INSTYTUT INFORMATYKI

WYDZIAŁ INFORMATYKI

POLITECHNIKA POZNAŃSKA

PRACA DYPLOMOWA MAGISTERSKA

**PROBLEM PAKOWANIA**

**- BIBLIOTEKA ALGORYTMÓW**

Marcin ROBASZYŃSKI

Promotor:

dr hab. inż. Małgorzata STERNA

Poznań, 2011 r.

KARTA PRACY DYPLOMOWEJ

SPIS TREŚCI

[KARTA PRACY DYPLOMOWEJ 3](#_Toc304507914)

[SPIS TREŚCI 5](#_Toc304507915)

[1. WSTĘP 7](#_Toc304507916)

[1.1. Cel i zakres pracy 8](#_Toc304507917)

[2. PROBLEM PAKOWANIA 10](#_Toc304507918)

[2.1. Sformułowanie problemu 10](#_Toc304507919)

[2.2. Dowód NP-zupełności 12](#_Toc304507920)

[2.3. Przegląd literatury 13](#_Toc304507921)

[3. ALGORYTMY 15](#_Toc304507922)

[3.1. Wstęp 15](#_Toc304507923)

[3.2. Dolne ograniczenia 15](#_Toc304507924)

[3.3. Algorytm dokładny 16](#_Toc304507925)

[3.4. Algorytmy listowe 19](#_Toc304507926)

[3.4.1. Algorytm następnego dopasowania (ang. *Next-Fit*) 19](#_Toc304507927)

[3.4.2. Algorytm pierwszego dopasowania (ang. *First-Fit*) 20](#_Toc304507928)

[3.4.3. Algorytm najlepszego dopasowania (ang. *Best-Fit*) 22](#_Toc304507929)

[3.4.4. First-Fit Decreasing (ang. *First-Fit Decreasing*) 23](#_Toc304507930)

[3.4.5. Best-Fit Decreasing (ang. *Best-Fit Decreasing*) 24](#_Toc304507931)

[3.4.6. Algorytm losowego dopasowania (ang. *Random-Fit*) 25](#_Toc304507932)

[3.5. Algorytm redukcji 26](#_Toc304507933)

[3.5.1. Kryterium dominacji 28](#_Toc304507934)

[3.6. Asymptotyczny schemat aproksymacyjny 29](#_Toc304507935)

[3.6.1. Podział elementów na listy 31](#_Toc304507936)

[3.6.2. Wykorzystanie programowania liniowego 31](#_Toc304507937)

[3.7. Własny algorytm 32](#_Toc304507938)

[4. IMPLEMENTACJA 34](#_Toc304507939)

[4.1. Opis systemu 34](#_Toc304507940)

[4.1.1. Architektura 34](#_Toc304507941)

[4.1.2. Wykorzystane technologie 35](#_Toc304507942)

[4.1.3. Wymagania funkcjonalne i pozafunkcjonalne 35](#_Toc304507943)

[4.1.4. Wymagania sprzętowe i systemowe 36](#_Toc304507944)

[4.2. Dokumentacja użytkownika 37](#_Toc304507945)

[4.2.1. Główne okno aplikacji 37](#_Toc304507946)

[4.2.2. Moduł wizualizacji 38](#_Toc304507947)

[4.2.3. Moduł eksperymentu obliczeniowego 43](#_Toc304507948)

[4.2.4. Ustawienia 47](#_Toc304507949)

[4.3. Dokumentacja techniczna 48](#_Toc304507950)

[4.3.1. Struktura klas – instancja problemu, skrzynka, element 48](#_Toc304507951)

[4.3.2. Struktura klas – algorytmy 49](#_Toc304507952)

[4.3.3. Struktura klas – generator 50](#_Toc304507953)

[4.3.4. Struktura klas – eksperyment 51](#_Toc304507954)

[4.3.5. Obsługiwane formaty plików 52](#_Toc304507955)

[5. EKSPERYMENT OBLICZENIOWY 55](#_Toc304507956)

[5.1. Heurystyki listowe 55](#_Toc304507957)

[5.2.1. Wpływ danych na wyniki 58](#_Toc304507958)

[5.2. Asymptotyczny schemat aproksymacyjny 60](#_Toc304507959)

[5.3. NF, BFD i inne 62](#_Toc304507960)

[5.4. Algorytm dokładny 64](#_Toc304507961)

[6. PODSUMOWANIE 65](#_Toc304507962)

[7. LITERATURA 67](#_Toc304507963)

[8. ZAŁĄCZNIKI 68](#_Toc304507964)

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, lepszej 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 takiej samej ilości towarów do mniejszej liczby pojazdów, skrzynek czy kontenerów. Znacząco obniża to koszty transportu, zwiększając w ten sposób końcowy zysk i konkurencyjność. Inny przykład stanowią wszelkie części, które wycina się z większej ilości materiału (np. z płatów blachy). W takim wypadku wykorzystanie pozostałych fragmentów często jest drogie lub wręcz niemożliwe ze względu na utratę pewnych właściwości podczas obróbki.

Podobnych przykładów można by wymienić bardzo wiele. Większość z nich można opisać jako wypełnianie pewnych pojemników elementami. Oczywiście istnieją algorytmy, rozwiązujące podobne problemy. W zależności od tego, co jest głównym celem, wyróżnić można m.in. problemy plecakowe (umieszczenie w plecaku o określonej pojemności elementów o jak największej sumarycznej wartości), pokrycia (wypełnienie jak największej liczby pojemników dostępnymi elementami) czy pakowania (umieszczenie 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 rozwiązania, 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 programu-biblioteki algorytmów rozwiązujących jednowymiarowy problem pakowania. Głównym zastosowaniem systemu ma być 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.

Poza prezentowaniem działania oraz wyników uzyskiwanych przez poszczególne algorytmy, system miał umożliwiać również ich prostą analizę. W tym celu należało opracować moduł eksperymentu obliczeniowego, umożliwiającego testy dla większej liczby większych instancji. Miał on też umożliwiać porównanie wybranych algorytmów za pomocą wykresów.

Niniejszy dokument przedstawia projekt koncepcyjny i techniczny oraz efekt realizacji aplikacji wraz z przykładowymi instancjami problemu. Stworzoną aplikację nazwano *Bin Packing*.

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 wykazano również NP-zupełność problemu i opisano konsekwencje.

W rozdziale 3. opisano poszczególne algorytmy zaimplementowane w systemie. Dla każdego z nich przedstawiono ideę działania oraz wady i zalety stosowanego podejścia. Podano również złożoność oraz oszacowanie jakości (jeżeli jest znane). Dodatkowo przedstawiono również wyniki działania dla instancji testowej. Ten rozdział zawiera również opis kryterium dominacji.

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

Przykładowy eksperyment obliczeniowy 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), wnioski.

2. PROBLEM PAKOWANIA

2.1. Sformułowanie problemu

**Formalna definicja (jednowymiarowego) problemu pakowania:**

*Dany jest zbiór elementów o (dodatnich) rozmiarach oraz pudełek o ustalonej pojemności . Należy zapakować elementy do pudełek nie przekraczając ich pojemności w taki sposób, aby liczba wykorzystanych pudełek była minimalna.*

Zgodnie z [MART 1990] można sformułować problem następująco:

|  |  |  |  |
| --- | --- | --- | --- |
| min |  |  | |
| przy ograniczeniach: |  |  | |
|  |  |  | |
|  |  |  |  |
|  |  |  | |
|  |  |  |  |

Niniejsza praca skupia się na zdefiniowanej powyżej jednowymiarowej wersji problemu. Oznacza to, że pod uwagę bierze się tylko 1 wymiar (z reguły wysokość) elementów. Istnieją jednak inne warianty, 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 (skrzynek) pośrednio minimalizuje się również pozostałą w skrzynkach 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 jeden z wariantów tzw. problemu rozkroju (ang. *cutting problem*). W zależności od wariantu 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.



Rys. 2.1. Przykładowe rozwiązanie 1-wymiarowego problemu pakowania



Rys. 2.2. Przykładowe rozwiązanie 2-wymiarowego problemu pakowania (źródło: [WWW1])



Rys. 2.3. Przykładowe rozwiązanie 3-wymiarowego problemu pakowania (źródło: [WWW1])

Problem pakowania znajduje zastosowanie m.in. w:

* transporcie – załadunek towarów do jak najmniejszej liczby kontenerów czy samochodów dostawczych
* przydziale pamięci – w systemach ze stronicowaniem pamięci
* produkcji – np. cięcie arkuszy papieru, blachy, itp.

2.2. Dowód NP-zupełności

Problem pakowania jest silnie NP-zupełny.

DOWÓD:

Jednym ze znanych problemów silnie NP-zupełnych jest tzw. problem trójpodziału (ang. *3-partition problem*) [STER]:

Dany jest zbiór elementów o rozmiarach , dla oraz ograniczenie . Ponadto, dla każdego zachodzi: oraz: .

Czy istnieje podział zbioru na rozłącznych podzbiorów taki że dla ?

Wystarczy zauważyć, że problem trójpodziału jest szczególnym przypadkiem problemu pakowania, co udowadnia silną NP-zupełność tego ostatniego.

Oznacza to, że nie istnieje algorytm wielomianowy, znajdujący rozwiązanie optymalne w ogólnym przypadku. Z tego powodu stosuje się heurystyki, znajdujące rozwiązanie przybliżone. Innym stosowanym rozwiązaniem są asymptotyczne schematy aproksymacyjne, które gwarantują znalezienie rozwiązania o zadanej dokładności w czasie zależnym od jej wzrostu. Zaletą takiego podejścia jest możliwość ustalenia kompromisu pomiędzy dokładnością a czasem obliczeń.

2.3. Przegląd literatury

Jedną z najbardziej znanych i często wykorzystywanych prac jest [MART 1990], napisana przez S. Martello i P. Totha. Jeden z rozdziałów poświęcony jest opisowi problemu pakowania. Dokonano w nim przeglądu najpopularniejszych algorytmów aproksymacyjnych wraz z oszacowaniem ich jakości. Wprowadzono również nowe, silniejsze dolne ograniczenia oraz procedurę redukcji, kryterium dominacji. Rozwiązania te wykorzystuje przedstawiony algorytm dokładny. Pozwalają one na znaczne zredukowanie przestrzeni rozwiązań w stosunku do najpopularniejszego algorytmu dokładnego, zaproponowanego w [EILO 1971] przez S. Eilona oraz N. Christofidesa (wykorzystuje on najprostsze dolne ograniczenie ). Dodatkowo, zaprezentowane metody zostały dołączone w postaci kodów źródłowych (niestety w języku Fortran).

Inny algorytm dokładny został zaprezentowany przez A. Fukanagę i R. Korfa w [FUKA 2007]. W odróżnieniu od poprzednich algorytmów, autorzy proponują rozwiązanie przeszukujące przestrzeń możliwych sposobów wypełnienia skrzynek (a nie miejsc, w których można umieścić pojedynczy element). Opisywana jest też metoda efektywnego wyszukiwanie niezdominowanych wypełnień oraz samo kryterium dominacji – w sposób bardziej przystępny niż w [MART 1990]. Zaprezentowano również kryteria dominacji dla problemów „pokrewnych”, m.in. problemu pokrycia (wypełnienia) skrzynek (ang. *bin covering problem*).

Ciekawą pozycję stanowi również [KORT 2000] autorstwa B. Korte i J. Vygena. Oprócz kilku podstawowych heurystyk listowych autorzy opisują również asymptotyczny schemat aproksymacyjny, zaproponowany przez W. Fernandez de la Vegę oraz G. Luekera w [FERN 1981].

Dowód NP-zupełności został przedstawiony m.in. w [STER], [BŁAŻ 1988] oraz [MART 1990]. W pierwszej pozycji przedstawione zostały również heurystyki listowe (wraz z przykładami) oraz zastosowanie i powiązania problemu pakowania z innymi algorytmami.

3. ALGORYTMY

3.1. Wstęp

W poniższym rozdziale przedstawiono zaimplementowane w systemie algorytmy. Jako oszacowanie jakości algorytmów wybrano asymptotyczny stosunek najgorszego rozwiązania (ang. *asymptotic worst-case performance ratio*), opisany w [MART 1990]. Dla algorytmu aproksymacyjnego definiuje się go jako minimalną liczbę rzeczywistą taką, że dla pewnej całkowitej dodatniej liczby :

dla wszystkich instancji , spełniających warunek , gdzie oznacza rozwiązanie optymalne.

Dodatkowo, dla porównania algorytmów przedstawiono wynik ich działania dla instancji testowej. Składa się ona z 11 elementów; wielkość skrzynki – 10.



Rys. 3.1. Instancja testowa

3.2. Dolne ograniczenia

W omawianym systemie zaimplementowano 2 dolne ograniczenia, przedstawione w [MART 1990]. Pierwsze z nich, jest bardzo proste – jego wartość stanowi cecha górna (lub sufit) sumy wszystkich elementów, podzielonej przez wielkość skrzynki:

Niestety ograniczenie to nie sprawdza się w wielu przypadkach – najlepiej sobie radzi gdy instancja zawiera głównie małe elementy. W przypadku większych elementów obliczony wynik może być znacznie niższy od rzeczywistego dolnego ograniczenia.

Drugie dolne ograniczenie, , odpowiada regule 8.20 z [MART 1990]. Zdecydowano się na nie zamiast reguły 8.19, ze względu na problematyczną kwestię doboru parametru oraz lepsze rezultaty. zdefiniowane jest jako:

gdzie oblicza się na podstawie wyznaczonych zbiorów , , :

jako:

Martello i Toth wykazali również, że wystarczy obliczać kolejno tylko dla unikalnych wartości elementów , posortowanych malejąco. Ponadto, często nie trzeba przeprowadzać obliczeń dla wszystkich wartości spełniających podaną nierówność.

Dolne ograniczenia i dla testowej instancji są sobie równe i wynoszą 4.

3.3. Algorytm dokładny

Wykorzystany algorytm dokładny opiera się na metodzie podziału i ograniczeń (ang. *Branch & Bound*) i stanowi zmodyfikowaną wersję algorytmu dokładnego opisanego w [MART 1990]. W pierwszym kroku elementy są sortowane wg malejących rozmiarów. Następnie tworzone jest drzewo rozwiązań. W każdym węźle nowe rozwiązania są generowane poprzez umieszczenie aktualnego elementu kolejno we wszystkich skrzynkach, w których jest wystarczająca ilość miejsca aby go pomieścić oraz w jednej, nowej skrzynce. Liście takiego drzewa stanowią dopuszczalne rozwiązania.

W celu ograniczenia liczby węzłów stosuje się tzw. odcięcia – polegają one na nie przeglądaniu gałęzi, w których na pewno nie zostaną znalezione rozwiązania lepsze od najlepszego znalezionego do tej pory rozwiązania. W omawianym algorytmie stosuje się 2 rodzaje odcięć. Pierwsze z nich pomija gałęzie umieszczające element w nowej skrzynce, dla których aktualna liczba skrzynek wynosi lub (w tym przypadku dodanie nowej skrzynki może doprowadzić do rozwiązania co najwyżej tak samo dobrego jak aktualne). W każdym węźle obliczana jest też wartość dolnych ograniczeń i . Jeżeli wartość któregoś z nich jest większa od najmniejszej obliczonej dotychczas, to dana gałąź nie jest przeglądana. 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 znalezionego rozwiązania dopuszczalnego odpowiada wartości dolnych ograniczeń, to obliczenia zostają zakończone.

Na rysunku 3.2. przedstawiono drzewo dla przykładowej instancji z poniższego rysunku.



Rys. 3.2. Przykładowa instancja dla algorytmu dokładnego



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

Dla instancji testowej algorytm daje (oczywiście) rozwiązanie optymalne, składające się z 4 skrzynek:



Rys. 3.4. Wynik działania algorytmu dokładnego dla instancji testowej

3.4. Algorytmy listowe

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 aktualnej skrzynce, dopóki pozwala na to ilość wolnego miejsca. W przypadku gdy w skrzynce nie ma już miejsca pozwalającego umieścić w niej aktualny element, dodawana jest nowa skrzynka i aktualny, oraz w miarę możliwości, kolejne elementy są umieszczane w nowej skrzynce. Cały proces jest powtarzany aż do wykorzystania wszystkich elementów.

Pseudokod:

1. *wybierz następny element*
2. *jeżeli w aktualnej skrzynce ilość wolnego miejsca jest większa lub równa od rozmiaru elementu, to umieść go w aktualnej skrzynce i przejdź do kroku 1.; w przeciwnym wypadku (brak miejsca lub brak skrzynek) przejdź do następnego kroku*
3. *dodaj nową skrzynkę i umieść w niej aktualny element. Wybierz dodaną skrzynkę jako aktualną i przejdź do kroku 1.*

Główne zalety tego algorytmu to prostota i szybkość działania – jego złożoność czasowa to . 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 jest nie wykorzystywanie wolnego miejsca w skrzynkach innych niż aktualna. Często prowadzi to do dokładania nowych skrzynek dla kolejnych elementów, podczas gdy w istniejących skrzynkach jest jeszcze dla nich miejsce. Przekłada się to oczywiście na słabe wyniki. Wartość dla tego algorytmu wynosi .

Dla instancji testowej *Next-Fit* daje wynik 6. Przedstawia to poniższy rysunek (liczby nad skrzynkami 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 otrzymania. Wybrany element jest umieszczany w pierwszej skrzynce, w której znajduje się wystarczająca ilość wolnego miejsca (skrzynki są numerowane w kolejności ich dodania). Nowa skrzynka jest dodawana tylko wtedy, gdy w żadnej istniejącej skrzynce nie ma miejsca dla aktualnego elementu.

Pseudokod:

1. *wybierz następny element*
2. *jeżeli brak skrzynek, to dodaj nową skrzynkę*
3. *wybierz następną skrzynkę (pierwszą, jeżeli żadna nie jest wybrana)*
4. *jeżeli element mieści się w aktualnej skrzynce, to umieść go w niej, wyczyść wybór aktualnej skrzynki i przejdź do kroku 1.; w przeciwnym wypadku przejdź do kroku 3.*

Algorytm eliminuje główną wadę swojego poprzednika – wykorzystuje wolną przestrzeń już istniejących skrzynek. Dzięki temu osiągane wyniki są znacznie lepsze: . Konieczność poszukiwania wolnego miejsca wśród istniejących skrzynek powoduje oczywiście zwiększenie złożoności. Przy zastosowaniu struktur tzw. 2-3 drzew (ang. *2-3 tree*) możliwe jest uzyskanie złożoności rzędu . W omawianym systemie zdecydowano się jednak na prostszą („klasyczną”) implementację algorytmu o złożoności , przeszukującą dodane skrzynki wg kolejności ich dodania (w czasie liniowym).

Wynik osiągany przez *First-Fit* dla instancji testowej to 5. 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 pierwszej skrzynce, w której element się mieści, umieszcza się go w skrzynce, w której pozostanie najmniej miejsca (po dodaniu elementu). Rozwiązanie to bazuje na założeniu, że należy minimalizować pozostałą w poszczególnych skrzynkach przestrzeń (wykorzystując je w maksymalnym stopniu) – w ten sposób minimalizuje się całkowitą wolną przestrzeń, a w konsekwencji zmniejsza liczbę skrzynek. Oczywiście istnieją przypadki, gdy takie podejście nie jest poprawne.

Pseudokod:

1. *wybierz następny element*
2. *jeżeli brak skrzynek, to przejdź do kroku 4.; w przeciwnym razie wybierz następną skrzynkę (pierwszą, jeżeli żadna nie jest wybrana) i przejdź do kroku 3.*
3. *oblicz ilość wolnego miejsca w skrzynce jeżeli umieszczono by w niej aktualny element. Jeżeli obliczone wolne miejsce jest mniejsze od tej samej wartości obliczonej dla najlepszej skrzynki lub żadna skrzynka nie jest oznaczona jako najlepsza, to oznacz aktualną skrzynkę jako najlepszą.*
4. *jeżeli żadna ze skrzynek nie została oznaczona jako najlepsza, to dodaj nową skrzynkę i oznacz ją jako najlepszą*
5. *wstaw element do najlepszej skrzynki, wyczyść wybór najlepszej skrzynki i przejdź do kroku 1.*

Wartość dla algorytmu *Best-Fit* jest taka sama jak dla *First-Fit* i wynosi . Złożoność czasowa również jest identyczna – przy zastosowaniu 2-3 drzew i  przy zastosowanym podejściu.

Rozwiązanie dla instancji testowej przedstawiono poniżej. Uzyskany wynik to 5.



Rys. 3.7. Wynik działania algorytmu Best-Fit

3.4.4. First-Fit Decreasing (ang. *First-Fit Decreasing*)

Wszystkie opisane do tej pory algorytmy posiadają wspólną cechę – elementy są umieszczane w skrzynkach wg ich kolejności „na wejściu”. Zaletą takiego podejścia jest brak konieczności znajomości wszystkich elementów *a priori*. Wadą natomiast duża zależność 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 tą 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.

Pseudokod:

1. *posortuj elementy wg malejących rozmiarów*
2. *rozwiąż uzyskaną instancję algorytmem First-Fit*

W związku z tym, że istnieją algorytmy sortowania o rzędzie złożoności takiej samej (lub mniejszej) niż złożoność algorytmu *First-Fit*, złożoność czasowa algorytmu *First-Fit Decreasing* jest taka sama – . 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. Stosunek wynosi

Dla instancji testowej *FFD* daje rozwiązanie optymalne, odpowiadające 4 skrzynkom:



Rys. 3.8. Wynik działania algorytmu First-Fit Decreasing

3.4.5. Best-Fit Decreasing (ang. *Best-Fit Decreasing*)

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

Pseudokod:

1. *posortuj elementy wg malejących rozmiarów*
2. *rozwiąż uzyskaną instancję algorytmem Best-Fit*

Złożoność czasowa jest identyczna jak w przypadku algorytmu *BF* (). Wartość wynosi .

Również w tym przypadku dla instancji testowej 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 wejścia w ściśle określonej kolejności, algorytm losowego dopasowania pobiera je w kolejności losowej. Kolejnym krokiem jest ponowne losowanie – tym razem losowana jest jedna ze skrzynek, w których element się zmieści (lista takich skrzynek jest uprzednio tworzona). Element zostaje umieszczony w wylosowanej skrzynce (lub w nowej – jeżeli w żadnej się nie mieści). Dzięki tworzeniu listy skrzynek, algorytm zawsze wykorzystuje miejsce, gdy jest ono dostępne.

Pseudokod:

1. *wybierz następny element poprzez losowanie*
2. *stwórz listę skrzynek, w których element się mieści*
3. *jeżeli lista skrzynek jest pusta, to dodaj nową skrzynkę, umieść w niej element i przejdź do kroku 1; w przeciwnym wypadku przejdź do kroku 4.*
4. *wybierz jedną ze skrzynek z listy poprzez losowanie, umieść w niej element i przejdź do kroku 1.*

Złożoność czasowa jest taka sama jak w algorytmie *Best-Fit* – w tym przypadku algorytm również przegląda wszystkie już dodane skrzynki.

Ze względu na element losowości może za każdym razem zwracać różne wyniki. Jedno z rozwiązań (5) uzyskanych dla instancji testowej 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 algorytmów. Opisywane do tej pory algorytmy pobierały z wejścia pojedynczo elementy, koncentrując się na ich optymalnym umieszczeniu w skrzynce. Jest to działanie zorientowane na element (ang. *item-oriented*).

Działanie algorytmu redukcji jest zorientowane na skrzynkę (ang. *bin-oriented*). Opiera się ono na procedurze redukcji, opisanej w [MART 1990]. Procedura ta poszukuje dopuszczalnego wypełnienia skrzynki, złożonego z co najwyżej 3 elementów, takiego, że dominuje ono wszystkie pozostałe (co najwyżej 3-elementowe) wypełnienia. Kryterium dominacji opisano w osobnym podrozdziale (3.6.1).

Algorytm redukcji znajduje takie wypełnienia dla wszystkich pozostałych elementów, a następnie dodaje je do aktualnego rozwiązania. Kolejny krok to usunięcie wykorzystanych elementów z tych, które pozostały. Dodatkowo usuwany jest także najmniejszy element i 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. 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.

Pseudokod:

1. *znajdź (rozłączne) wypełnienia skrzynki, złożone z pozostałych elementów, które dominują inne wypełnienia*
2. *zastosuj znalezione wypełnienia, dodając w ten sposób wypełnione skrzynki do rozwiązania; wykorzystane elementy usuń z pozostałych elementów*
3. *z pozostałych elementów usuń najmniejszy element (dodaj go do listy L)*
4. *jeżeli lista pozostałych elementów jest niepusta, to powróć do kroku 1; w przeciwnym razie przejdź do następnego kroku*
5. *dopóki to możliwe, umieszczaj kolejne elementy listy w wolnych miejscach w aktualnym rozwiązaniu*
6. *pozostałe elementy zapakuj do nowych skrzynek stosując algorytm Next-Fit*

Złożoność czasowa zaimplementowanego rozwiązania to . Możliwe jest jednak uzyskanie złożoności kwadratowej.

Dla instancji testowej algorytm znajduje rozwiązanie optymalne, składające się z 4 skrzynek.



Rys. 3.11. Wynik działania algorytmu redukcji

3.5.1. Kryterium dominacji

Znajdowanie optymalnych wypełnień skrzynek wymaga określenia relacji pozwalającej porównywać ze sobą 2 dane wypełnienia, tj. określać, które z nich jest lepsze. Relację tą nazywamy dominacją i zgodnie z [FUKA 2007] definiujemy jako:

**Definicja (dominacja):** *Dla dwóch danych dopuszczalnych (poprawnych) sposobów wypełnienia skrzynek oraz mówimy, że* ***dominuje*** *jeżeli wartość optymalnego rozwiązania które można uzyskać wykorzystując wypełnienie skrzynki w sposób jest* ***nie gorsza*** *od wartości optymalnego rozwiązania, które można uzyskać stosując wypełnienie dla tej samej skrzynki.*

W swojej pracy Fukunaga i Korf opisują kilka kryteriów dominacji, m.in. zaproponowane przez Martello i Totha (i stosowane w procedurze redukcji):

**Kryterium dominacji (M&T):** *Niech i będą poprawnymi wypełnieniami skrzynek.  dominuje jeżeli można podzielić na podzbiorów takich, że każdemu podzbiorowi przyporządkowany jest element taki, że suma wag (rozmiarów) elementów zbioru jest mniejsza lub równa niż waga (rozmiar) elementu .*

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

Przykład:

dominuje , ponieważ można podzielić na podzbiory , oraz , z których każdy można by zmieścić w skrzynce odpowiadającej wielkości pojedynczego elementu . W tym wypadku wymienione zbiory można przyporządkować kolejno do elementów , i . Oczywiście istnieją również inne przyporządkowania.

3.6. Asymptotyczny schemat aproksymacyjny

Kolejnym z zastosowanych algorytmów jest asymptotyczny schemat aproksymacyjny (ang. *asymptotic* *approximation scheme*, *AAS*), zaproponowany w [FERN 1981] oraz opisany m.in. w [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 skrzynek. Trzecia składa się z elementów najmniejszych. Druga, złożona z elementów o średniej wielkości jest dzielona na równoliczne zbiory elementów. Elementy poszczególnych zbiorów zostają zaokrąglone do rozmiaru największego elementu zbioru. Dzięki temu liczba różnych elementów zostaje znacznie zmniejszona. Elementy te są pakowane poprzez rozwiązanie zadania programowania liniowego, które jest następnie transformowane do rozwiązania rzeczywistego. Ostatnim krokiem jest wypełnienie w miarę możliwości pozostałych miejsc elementami z trzeciej grupy. Te, których nie uda się pomieścić są pakowane algorytmem *Next-Fit*.

Pseudokod:

*1. dla podanego oblicz wartości określające rozmiary elementów poszczególnych grup.*

*2. na podstawie obliczonych wartości dokonaj podziału elementów na 3 listy: , i .*

*3. zapakuj elementy listy stosując skrzynek (każdy element zostaje umieszczony w osobnej skrzynce).*

*4. na podstawie listy stwórz nową listę o mniejszej liczbie różnych rozmiarów elementów. Znajdź zapakowanie elementów tej listy, stosując programowanie liniowe. Przekształć uzyskane zapakowanie w zapakowanie listy .*

*5. dopóki to możliwe, umieszczaj kolejne elementy listy w wolnych miejscach w aktualnym rozwiązaniu.*

*6. pozostałe elementy zapakuj do nowych skrzynek stosując algorytm Next-Fit.*

Dodatkowe wyjaśnienie kroków pierwszego, drugiego i czwartego umieszczono w osobnych podrozdziałach 3.6.1. oraz 3.6.2.

Dla dowolnego złożoność czasowa to . Metoda gwarantuje uzyskanie rozwiązania o liczbie pudełek . Poniżej przedstawiono rezultaty uzyskane dla instancji testowej dla różnych wartości parametru .



Rys. 3.12. Wynik działania asymptotycznego schematu aproksymacji dla



Rys. 3.13. Wynik działania asymptotycznego schematu aproksymacji dla



Rys. 3.14. Wynik działania asymptotycznego schematu aproksymacji dla

3.6.1. Podział elementów na listy

Podział listy elementów jest dokonywany na podstawie wartości obliczonych w korku pierwszym. Są one następujące: , . Do pierwszej listy  trafiają elementy mniejsze od . Następnie obliczana jest liczba podzbiorów (list) listy : . Kolejny krok, to obliczenie wartości jako -ty najmniejszy element . Na podstawie uzyskanych wartości tworzy się listy i :

3.6.2. Wykorzystanie programowania liniowego

Pierwszym krokiem jest uzyskanie stworzenie na podstawie listy nowej listy elementów (). Lista składa się z elementów , z których każdy występuje razy. Następnie tworzone są wszystkie możliwe sposoby zapakowania skrzynki za pomocą elementów nowej listy. Na tej podstawie tworzone są ograniczenia i funkcja celu dla *solvera* PL – minimalizacja liczby wykorzystanych sposobów zapakowania.

Otrzymany wynik może zawierać wartości nie będące liczbami całkowitymi (np. wykorzystaj pakowanie (1 raz drugi element oraz 2 piąte elementy) 1,5 razy. Z tego względu uzyskane wartości są zaokrąglane w dół (funkcja podłoga). Wykorzystujemy uzyskane w ten sposób pakowania a pozostałe elementy są pakowane algorytmem *Next-Fit*.

Przekształcenie uzyskanego w ten sposób rozwiązania (zapakowanie ) przekształca się w zapakowanie poprzez zastąpienie wartości poszczególnych elementów o rozmiarach oryginalnymi elementami z listy .

3.7. Własny algorytm

Poza implementacją znanych rozwiązań, zdecydowano się również na stworzenie własnego rozwiązania. Opracowany algorytm jest połączeniem dwóch wcześniej opisywanych rozwiązań – algorytmu *Next-Fit* oraz dokładnego. Nazwano go *PBI* (ang. *Probably Best Improvement*) – przypuszczalnie najlepsza poprawa. W pierwszym etapie elementy są pakowane algorytmem następnego dopasowania. Następnie podejmowane są próby poprawienia uzyskanego rozwiązania. Procedura poprawy jest wywoływana maksymalnie razy, gdzie oznacza liczbę skrzynek 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 skrzynek więcej niż dwukrotnie. Poza tym jeżeli w 3 kolejnych próbach nie uzyskano poprawy, algorytm jest przerywany.

Skrzynki, których wypełnienie próbuje się poprawić to:

* 3 skrzynki z największą ilością wolnego miejsca – teoretycznie ich zawartość najłatwiej będzie umieścić w innych skrzynkach,
* 3 skrzynki z największą liczbą elementów – największa liczba elementów oznacza, w przeciętnym przypadku, najmniejsze elementy – dają one najwięcej możliwości przenoszenia elementów poprzez nieznaczne zwiększenie wolnej przestrzeni
* 3 losowe skrzynki

Elementy z tak wybranych 9 skrzynek próbuje się zapakować algorytmem dokładnym.

Złożoność algorytmu to - w najgorszym przypadku rozwiązanie uzyskane przez *Next-Fit* będzie odpowiadało liczbie elementów, czyli . W takim wypadku liczba prób poprawienia wyniku może wynieść , co daje złożoność a więc kwadratową. Czas pojedynczej poprawy jest niezależny od rozmiaru instancji i można go uznać za wartość stałą. W stosunku do algorytmu *BFD* czasowo *PBI* wypada gorzej dla mniejszych instancji, gdzie poszukiwanie dokładnych rozwiązań ma znaczący wpływ na czas działania algorytmu. Wraz ze wzrostem liczby elementów algorytm zyskuje i jego czas działania jest krótszy od *BFD*.

Przeprowadzone testy wykazały, że błąd uzyskiwany przez *PBI* jest ok. 3 lub więcej razy mniejszy od błędu algorytmu *Next-Fit*. Jakość rozwiązań pogarsza się jednak wraz ze wzrostem liczby elementów – jest to spowodowane tym, że mniejsza część rozwiązania jest pakowana optymalnie.

Ze względu na sposób działania dla instancji testowej po pierwszym kroku do poprawy wybrane zostaną wszystkie skrzynki. Działanie algorytmu będzie więc równoważne z uruchomieniem algorytmu dokładnego dla tej instancji. Otrzymane rozwiązanie jest więc optymalne.

4. IMPLEMENTACJA

4.1. Opis systemu

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 problemem pakowania. Główne elementy biblioteki to:

* klasy bazowe (ang. *base*) – zawiera podstawowe klasy, reprezentujące skrzynkę, instancję problemu oraz algorytm i algorytm listowy
* dolne ograniczenia – obliczanie dolnych ograniczeń dla zadanej instancji
* algorytmy – implementacje poszczególnych algorytmów
* eksperyment – klasy odpowiedzialne za przeprowadzanie eksperymentu oraz klasy reprezentujące m.in. aktualny stan eksperymenty, jego parametry wejściowe, oraz wyjściowe (próbki danych), itp.
* generator danych – generowanie losowych elementów, spełniających wymagania co do zakresu rozmiarów i rozkładu

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

* moduł wizualizacji – składający się z widoku w głównym oknie programu oraz okien prezentujących algorytmy
* moduł eksperymentu – widok w oknie głównym oraz okna wyświetlające postęp eksperymentu i jego wynik
* kontrolki – własne kontrolki, reprezentujące pojedynczy element, skrzynkę oraz wykres przedstawiający wyniki eksperymentu; korzystają z nich 2 poprzednie moduły
* odczyt pliku – ten element odpowiada za okno wyboru typu otwieranego pliku

Całość przedstawiono na poniższym rysunku:



Rys. 4.1. Architektura systemu

4.1.2. Wykorzystane technologie

Aplikacja została napisana w całości w języku programowania C#, z wykorzystaniem platformy .NET w wersji 3.5. Interfejs oparto na technologii WPF (ang. *Windows Presentation Foundation*). Wyświetlanie elementów i skrzynek zrealizowano za pomocą własnych kontrolek.

W celu wyświetlania wykresów, przedstawiających wyniki eksperymentu obliczeniowego, zdecydowano się na napisanie własnej kontrolki. Główne przyczyny podjętej decyzji to:

* niewielka liczba skończonych (lub nadal rozwijanych), darmowych rozwiązań, oferujących potrzebną funkcjonalność
* konieczność nauki wykorzystania wybranego rozwiązania, przy częstym braku przykładów i ubogiej dokumentacji technicznej.

4.1.3. Wymagania funkcjonalne i pozafunkcjonalne

Wymagania funkcjonalne:

* implementacja dolnych ograniczeń
* implementacja algorytmów – w szczególności najpopularniejszych heurystyk listowych oraz algorytmu dokładnego
* wizualizacja rozwiązań uzyskiwanych przez poszczególne algorytmy
* prezentacja obliczeń (działania) algorytmów – dotyczy heurystyk listowych
* odczyt/zapis instancji problemu z/do pliku
* moduł eksperymentu obliczeniowego – umożliwiający wykonywanie obliczeń dla dużych zbiorów danych i porównanie algorytmów
* prezentacja wyników eksperymentu w postaci wykresu – wraz z możliwością wyboru wyświetlanych parametrów i dodatkowych funkcji
* zapis wyników eksperymentu do pliku
* odczyt/zapis ustawień generatora danych dla eksperymentu.

Wymagania pozafunkcjonalne:

* intuicyjność, prostota – program powinien być przede wszystkim prosty w obsłudze. Wprowadzanie przykładowych instancji problemu powinno być proste i szybkie. Po wprowadzeniu danych użytkownik powinien mieć możliwość „natychmiastowego” wyświetlenia wyniku działania wybranego algorytmu.

4.1.4. 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 3.0.

4.2. Dokumentacja użytkownika

4.2.1. Główne okno aplikacji

Po uruchomieniu aplikacji użytkownikowi 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. 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, natomiast ostatnia – wyświetla podstawowe informacje o autorze programu.

4.2.2. Moduł wizualizacji

Moduł wizualizacji umożliwia prezentację działania algorytmów krok po kroku (tylko algorytmy listowe) oraz sprawdzenie wyniku działania (wszystkie algorytmy) dla pojedynczych instancji problemu. Na rysunku 4.3. przedstawiono widok modułu wizualizacji.



Rys. 4.3. Widok modułu wizualizacji

4.2.2.1. Wprowadzanie danych

Pierwszym krokiem jest wprowadzenie danych. Dane do prezentacji można wprowadzić na 3 sposoby:

* załadowanie z pliku
* automatyczne wygenerowanie
* ręczne wprowadzenie elementów.

Aby załadować instancję 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. (poszczególne typy opisano w rozdziale 5.) oraz podać dodatkowe informacje (dla pliku z pojedynczą instancją – poz. 3.). 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

Drugi sposób to automatyczne wygenerowanie danych. Parametry generowanych danych należy wprowadzić 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. 4.5. umożliwiają uzyskanie konkretnej kolejności elementów (od lewej): losowej, rosnącej, malejącej.

Ostatni, trzeci sposób 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 pojedynczych elementów, itp.

Niezależnie od wybranej metody, każda zmiana elementów spowoduje wyświetlenie podglądu uzyskanej w ten sposób instancji rys. 4.3. poz. 6., wraz z wartościami obliczonych dolnych ograniczeń i . 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 należy wybrać algorytmy, które zostaną zaprezentowane. Prezentacja działania jest możliwa tylko dla algorytmów z górnego rzędu. Możliwy jest wybór kilku algorytmów – dla każdego z nich zostanie utworzone osobne okno. Aby rozpocząć prezentację (lub wyświetlić wynik) należy kliknąć odpowiednio przycisk „Wizualizacja” lub „Wynik”.

4.2.2.2. Prezentacja działania algorytmu

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



Rys. 4.6. Okno prezentacji działania algorytmu

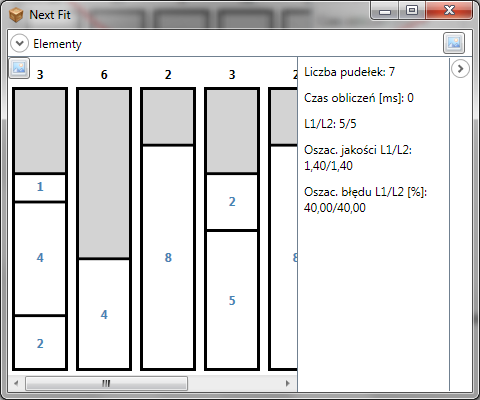
W górnej części (rys. 4.6. poz. 1.) wyświetlany jest podgląd wszystkich elementów wraz z ich rozmiarami. Elementy, które już zostały wykorzystane są wyblakłe, natomiast aktualnie wybrany element pulsuje. Za pomocą przycisków na górnej belce możliwe jest zwinięcie podglądu lub zapisanie go do pliku graficznego.

W środkowej części umieszczono podgląd aktualnego stanu. Aktualnie wybrana skrzynka pulsuje. Przycisk w lewym górnym rogu umożliwia zapis 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.

4.2.2.3. Wyświetlanie wyniku działania algorytmu



Rys. 4.7. Okno wyświetlające wynik działania algorytmu

Okno wyświetlania wyniku działania algorytmu w stosunku do okna prezentacji (po zakończeniu algorytmu) różni się tylko dodatkową informacją nt. czasu działania algorytmu. Przedstawiono je na powyższym rysunku.

4.2.3. Moduł eksperymentu obliczeniowego

Moduł eksperymentu obliczeniowego służy do porównywania algorytmów (lub badania pojedynczego algorytmu) dla zbiorów danych, spełniających określone parametry. Możliwe jest też sprawdzenie działania algorytmów dla pojedynczej instancji załadowanej z pliku. Widok modułu eksperymentu obliczeniowego przedstawiono na poniższym rysunku:

****

Rys. 4.8. Widok modułu eksperymentu obliczeniowego

4.2.3.1. Wprowadzanie parametrów eksperymentu

W przypadku testowania pojedynczej instancji załadowanej z pliku należy kliknąć przycisk rys. 4.8. poz. 1. i załadować plik w sposób analogiczny do ładowania instancji do prezentacji (patrz: punkt 4.2.2.1.). Następnie należy zaznaczyć opcję rys. 4.8. poz. 2..

Możliwe jest też generowanie danych testowych „na bieżąco”, w trakcie trwania eksperymentu. Parametry generatora należy podać w górnej części okna (rys. 4.8. poz. 3.). Pierwsza kolumna odpowiada za rozmiary generowanych instancji i liczbę powtórzeń obliczeń. Druga kolumna ustawień generatora odpowiada za rozmiar skrzynki i elementów. Będą one miały rozmiary z przedziału -, gdzie odpowiada rozmiarowi skrzynki pomnożonemu przez wartość wpisaną w polu 4., a rozmiarowi skrzynki pomnożonemu przez wartość wpisaną w polu 5.. Najmniejsza wartość, jaką można wprowadzić do pola minimalnej wartości to , natomiast największa wartość pola maksymalnej wartości to . Ostatnia kolumna odpowiada za sposób losowania elementów (różne rozkłady prawdopodobieństwa). Możliwe jest wybranie kilku rozkładów.

Niezależnie od tego źródła danych, przed rozpoczęciem eksperymentu należy jeszcze dokonać wyboru testowanych algorytmów za pomocą pól wyboru (rys. 4.8. poz. 6.), oraz tego, czy elementy mają być sortowane (możliwy jest wybór kilku opcji jednocześnie). Kliknięcie przycisku „Eksperyment” spowoduje wyświetlenie okna postępu przebiegu eksperymentu, które opisano w kolejnym punkcie.

4.2.3.3. Śledzenie przebiegu eksperymentu

Po uruchomieniu eksperymentu program wyświetli okno postępu przebiegu eksperymentu. Przedstawiono je na rysunku 4.9.. Wyświetla ono informacje o aktualnie uruchomionym algorytmie i danych, na których działa (rys. 4.9. poz. 1.) oraz pasek, przedstawiający całkowity postęp przebiegu eksperymentu. W celu rozpoczęcia eksperymentu należy kliknąć przycisk „Start”. Kliknięcie przycisku „Przerwij” w trakcie eksperymentu spowoduje jego przerwanie i powrót do głównego okna programu.



Rys. 4.9. Okno przebiegu eksperymentu

4.2.3.4. Wyświetlanie wyników działania eksperymentu

Po zakończeniu eksperymentu wyświetlane jest okno prezentujące wyniki:



Rys. 4.10. Okno prezentujące wynik eksperymentu

Jest ono podzielone na 4 części. W centralnej części okna (rys. 4.10. 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. 4.10. poz. 2.). Pierwsza lista (poz. 3.) pozwala na wybór wyników pogrupowanych wg algorytmów, rozkładów danych czy sortowania danych. Możliwy jest też wybór grupowania wg par: algorytm algorytm/rozkład, algorytm/sortowanie, rozkład/sortowanie. Umożliwia to zbadanie wpływy danych wejściowych na wyniki uzyskiwane przez zbiór algorytmów, itp. Druga lista wyboru (poz. 4.) umożliwia wybór konkretnego parametru oceny:

* czas działania
* wynik
* oszacowanie jakości – obliczane jako:
* oszacowanie błędu – obliczane jako:

Za pomocą przycisku (rys. 4.10. poz. 5.) możliwa jest zmiana typu wykresu na słupkowy, liniowy lub punktowy. Przycisk (poz. 6.) służy do zmiany skali (oś ) 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. 4.10. 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. Ustawienia

W oknie ustawień (rys. 4.11.) możliwe jest wybranie kilku podstawowych opcji:

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

Okno z ustawieniami przedstawia poniższy rysunek 4.11. (na następnej stronie).



Rys. 4.11. Główne okno programu – ustawienia

4.3. Dokumentacja techniczna

W tym podrozdziale skupiono się na opisie struktury klas głównych elementów biblioteki. 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, skrzynka, element

Do opisu problemu pakowania potrzebne są 3 główne obiekty: element, skrzynka oraz instancja (reprezentująca dany stan lub np. końcowy wynik). W stworzonym systemie każdy element jest reprezentowany jako pojedyncza (dodatnia) liczba całkowita. Skrzynka i instancja są natomiast reprezentowane przez osobne klasy.

Klasa *Bin*, reprezentująca skrzynkę zawiera listę elementów, które w niej umieszczono oraz rozmiar skrzynki (właściwość *Size*). Utworzono również 2 metody pomocnicze, dodającą element do skrzynki – *Insert()* i obliczającą pozostałe miejsce – *FreeSpace()*.

Klasa reprezentująca instancję problemu pakowania, *Instance*, zawiera listę skrzynek, elementów oraz nazwę instancji i rozmiar skrzynki. Przedstawiono to na poniższym rysunku:



Rys. 4.12. 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. Zdecydowano się na rozwiązanie wykorzystujące po jednej implementacji każdego algorytmu (zamiast tworzenia osobnych wersji kodu dla prezentacji i samych obliczeń). 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 wiadomość (informację) oraz za element i skrzynkę, które należy zaznaczyć jako aktualnie wybrane. Rys. 4.13. przedstawia klasy *BaseAlgorithm* oraz *ListAlgorithm*.



Rys. 4.13. 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.4.14.Klasy generujące dane

4.3.4. Struktura klas – eksperyment

Największa ilość 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 informacje o aktualnie uruchomionym algorytmie, sortowaniu, licznie 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.



Rys. 4.15. Struktura klas wykorzystywanych w eksperymencie

4.3.5. Obsługiwane formaty plików

*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 linijki mogą (ale nie muszą zawierać wielkość skrzynki i/lub liczbę elementów). W przypadku nie podania wielkości skrzynki 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\_skrzynki]

[liczba\_elementów]

element\_1

element\_2

...

element\_n

**Plik z wieloma instancjami**

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

Pierwsza linijka opisu instancji zawiera jej nazwę. W drugiej znajdują się wartości (rozdzielone dowolną liczbą spacji lub tabulatorów) określające: wielkości skrzynki, liczbę elementów, wartość najlepszego znanego rozwiązania (ta wartość jest ignorowana). Kolejne linie zawierają rozmiary elementów (1/linijka). Przedstawiono to poniżej (wcięcia dodano w celu zwiększenia czytelności):

liczba\_instancji

nazwa\_instancji\_1

wielkość\_skrzynki\_1 liczba\_elementów\_1 [najlepsze\_znane\_rozw\_1]

element\_1\_1

element\_1\_2

...

element\_1\_m

nazwa\_instancji\_2

wielkość\_skrzynki\_2 liczba\_elementów\_2 [najlepsze\_znane\_rozw\_2]

element\_2\_1

element\_2\_2

...

element\_2\_m

...

nazwa\_instancji\_n

wielkość\_skrzynki\_n liczba\_elementów\_n [najlepsze\_znane\_rozw\_n]

element\_n\_1

element\_n\_2

...

element\_n\_m

**Plik z wieloma instancjami z wagami 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 wag (lub liczbę par *rozmiar elementu (waga)-liczba elementów*). Następne linie zawierają po 2 wartości, oddzielone dowolną liczbą spacji bądź tabulatorów, określające kolejno: rozmiar (wagę) elementu oraz liczbę elementów o podanym rozmiarze. Schemat pliku znajduje się poniżej:

nazwa\_instancji\_1

liczba\_wag\_1

wielkość\_skrzynki\_1

waga\_elem\_1\_1 liczba\_elem\_1\_1

waga\_elem\_1\_2 liczba\_elem\_1\_2

...

waga\_elem\_1\_m liczba\_elem\_1\_m

nazwa\_instancji\_2

liczba\_wag\_2

wielkość\_skrzynki\_2

waga\_elem\_2\_1 liczba\_elem\_2\_1

waga\_elem\_2\_2 liczba\_elem\_2\_2

...

waga\_elem\_2\_m liczba\_elem\_2\_m

...

nazwa\_instancji\_n

liczba\_wag\_n

wielkość\_skrzynki\_n

waga\_elem\_n\_1 liczba\_elem\_n\_1

waga\_elem\_n\_2 liczba\_elem\_n\_2

...

waga\_elem\_n\_m liczba\_elem\_n\_m

**Format zapisywanych plików**

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

5. EKSPERYMENT OBLICZENIOWY

Eksperyment obliczeniowy został podzielony na 4 główne części. W pierwszej testowane były algorytmy listowe. W osobnym podrozdziale pokazano również wpływ samych danych na wyniki. Druga część skupia się na asymptotycznym schemacie aproksymacyjnym – czasie działania i jakości uzyskiwanych rozwiązań, w zależności od parametru . 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*. Na końcu poświęcono kilka słów algorytmowi dokładnemu.

5.1. Heurystyki listowe

Algorytmy listowe testowano dla następujących parametrów:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **od-do (krok)** | **rozkłady danych** | **sortowanie elementów** | **rozmiar skrzynki** | **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 złożoności . 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 skrzynek) uzyskane przez poszczególne algorytmy. Już na pierwszy rzut oka widać, że najsłabiej wypada algorytm *następnego dopasowania*, 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 jak i bez. Dobre wyniki uzyskuje również algorytm *losowego dopasowania* – wynika to z zastosowanej strategii, dodającej nowe skrzynki tylko wtedy, gdy jest to konieczne.

Drugi z wykresów (rys. 5.4.) prezentuje oszacowanie jakości – im jego wartość jest bliższa , tym lepiej. Dodatkowo wyświetlone zostały linie oszacowania asymptotycznego: , , . 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 rozwiązania o rozmiarach 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 skrzynek nie jest zauważalna.



Rys. 5.3. Wyniki algorytmów listowych



Rys. 5.4.Oszacowanie jakości algorytmów listowych



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

5.2.1. Wpływ danych na wyniki

W tym podrozdziale skupiono się na wpływie danych wejściowych na uzyskiwane wyniki. Wykorzystane zostały wyniki tego samego eksperymentu – nie podawano ręcznie zakresu rozmiarów elementów w generatorze. Zamiast tego skupiono się na rozkładzie elementów oraz sposobie ich sortowania.



Rys. 5.6.Oszacowanie jakości w zależności od sortowania elementów



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). Nietrudno zauważyć, że w przypadku posortowania elementów rosnąco wyniki mogą być znacznie gorsze niż w przypadku sortowania odwrotnego. Okazuje się też, że badane heurystyki całkiem dobrze radzą sobie z danymi nie posortowanymi w ogóle.

Jak się okazuje, istotnym czynnikiem wpływającym na jakość uzyskanego rozwiązania jest rozkład losowy danych. 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. Głównym celem było zbadanie wpływu występującego w tej metodzie parametru . Stworzony system nie pozwala na przeprowadzenie eksperymentu dla różnych jednocześnie. Zamiast tego obliczenia przeprowadzono osobno dla każdej badanej wartości parametru a następnie eksportowano wyniki do programu Excel w celu dalszej obróbki. Z tego powodu wykresy pochodzą z tej aplikacji a nie opisywanej w tej pracy.

Warto również dodać, że dzięki kilkukrotnemu powtórzeniu eksperymentu oraz rozmiarom instancji działanie na różnych danych dla różnych nie ma większego wpływu na wyniki.

W tabeli 5.10. zebrano parametry tego eksperymentu. Pierwotnie planowano również test dla - powodowało to jednak znaczny wzrost liczby możliwych sposobów wypełnienia skrzynki, co skutkowało kończeniem się wolnej pamięci operacyjnej.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **od-do (krok)** | **rozkłady danych** | **sortowanie elementów** | **rozmiar skrzynki** | **rozmiar elementów** | **powtórzenia** |  |
| 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 (w %)

Powyższy rysunek przedstawia oszacowanie błędu dla różnych wartości - 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 z do spowodowało „skok” czasu obliczeń:

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

5.3. NF, BFD i inne

Ze względu na długi czas działania jednego z algorytmów, w ostatnim eksperymencie generowane instancje testowe były mniejsze. Bazując na poprzednim eksperymencie dla schematu aproksymacyjnego przyjęto .

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

Tab. 5.13. Parametry eksperymentu testującego inne algorytmy

Wspomnianym algorytmem był algorytm redukcji – czas jego działania wzrastał bardzo szybko w stosunku do liczby elementów, co pokazuje wykres na rys. 5.14.. 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 w bardzo dużych instancjach, dla których obliczenia związane ze asymptotycznym schematem aproksymacyjnym będą trwały zbyt długo lub wymagały zbyt dużej ilości pamięci.

5.4. Algorytm dokładny

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).

6. PODSUMOWANIE

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

Udostępniono również możliwość prostego i szybkiego generowania instancji problemu o zadanych parametrach. Elementy mogą być losowane z różnymi rozkładami prawdopodobieństwa. 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 działania najbardziej znanych algorytmów.

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 na wyniki. 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 – dzięki temu przy następnym uruchomieniu programu umożliwia on szybkie wznowienie pracy. Dodatkowo wiele elementów interfejsu można zwinąć, dzięki czemu nie zajmują niepotrzebnie miejsca.

Pomimo tego, wiele elementów systemu można by dopracować lub rozbudować. Dotyczy to np. algorytmu dokładnego, w którym nie zastosowano niektórych odcięć (kryterium dominacji). W trybie prezentacji poszczególne elementy mogły by być pokolorowane. Przydatne byłoby również ładowanie z pliku wielu instancji jednocześnie oraz zwiększenie możliwości wykresów – np. skalowanie, wyświetlanie większych zbiorów danych z możliwością przewijania, itp. Istnieje wiele możliwości rozwoju produktu.

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* oraz przedstawiony schemat aproksymacyjny. Złożoność dwóch pierwszych algorytmów (kwadratowa w najpopularniejszej implementacji) pozwala na stosowanie ich do nawet naprawdę dużych problemów. Poza tym są one bardzo proste. Trzeci algorytm radzi sobie jeszcze lepiej i działa bardzo szybko. To wszystko sprawia, że można z powodzeniem stosować je w wielu rzeczywistych problemach.

7. LITERATURA

|  |  |
| --- | --- |
| [BŁAŻ 1988] | Błażewicz J., Złożoność obliczeniowa problemów kombinatorycznych.  WNT, Warszawa, 1988 |
| [COFF 2006] | Coffmann E.G., Lueker G.S., Approximation algorithms for extensible bin packing, w: Journal of Scheduling 9, 2006, s. 63-69  Springer Science + Business Media, Inc., 2006 |
| [EILO 1971] | Eilon S., Christofides N., The loading problem, w: Management Scienge 17, 1971, s. 259-267 |
| [FERN 1981] | Fernandez de la Vega W., Lueker G.S., Bin packing can be solved within 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, w: Journal of Artificial Intelligence Research 28, 2007, s. 393-429 |
| [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 |
| [STER] | Sterna M., Prezentacja: Problem pakowania (bin-packing problem) |
| [WIK1] | Bin packing problem, w: Wikipedia, the free encyclopedia  http://en.wikipedia.org/wiki/Bin\_packing\_problem |
| [WWW1] | http://www.astrokettle.com |

8. ZAŁĄCZNIKI

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