1. Kod źródłowy

```
#create the density value for each item
for sublist in lst:
    field = sublist[1] * sublist[2]
    sublist.append(round(sublist[3]/field, 3))
#sort the list by the density of subjects
lst.sort(key = lambda x: x[4], reverse = True)
```

Wybrałem powyższy fragment kodu, ponieważ jest on w moim przypadku najważniejszy biorąc pod uwagę całościowy projekt poszukiwania najlepszego rozwiązania dla problemu plecakowego. Można powiedzieć, że mój algorytm ewoluował zaczynając od losowego wypełniania plecaka przypadkowymi wartościami poprzez algorytm zachłanny wybierający w danej chwili element o największej wartości, aby finalnie wykorzystać aproksymacyjne rozwiązanie zaproponowane przez Georga Dantziga. Koncepcja zawarta w tym fragmencie kodu uświadomiła mi także, że dla danego programu może istnieć więcej niż jedno rozwiązanie zachłanne i od jego wyboru może zależeć to na ile skuteczny będzie nasz program.

2. Wyniki i podsumowanie

Problem plecakowy jest problemem NP-trudnym. Z tego względu skupiam się na znalezieniu rozwiązania, które przyniesie jak najlepsze rezultaty, lecz niekoniecznie optymalne. W tym celu porównuję ze sobą trzy rozwiązania. Pierwsze z nich to po prostu przejście przez całą listę i próba włożenia do plecaka każdego elementu. Jest to oczywiście mało wydajne rozwiązanie, które tylko w bardzo nielicznych przypadkach może przynieść stosunkowo dobre rezultaty. Drugim użytym przez mnie algorytmem jest klasyczny algorytm zachłanny. W danym momencie znajdujemy najlepsze rozwiązanie czyli przedmiot o największej wartości i wkładamy go do plecaka. Co prawda dla tego typu problemu nie przynosi on najlepszych rezultatów, aczkolwiek można zaobserwować znaczą różnicę w porównaniu z losowym doborem przedmiotów.

Na końcu stosuję algorytm wynaleziony przez Georga Danziga. Wyznaczam „gęstość przedmiotu" tzn. $\frac{wartość}{pole}$ i na tej podstawie sortuję tablicę obiektów. Następnie począwszy od przedmiotów o największej gęstości wypełniam nimi plecak. Rozwiązanie to jest tym lepsze im więcej do dyspozycji mamy przedmiotów pierwszej klasy (tych o największej gęstości). Jest to co prawda także rodzaj algorytmu zachłannego, ale im większa jest ilość przedmiotów i pojemność plecaka tym większą możemy zaobserwować przewagę tego algorytmu nad podstawowym algorytmem zachłannym; w przypadku tablicy 1000x1000 jest to wartość niespełna 10000. Zebrane przeze mnie wyniki umieszczam w poniższych tabelach.

losowe pakowanie elementów	
rozmiar plecaka	**osiągnięta wartość**
20x20	92
100x100	1881
500x500	47633
1000x1000	188991

algorytm zachłanny	
rozmiar plecaka	**osiągnięta wartość**
20x20	130
100x100	2353
500x500	60114
1000x1000	239806

algorytm aproksymacyjny	
rozmiar plecaka	**osiągnięta wartość**
20x20	131
100x100	2392
500x500	62274
1000x1000	249416

W celu reprezentacji plecaka utworzyłem tablicę, która składa się z X list o X wartościach każda, gdzie X jest długością jednego boku plecaka. Początkowo wszystkie wartości to zera. Sprawdzając czy jest w plecaku miejsce dla danego przedmiotu algorytm sprawdza czy odpowiednie indeksy w podlistach to zera. Jeśli tak to program zwraca informację, że w plecaku zostaje spakowany dany przedmiot, a wszystkie zera w odpowiednich miejscach zostają zamienione na jedynki. Dla algorytmu losowego i zachłannego czas przetwarzania jest podobny dlatego dokonuję porównania tylko z algorytmem aproksymacyjnym, gdzie trzeba też uwzględnić czas przetwarzania funkcji gęstości. Wyniki prezentuję w poniższych tabelach oraz na wykresie.

losowe pakowanie elementów	
rozmiar plecaka	**czas przetwarzania [s]**
20	0.010
100	0.756
500	813.130
1000	6189.160

algorytm aproksymacyjny	
rozmiar plecaka	**czas przetwarzania [s]**
20	0.017
100	0.953
500	946.692
1000	6728.793