Olaf Krawczyk 21816

Pn TP 9:15

12.12.2016

Projektowanie Efektywnych Algorytmów

# Metoda podziału i ograniczeń dla problemu plecakowego

Prowadzący: dr inż. Jarosław Mierzwa

**Cel ćwiczenia:**

Celem ćwiczenia była implementacja algorytmu podziału i ograniczeń dla problemu plecakowego. Problem plecakowy jest jednym z najbardziej rozpowszechnionych problemów optymalizacyjnych. W problemie tym mamy daną listę dostępnych przedmiotów, z której każdy przedmiot ma określoną wagę i wartość oraz plecak ustalonej pojemności. Naszym celem wybranie takiego podzbioru przedmiotów w którym sumaryczna waga nie przekracza pojemności plecaka oraz łączna wartość przedmiotów jest największa.

**Opis programu:**

Lista przedmiotów wczytywana jest do programu z pliku podanego przez użytkownika. Wczytane przedmioty zostają zapisane w wektorze zawierającym obiekty klasy Item. Każdy obiekt klasy Item posiada atrybuty przechowujące informacje o wadze oraz wartości danego przedmiotu. Dodatkowo klasa Item udostępnia szereg metod pozwalających np. na dostęp do atrybutów, co umożliwia ich porównywanie i tym samym sortowanie na podstawie np. współczynnika wartość/waga. Sortowanie tego typu jest kluczowe przy wyznaczaniu granicy kolejnych rozwiązań. Wyznaczanie granicy opera się na zachłannym dobieraniu kolejnych elementów na podstawie wspomnianego wcześniej kryterium. Program dobiera kolejne możliwe przedmioty, a w przypadku, gdy waga aktualnie dobranych przedmiotów przekracza pojemność plecaka granica powiększana jest o ułamek wartości ostatniego badanego elementu. Drzewo rozwiązań przeszukiwane jest za pomocą metody przeszukiwania wszerz. Kolejne rozwiązania reprezentowane są przez struktury Node, które przechowują informacje o rozmiarze, wartości danego rozwiązania oraz historię wybranych elementów.

Testy algorytmu zostały przeprowadzone dla problemów o rozmiarach 10, 50, 80, 150, 200, 230, 250, 260 oraz stałego rozmiaru plecaka. Dla każdego rozmiaru wykonanych zostało 100 powtórzeń algorytmu. Dla każdej ze 100 operacji generowany był losowy problem o zadanym rozmiarze. instancji. Do pomiaru czasu wykorzystana została biblioteka windows.h oraz funkcje performanceCountStart i performanceCountEndPoniżej.

**Wyniki:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 10 | 50 | 100 | 150 | 200 | 250 | 350 | 10 |
| czas [μs] | 0,0101685 | 0,132937 | 0,61744 | 3,99879 | 19,4547 | 56,226 | 378,813 | 0,0101685 |

Tabela 1 Wyniki pomiarów dla B&B

Rysunek 1 Czas wykonania w zależności od rozmiaru problemu B&B

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | 5 | 10 | 15 | 20 | 25 | 27 |
| czas [μs] | 0.00159475 | 0.018908 | 0.517232 | 15.2405 | 495.426 | 2147.1 |

Tabela 1 Wyniki pomiarów dla Brute Force

Rysunek 2 Czas wykonania w zależności od rozmiaru problemu BF

Powyższy wykres pokazuje, że czas znalezienia korzystnego rozwiązania jest ściśle związany z rozmiarem problemu. Algorytm Branch and Bound radzi sobie z znacznie większymi instancjami problemu niż Brute Force. Czas wykonania B&B również jest zauważalnie krótszy, niż Brute Force.

**Bibliografia:**

Z. Michalewicz, D. Fogel „Jak to rozwiązać, czyli nowoczesna heurystyka”