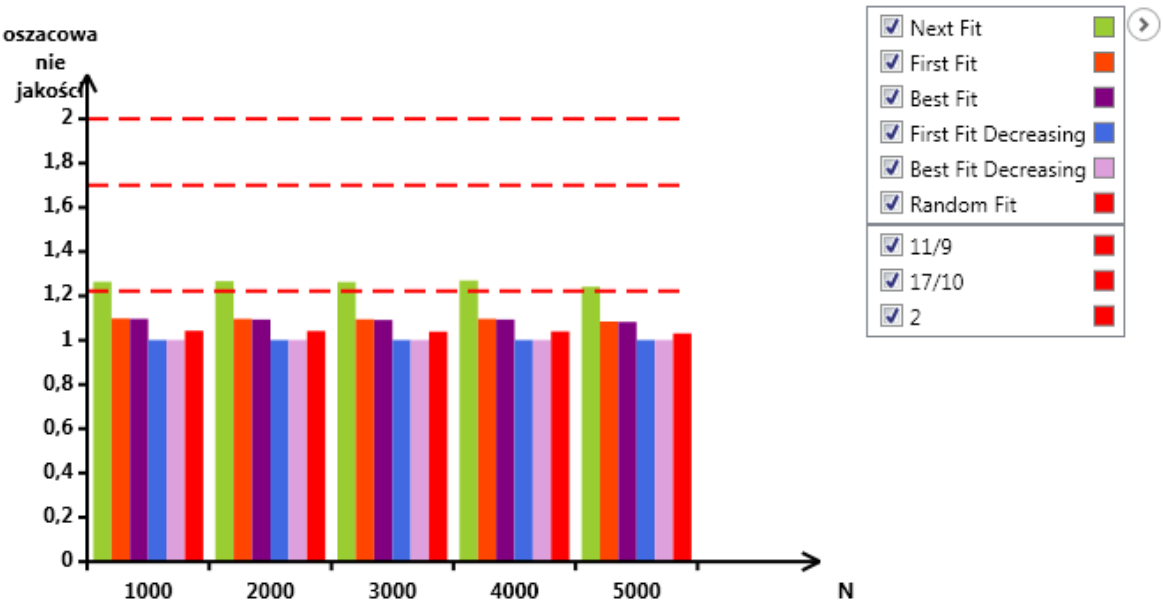
Kolejne wykresy skupiają się na pokazaniu jakości rozwiązań. Pierwszy z nich (rys. 5.3.) przedstawia wyniki (liczbę pudełek) uzyskane przez poszczególne algorytmy. Już na pierwszy rzut oka widać, że najsłabiej wypada algorytm *Next-Fit*, co jest zgodne z przewidywaniami. Znacznie lepiej sprawdzają się pozostałe algorytmy. Wyniki algorytmów *First-Fit* i *Best-Fit* są porównywalne ze sobą zarówno w wersji z sortowaniem elementów w instancjach wejściowych jak i bez. Dobre wyniki uzyskuje również algorytm *Random-Fit* – wynika to z zastosowanej strategii, dodającej nowe pudełka tylko wtedy, gdy jest to konieczne.

Drugi z wykresów (rys. 5.4.) prezentuje oszacowanie jakości rozwiązań generowanych przez algorytmy – im jego wartość jest bliższa 1, tym lepiej. Dodatkowo wyświetlone zostały linie oszacowania asymptotycznego: 2, $\frac{17}{10}$, $\frac{11}{9}$. Jak widać, algorytmy *FFD* i *BFD* znajdują rozwiązanie bardzo bliskie pożądanej wartości, co potwierdza również trzeci wykres, na którym umieszczono oszacowanie błędu. Określa ono o ile procent uzyskane rozwiązanie jest większe od wartości dolnego ograniczenia. Wartości dla wspomnianych algorytmów nie są widoczne ze względu na niewielką różnicę (lub jej

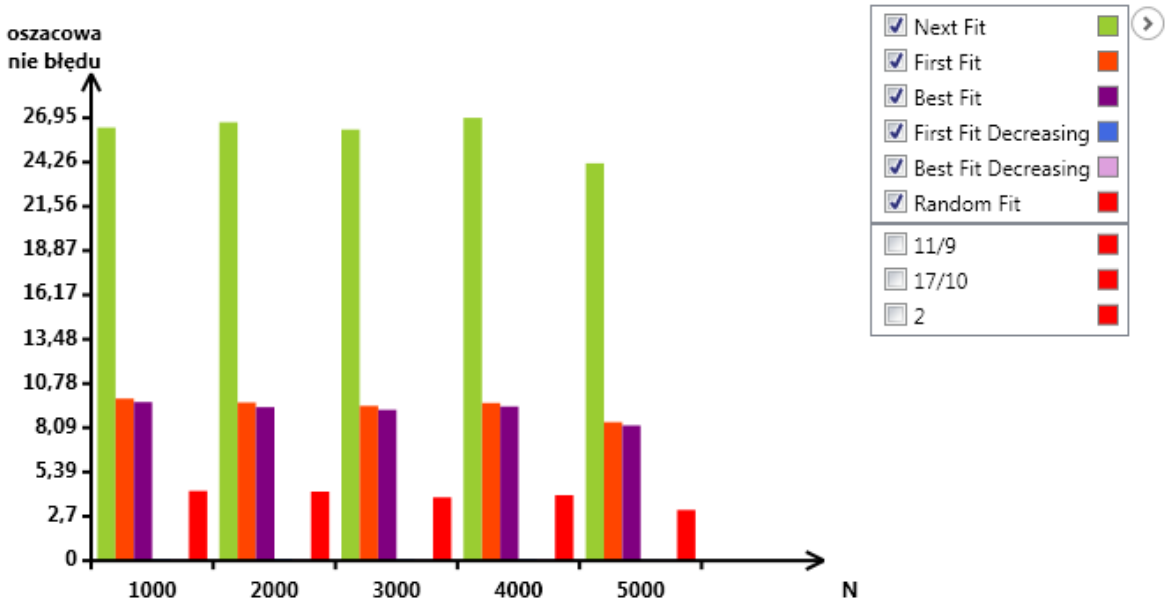
brak) oraz duże rozmiary instancji – różnica na poziomie kilku pudełek nie jest zauważalna.



Rys. 5.3. Wyniki algorytmów listowych



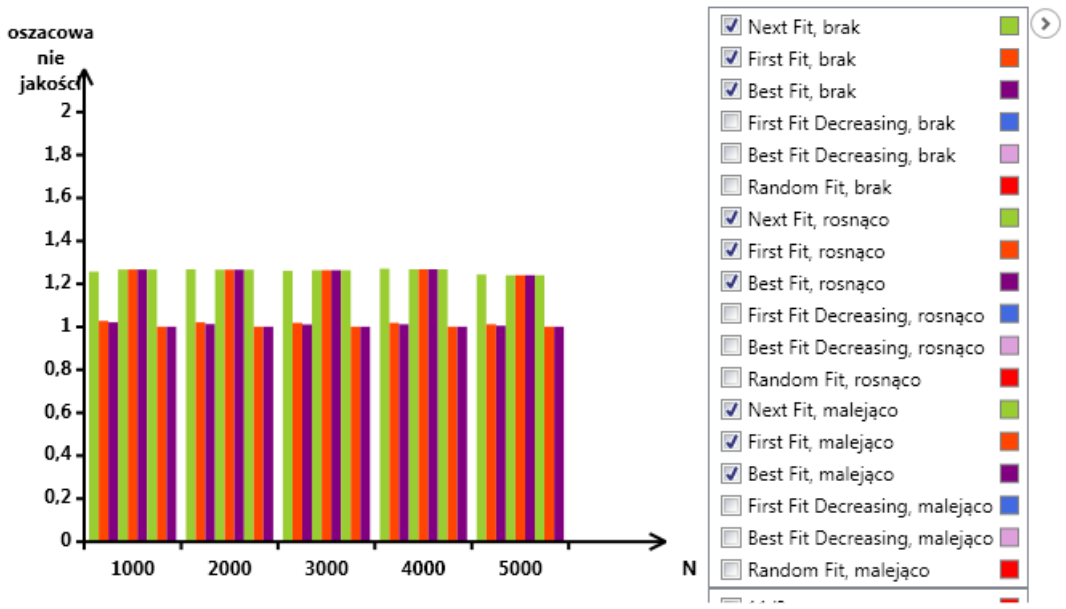
Rys. 5.4.Oszacowanie jakości algorytmów listowych



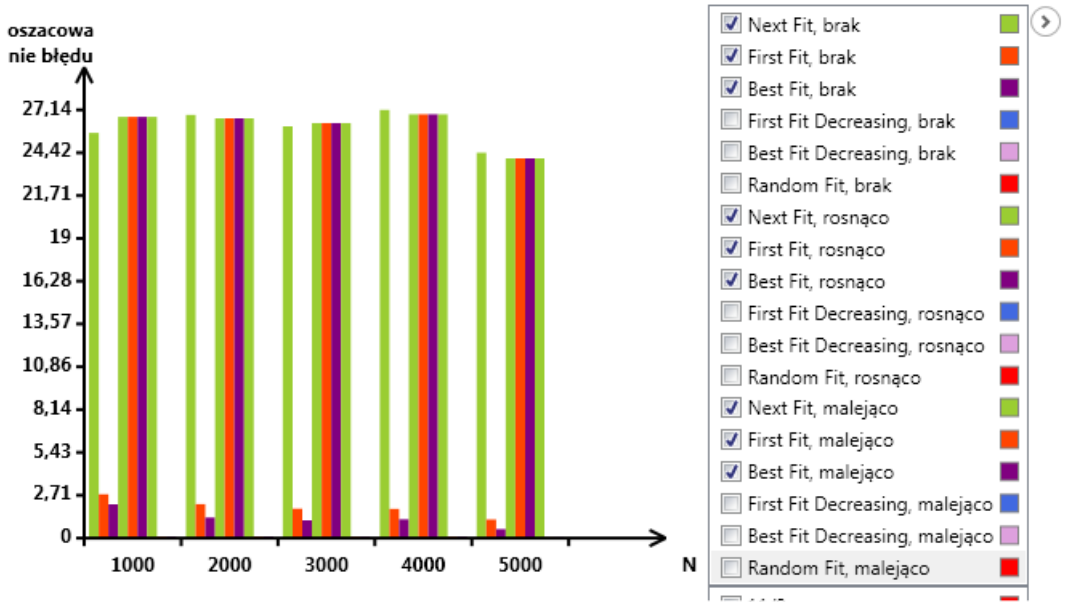
Rys. 5.5.Oszacowanie błędu algorytmów listowych (w %)

5.1.1. Wpływ danych na efektywność algorytmów

W tym podrozdziale skupiono się na wpływie charakterystyki danych wejściowych na uzyskiwane wyniki. Wykorzystane zostały te same wyniki co w poprzednim rozdziale – nie podawano ręcznie zakresu rozmiarów elementów w generatorze. Zamiast tego skupiono się na rozkładzie elementów oraz sposobie ich sortowania w instancji wejściowej. Pominięte zostały wyniki działania algorytmów, dla których kolejność elementów nie ma znaczenia: *Random-Fit*, *First-Fit Decreasing* oraz *Best-Fit Decreasing*.



Rys. 5.6.Oszacowanie jakości w zależności od sortowania elementów w instancji

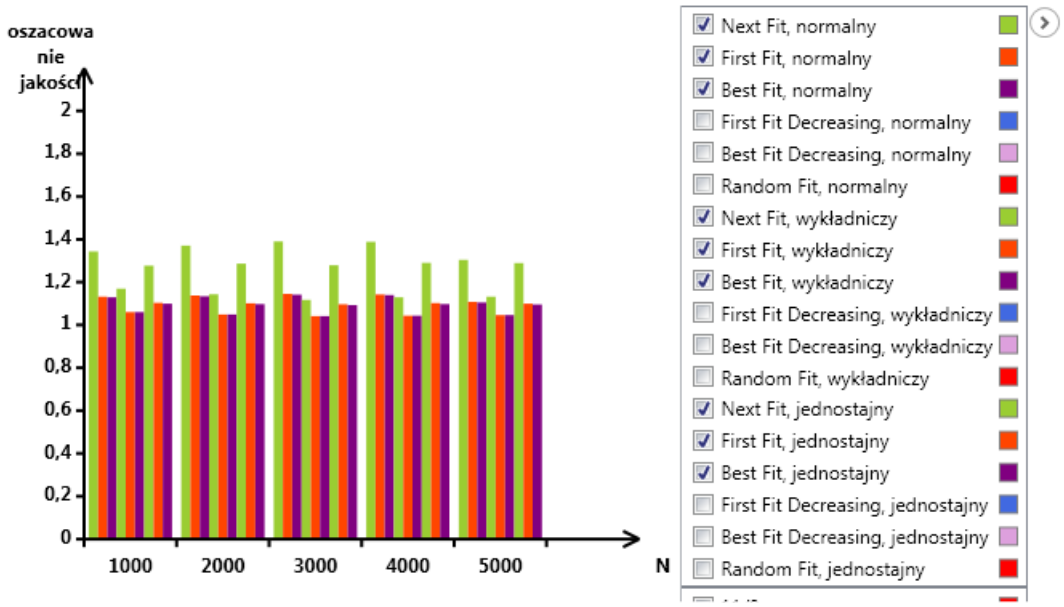


Rys. 5.7.Oszacowanie błędu w zależności od sortowania elementów (w %)

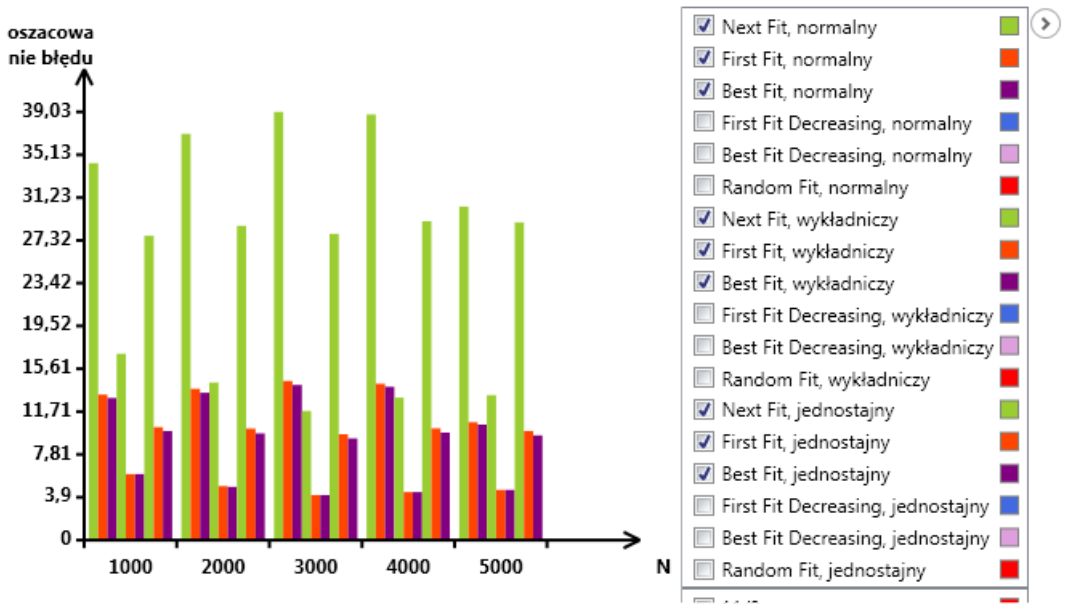
Powyższe wykresy (rys. 5.6. i 5.7.) przedstawiają odpowiednio średnie oszacowanie jakości oraz błędu dla danych posortowanych (lub nie) dla poszczególnych algorytmów. W przypadku algorytmu *Next-Fit* większe znaczenie na jakość uzyskiwanych rozwiązań ma sam fakt posortowania elementów niż to, czy zostały posortowane rosnąco czy malejąco. Zupełnie inaczej jest w przypadku algorytmów *First-Fit* i *Best-Fit* – oba algorytmy dobrze sobie radzą w przypadku gdy elementy są

nieuporządkowane. Z tego powodu posortowanie elementów malejąco nie powoduje już znacznej poprawy wyników. Posortowanie elementów rosnąco powoduje natomiast znaczne pogorszenie jakości uzyskiwanych rozwiązań.

Jak się okazuje, istotnym czynnikiem wpływającym na jakość uzyskanego rozwiązania jest rozkład wielkości elementów. Algorytmy najlepiej sobie radzą z danymi o rozkładzie wykładniczym. Jest to zrozumiałe, ponieważ w takim wypadku mamy niewiele większych elementów i dużo małych, których zapakowanie jest o wiele prostsze. O wiele gorzej jest w przypadku rozkładów jednostajnego i normalnego. W tym drugim przypadku jest najwięcej większych elementów, co utrudnia ich zapakowanie w optymalny sposób. Przedstawiono to na poniższych wykresach. Na pierwszym z nich (rys. 5.8.) zaprezentowano oszacowanie jakości; na drugim (rys. 5.9.) – oszacowanie błędu.



Rys. 5.8.Oszacowanie jakości w zależności od rozkładu elementów



Rys. 5.9.Oszacowanie błędu w zależności od rozkładu elementów (w %)

5.2. Asymptotyczny schemat aproksymacyjny

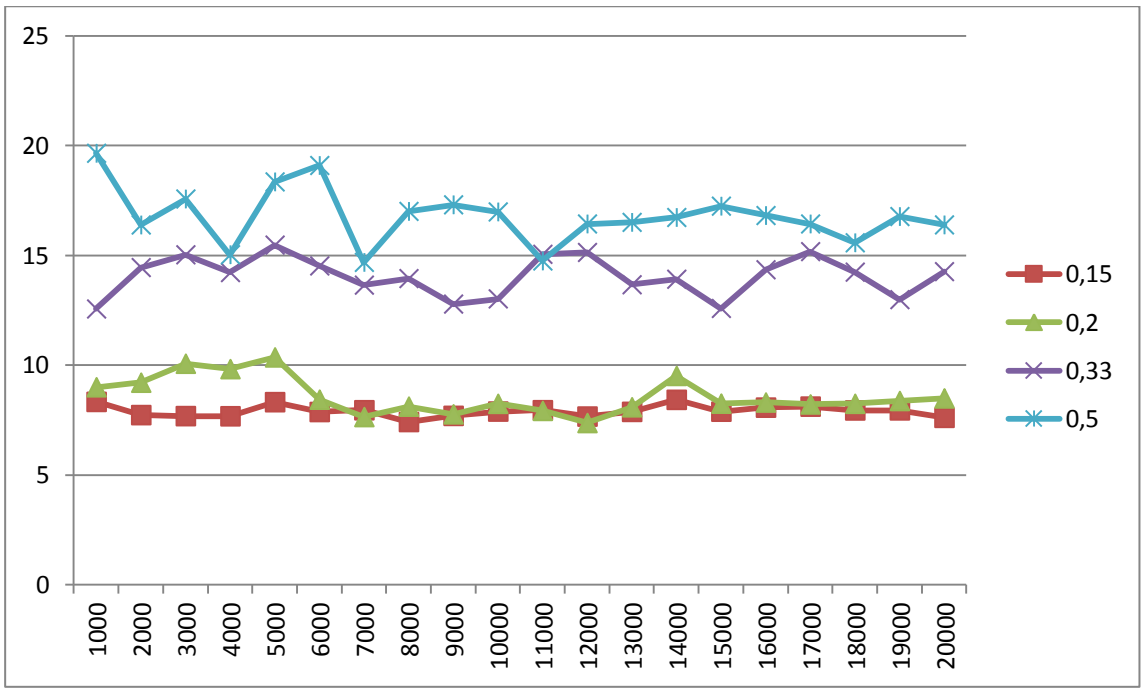
Ze względu na szybkość działania algorytm ten został przetestowany dla znacznie większych zbiorów danych niż algorytmy listowe. Głównym celem eksperymentu było zbadanie wpływu występującego w tej metodzie parametru ε na czas obliczeń i jakość rozwiązania. Stworzony system nie pozwala na przeprowadzenie eksperymentu dla różnych ε jednocześnie. Zamiast tego obliczenia przeprowadzono osobno dla każdej badanej wartości parametru ε a następnie eksportowano wyniki do programu Excel w celu dalszej obróbki. Z tego powodu wykresy pochodzą z tej aplikacji, a nie z systemu *Bin Packing* opisywanego w tej pracy.

Porównanie wyników *AAS* dla różnych wartości ε jest wiarygodne mimo wykorzystania innych zbiorów instancji wejściowych ze względu na wielokrotne powtórzenie eksperymentu i wykorzystanie instancji o dużych rozmiarach.

W tabeli 5.10. zebrano parametry przeprowadzonego w ramach badań eksperymentu. Pierwotnie planowano również test dla $\varepsilon = 0{,}1$ - powodowało to jednak znaczny wzrost liczby możliwych sposobów wypełnienia pudełka, co skutkowało kończeniem się wolnej pamięci operacyjnej.

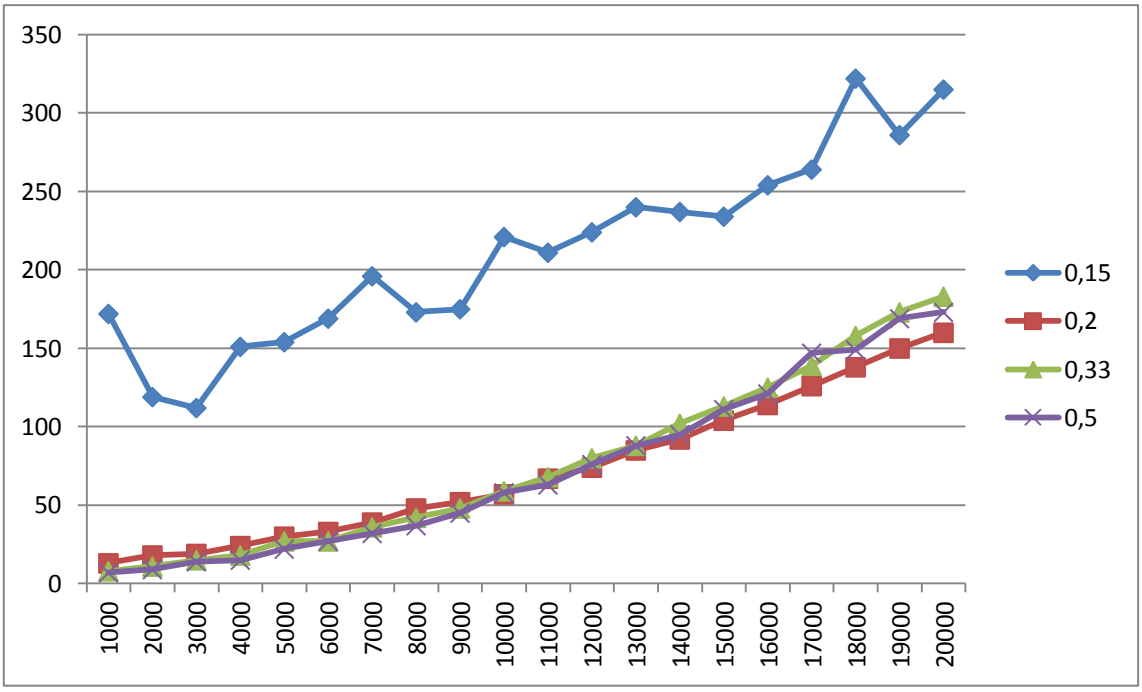
od-do (krok)	**rozkłady danych**	**sortowanie elementów**	**rozmiar pudełka**	**rozmiar elementów**	**powtórzenia**	ε
1000-20000 (1000)	normalny jednostajny wykładniczy	brak rosnąco malejąco	100	1-100	4	0,15 0,2 0,33 0,5

Tab. 5.10. Parametry eksperymentu testującego schemat aproksymacyjny



Rys. 5.11. Oszacowanie błędu w zależności od ε (w %)

Powyższy rysunek przedstawia oszacowanie błędu dla różnych wartości ε - zmniejszanie wartości parametru powoduje poprawę uzyskiwanych rozwiązań i to znaczną. Wraz ze wzrostem dokładności czas obliczeń znacznie wzrasta. Jest to widoczne na poniższym rysunku, przedstawiającym czas obliczeń, gdzie nieznaczne zmniejszenie ε z 0,2 do 0,15 spowodowało „skok” czasu obliczeń:



Rys. 5.12. Czas obliczeń w zależności od ε (w ms)

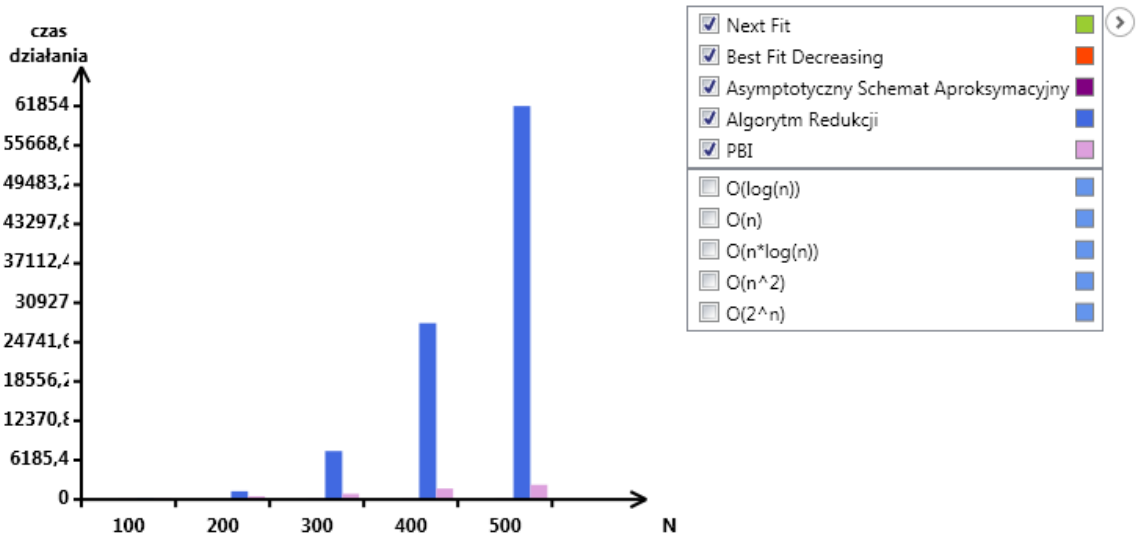
5.3. Porównanie różnych typów algorytmów

Ze względu na długi czas działania jednego z algorytmów, w ostatnim eksperymencie generowane instancje testowe były mniejsze niż omówione poprzednio. Bazując na wcześniejszym eksperymencie dla schematu aproksymacyjnego przyjęto wartość $\varepsilon = 0{,}2$.

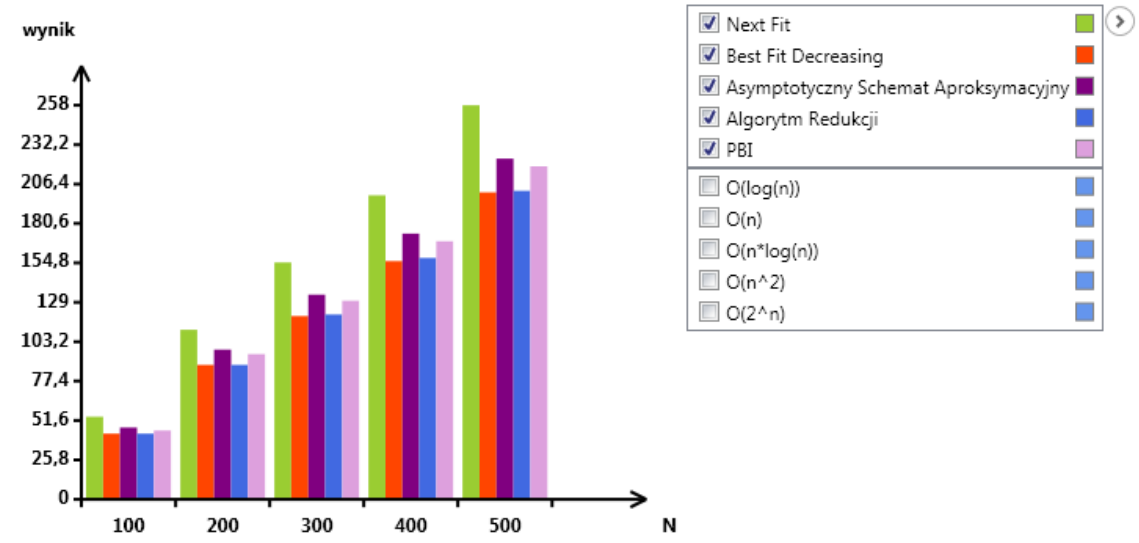
od-do (krok)	**rozkłady danych**	**sortowanie elementów**	**rozmiar pudełka**	**rozmiar elementów**	**powtórzenia**
100-500 (100)	normalny jednostajny wykładniczy	brak rosnąco malejąco	100	1-100	4

Tab. 5.13. Parametry eksperymentu testującego różne typy algorytmów

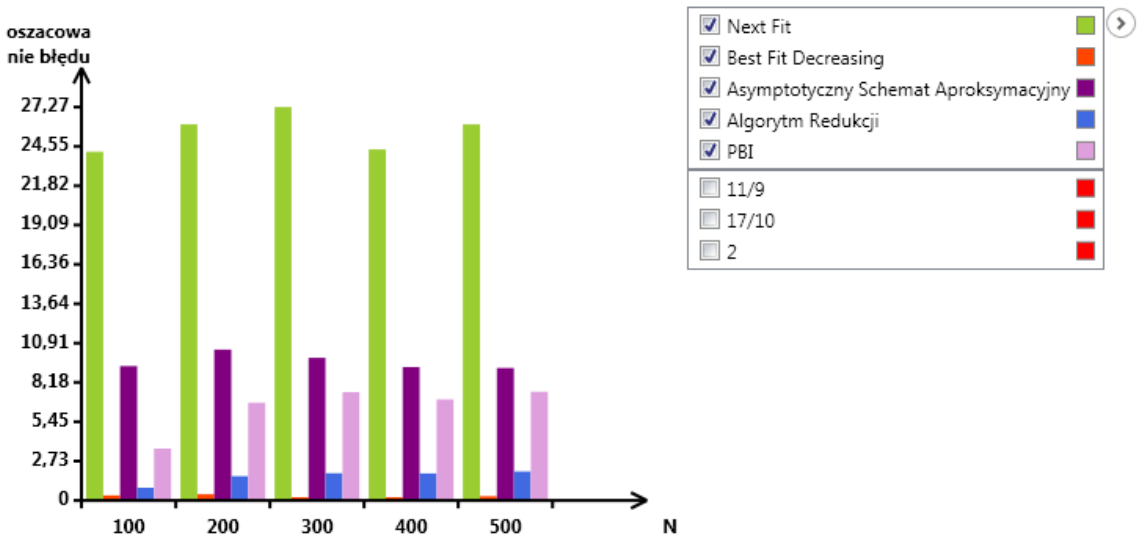
Wspomnianym czasochłonnym algorytmem był algorytm redukcji – czas jego działania wzrastał bardzo szybko wraz ze wzrostem liczby elementów, co pokazuje wykres na rys. 5.14. Jest to spowodowane dużą złożonością obliczeniową $O(n^3)$, wspomnianą w rozdziale o algorytmach. Czas obliczeń dla pozostałych algorytmów był znacznie krótszy – na wykresie widoczne są jeszcze tylko czasy dla algorytmu *PBI*.



Rys. 5.14.Czas obliczeń algorytmów (w ms)



Rys. 5.15.Wyniki uzyskane przez algorytmy



Rys. 5.16.Oszacowanie błędu algorytmów (w %)

Na dwóch kolejnych wykresach (rys. 5.15. i 5.16.) widoczne są różnice w jakości uzyskiwanych rozwiązań. Zdecydowanie najlepiej wypadają tutaj *BFD* oraz algorytm redukcji. Całkiem dobre wyniki osiąga zaproponowany algorytm, *PBI*. Biorąc jednak pod uwagę czas obliczeń (lub dokładność) lepszym wyborem okazuje się schemat aproksymacyjny uzyskujący podobne wyniki w znacznie krótszym czasie bądź też wspomniany *BFD*, który czasowo jest niewiele gorszy, ale osiąga znacznie lepsze rezultaty. Algorytm *Next-Fit* zdaje się mieć zastosowanie głównie dla bardzo dużych instancji, dla których obliczenia związane z asymptotycznym schematem aproksymacyjnym będą trwały zbyt długo lub będą wymagały zbyt dużej ilości pamięci.

6. PODSUMOWANIE

W opracowanym systemie *Bin Packing* udało się osiągnąć w zasadzie wszystkie z postawionych zadań. Udało się zaimplementować zarówno najbardziej znane, proste algorytmy jak i kilka bardziej skomplikowanych – w tym korzystający z programowania liniowego asymptotyczny schemat aproksymacyjny.

Zagwarantowano również możliwość prostego i szybkiego generowania instancji problemu o zadanych parametrach. Elementy mogą być losowane z różnymi rozkładami wielkości elementów. Umożliwiono również odczyt plików w kilku najczęściej spotykanych formatach, w tym tych zawierających wiele instancji problemu. Poza obliczaniem wyniku i jego wyświetlaniem możliwe jest również prześledzenie sposobu działania najbardziej znanych algorytmów.

Za pomocą systemu *Bin Packing* możliwa jest również analiza i porównywanie ze sobą algorytmów. Umożliwia to moduł eksperymentu obliczeniowego, w którym zawarto generator danych, dający możliwość generowania danych testowych spełniających określone warunki. Wyniki eksperymentu prezentowane są w postaci wykresów i/lub tabel. W zależności od potrzeb można skorzystać ze skali logarytmicznej oraz z różnych typów wykresu. Poza porównaniem algorytmów umożliwiono również badanie wpływu rozkładu danych bądź sortowania elementów w instancji na wyniki algorytmów. Uzyskane rezultaty można wyeksportować do zewnętrznego pliku, możliwego do odczytania np. za pomocą programu Excel.

Aby móc lepiej wykorzystać program dano użytkownikowi możliwość zapisania wielu elementów do pliku graficznego. Dotyczy to m.in. podglądu elementów, wykresów, tabel, wyników działania.

Wprowadzono również kilka usprawnień, ułatwiających pracę z programem. Pierwsze to zapamiętywanie większości wprowadzonych parametrów konfiguracyjnych – dzięki temu przy następnym uruchomieniu programu możliwe jest szybkie wznowienie pracy. Dodatkowo wiele elementów interfejsu można zwinąć, dzięki czemu nie zajmują niepotrzebnie miejsca na ekranie.

Prace nad programem pozwoliły autorowi powiększyć wiedzę nt. technologii *WPF* oraz platformy *.NET*. Dotyczy to w szczególności rysowania, wykorzystania grupowania w zapytaniach *LINQ* oraz wykorzystywania wielowątkowości.

Jeżeli chodzi o sam problem pakowania, to warto zauważyć, ze pomimo jego dużej złożoności istnieje wiele algorytmów, które znajdują rozwiązania bardzo zbliżone do optymalnego. Najlepszym przykładem są algorytmy *First-Fit Decreasing*, *Best-Fit Decreasing*, których złożoność obliczeniowa (kwadratowa w najpopularniejszej implementacji) pozwala na stosowanie ich do nawet naprawdę dużych problemów. Poza tym są one bardzo proste, a jakość uzyskiwanych przez nie rozwiązań w większości przypadków jest bliska wartości optymalnej i powinna okazać się wystarczająca w wielu zastosowaniach. Algorytmy listowe typu *on-line* osiągają z reguły wyniki gorsze. Ich praktyczne zastosowanie ogranicza się do przypadków, w których nigdy nie będą znane wszystkie elementy jednocześnie. O ile szybkość działania nie jest najważniejsza, w takim wypadku lepiej wykorzystać algorytm *First-Fit* lub *Best-Fit*, które pozwalają uzyskać lepsze wyniki.

Bardzo dobrym algorytmem okazał się zaprezentowany asymptotyczny schemat aproksymacyjny. Jego główne atuty to duża szybkość działania oraz możliwość określenia pożądanej jakości uzyskiwanych rozwiązań. Trzeba jednak pamiętać, że wykorzystuje on programowanie liniowe, którego implementacja w niektórych przypadkach może być bardzo trudna bądź też niemożliwa.

Pozostałe algorytmy wypadają gorzej od poprzedników. Algorytm dokładny zwraca optymalne rozwiązania, jednak czas jego działania praktycznie wyklucza zastosowanie go w nawet niewielkich problemach. Algorytm redukcji oferuje bardzo dobre wyniki, jednak są one gorsze od wyników uzyskiwanych przez metodę *Best-Fit Decreasing*, a czas jego działania jest znacznie dłuższy. Z kolei algorytm *Next-Fit* z dodatkową optymalizacją działa w czasie porównywalnym, ale uzyskuje znacznie gorsze wyniki.

LITERATURA

[BŁAŻ 1982] Błażewicz J., Cellary W., Słowiński R., Węglarz J., Badania operacyjne dla informatyków.
WNT, Warszawa, 1983

[BŁAŻ 1988] Błażewicz J., Złożoność obliczeniowa problemów kombinatorycznych.
WNT, Warszawa, 1988

[EILO 1971] Eilon S., Christofides N., The loading problem, Management Science 17, 1971, s. 259-267

[FERN 1981] Fernandez de la Vega W., Lueker G.S., Bin packing can be solved within $1 + \varepsilon$ in linear time, w: Combinatorica 1, 1981, s. 349-355

[FUKA 2007] Fukanaga A.S., Korf R.E., Bin completion algorithms for multicontainer packing, knapsack, and covering problems, Journal of Artificial Intelligence Research 28, 2007, s. 393-429

[GARE 1979] Garey M.R., Johnson D.S., Computers and intractability: a guide to the theory of NP-completeness
Freeman, San Francisco, 1979

[KORT 2000] Korte B., Vygen J., Combinatorial optimization. Theory and algorithms.
Springer, Berlin, 2000

[MART 1990] Martello S., Toth P., Knapsack problems. Algorithms and computer implementations.
Wiley, New York, 1990

[MING 2008] Ming-Yang K. (Ed.), Encyclopedia of Algorithms.
Springer, New York, 2008

[PAPA 1982] Papadimitriou C.H, Steiglitz K., Combinatorial Optimization.
Prentice-Hall, Englewood Cliffs, NJ, 1982

ZAŁĄCZNIKI

- płyta z programem, instalatorem, kodem źródłowym i pracą w wersji elektronicznej.