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](#_Toc302303813)

[SPIS TREŚCI 5](#_Toc302303814)

[1. WSTĘP 7](#_Toc302303815)

[1.1. Cel i zakres pracy 7](#_Toc302303816)

[2. PROBLEM PAKOWANIA 8](#_Toc302303817)

[2.1. Sformułowanie problemu 8](#_Toc302303818)

[2.2. Dowód NP-zupełności 8](#_Toc302303819)

[2.3. Przegląd literatury 8](#_Toc302303820)

[3. ALGORYTMY 10](#_Toc302303821)

[3.1. Wstęp 10](#_Toc302303822)

[3.2. Dolne ograniczenia 10](#_Toc302303823)

[3.3. Algorytm pełnego przeglądu 11](#_Toc302303824)

[3.4. Algorytmy listowe 11](#_Toc302303825)

[3.4.1. Algorytm następnego dopasowania (ang. *Next-Fit*) 12](#_Toc302303826)

[3.4.2. Algorytm pierwszego dopasowania (ang. *First-Fit*) 13](#_Toc302303827)

[3.4.3. Algorytm najlepszego dopasowania (ang. *Best-Fit*) 14](#_Toc302303828)

[3.4.4. First-Fit Decreasing (ang. *First-Fit Decreasing*) 15](#_Toc302303829)

[3.4.5. Best-Fit Decreasing (ang. *Best-Fit Decreasing*) 16](#_Toc302303830)

[3.4.6. Algorytm losowego dopasowania/losowy 17](#_Toc302303831)

[3.5. Asymptotyczny schemat aproksymacyjny 17](#_Toc302303832)

[3.6. Algorytm redukcji 17](#_Toc302303833)

[3.7. Algorytm meta-heurystyczny 18](#_Toc302303834)

[4. IMPLEMENTACJA 19](#_Toc302303835)

[4.1. Opis systemu 19](#_Toc302303836)

[4.1.1. Architektura 19](#_Toc302303837)

[4.1.2. Wykorzystane technologie 19](#_Toc302303838)

[4.1.3. Wymagania funkcjonalne i pozafunkcjonalne 19](#_Toc302303839)

[4.2. Dokumentacja użytkownika 19](#_Toc302303840)

[4.2.1. Główne okno aplikacji 20](#_Toc302303841)

[4.2.2. Moduł wizualizacji 20](#_Toc302303842)

[4.2.3. Moduł eksperymentu obliczeniowego 20](#_Toc302303843)

[4.2.4. Ustawienia 21](#_Toc302303844)

[4.3. Dokumentacja techniczna 21](#_Toc302303845)

[5. EKSPERYMENT OBLICZENIOWY 22](#_Toc302303846)

[6. PODSUMOWANIE 23](#_Toc302303847)

[7. LITERATURA 24](#_Toc302303848)

[8. ZAŁĄCZNIKI 25](#_Toc302303849)

1. WSTĘP

Wprowadzenie do pracy: można wspomnieć o motywacjach projektu lub o znaczeniu samego problemu. Ok 1.5 strony – zawartość zależy od koncepcji autora. Należy pamiętać, że jest to pierwsza część pracy którą czyta recenzent.

1.1. Cel i zakres pracy

Wstępna specyfikacja wymagań – zawartość projektu. Podanie co znajduje się w kolejnych rozdziałach.

2. PROBLEM PAKOWANIA

2.1. Sformułowanie problemu

Formalna definicja problemu. Można wspomnieć o wariantach, związku z rozkrojem. Przykładowe zastosowania. Można załączyć przykład.

2.2. Dowód NP-zupełności

Określenie złożoności – dowód w oparciu o literaturę. Należy wspomnieć konsekwencje – jakie metody rozwiązywania mogą być stosowane (jest to wprowadzenie do rozdziału o algorytmach).

Problem pakowania jest silnie NP-zupełny.

DOWÓD:

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

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.

konsekwencje

2.3. Przegląd literatury

Krótki przegląd literatury w oparciu o dostarczone materiały i znajdujące się w nich referencje. Wyszczególnienie stosowanych podejść. Można poszukać w sieci artykułów.

3. ALGORYTMY

Prezentacja algorytmów – ogólny opis, idea, zasada działania w postaci pseudo-kodu, najlepiej ilustrowana np. uzyskanym rozwiązaniem dla przykładowej instancji. Jedna instancja we wszystkich algorytmach pozwoli na porównanie. W ramach opisu należy podać złożoność czasową i oszacowanie jakości.

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 [LIT]. 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.



3.2. Dolne ograniczenia

W omawianym systemie zaimplementowano 2 dolne ograniczenia, przedstawione w [LIT]. 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 [LIT]. 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 [LIT] 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 pełnego przeglądu

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):



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 i umieść w niej element; przejdź do kroku 1*
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***dokończyć**

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:



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:

**dokończyć**

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.



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:



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 *BF*. Wartość wynosi .

Również w tym przypadku dla instancji testowej otrzymamy rozwiązanie optymalne (4):



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

Opis

3.5. Asymptotyczny schemat aproksymacyjny

Opis

3.6. Algorytm redukcji

Kolejnym z wykorzystanych algorytmów jest algorytm redukcji. Opiera się on na procedurze redukcji, opisanej w [LIT]. Procedura ta poszukuje dopuszczalnego wypełnienia skrzynki, złożonego z co najwyżej 3 elementów, takiego, że dominuje ono wszystkie pozostałe (co najwyżej 3-elementowe) wypełnienia.

**opisać kryterium dominacji**

Algorytm znajduje takie wypełnienia dla pozostałych elementów, a następnie dodaje je do rozwiązania. Kolejny krok to usunięcie wykorzystanych elementów z pozostałych elementów. 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 rozwiązaniu elementami, które były odrzucane jako najmniejsze. Pozostałe elementy (jeżeli nie wszystkie udało się umieścić w wolnych miejscach) są pakowane algorytmem *Next-Fit*.

Pseudokod:

**dokończyć**

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



3.7. Algorytm meta-heurystyczny

Opis

4. IMPLEMENTACJA

Struktura rozdziału zależy od koncepcji autora. Można opisać: architekturę systemu, użyte technologie, wymagania funkcjonalne i pozafunkcjonalne, dołączyć dokumentację użytkownika, dokumentację techniczną itd. Tematy te mogą stanowić podrozdziały.

4.1. Opis systemu

Opis

4.1.1. Architektura

Opis

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

Opis

4.2. Dokumentacja użytkownika

Inter

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], udostępniające główne opcje. Druga natomiast [RYS] służy do wyświetlania szczegółowych opcji/ustawień dla aktualnie wybranej pozycji z menu głównego.

[RYS]

Menu główne składa się z 3 pozycji. Pierwsza z nich („Wizualizacja”) udostępnia moduł wizualizacji. Druga („Ekspryment”) powoduje przejście do modułu eksperymentu obliczeniowego. Ostatnia pozycja odpowiada za ustawienia.

4.2.2. Moduł wizualizacji

Moduł wizualizacji umożliwia prezentację działania algorytmów (tylko algorytmy listowe) krok po kroku oraz sprawdzenie wyniku działania (wszystkie algorytmy) dla niewielkich instancji.

4.2.2.1. Wprowadzanie danych

Opis

4.2.2.2. Prezentacja działania algorytmu

Opis

4.2.2.3. Wyświetlanie wyniku

Opis

4.2.3. Moduł eksperymentu obliczeniowego

Opis

4.2.3.1. Wprowadzanie parametrów eksperymentu

Opis

4.2.3.2. Wczytywanie danych z pliku

Opis

4.2.3.3. Śledzenie przebiegu eksperymentu

Opis

4.2.3.4. Wyświetlanie wyników działania eksperymentu

Opis

4.2.4. Ustawienia

Opis

4.3. Dokumentacja techniczna

Opis

5. EKSPERYMENT OBLICZENIOWY

Wyniki eksperymentu przeprowadzonego z użyciem systemu. Struktura rozdziału zależy od koncepcji eksperymentu np. dane testowe, efektywność czasowa algorytmów, jakość rozwiązań heurystycznych itp.

6. PODSUMOWANIE

Krótkie podsumowanie pracy (ok. 2 str.) .Powtórzenie co zostało zrobione. Można podkreślić zdobyte doświadczenie, powtórzyć najbardziej interesujące wnioski z eksperymentu,

7. LITERATURA

Opis

8. ZAŁĄCZNIKI

Opis