SK.POP.5
22 stycznia 2023

Jan Jędrzejewski
Dominik Łopatecki

Problem plecakowy, PBIL

Projekt Końcowy

Streszczenie założeń z Projektu Wstępnego

W ramach projektu zaimplementowany zostanie algorytm PBIL, algorytm programowania dynamicznego, algorytm heurystyczny oraz A* do rozwiązania problemu plecakowego.
W celu zbadania jakości działania algorytmu PBIL oraz porównania go z pozostałymi algorytmami przeprowadzone zostaną badania. Plan eksperymentów przewiduje sprawdzenie zachowania algorytmów, dla:

- Zbiorów o następujących licznościach przedmiotów n = [10, 30, 40, 100, 500]
- Rozmiarze plecaka w dwóch wariantach: 2-krotność maksymalnej wagi przedmiotu, połowie sumy wag wszystkich przedmiotów
- Skorelowanych oraz nieskorelowanych danych

W przypadku algorytmu PBIL zbadany zostanie wpływ parametrów: M-rozmiar generowanej populacji, N-liczba wyselekcjonowanych najlepszych chromosomów, lr- learning rate (pl. wskaźnik uczenia się) na jego zachowanie.

Opis funkcjonalny

pbil.py

Implementacja algorytmu PBIL. W naszej implementacji dodana została funkcja kary, która wyliczana jest jako logarytm naturalny z nadwyżki wag w plecaku(różnica sumy wag uwzględnionych przedmiotów i pojemności plecaka). Funkcja ta liczona jest tylko dla rozwiązań niedopuszczalnych oraz mnożymy ją przez wybrany przez nas współczynnik.

Atrybuty klasy:

- lr - learning rate
- N - liczba wyselekcjonowanych najlepszych chromosomów
- M - rozmiar generowanej populacji
- num_of_iter - liczba iteracji algorytmy
- mut_prob - prawdopodobieństwo mutacji danego elementu w rozwiązaniu (mutujemy zawsze)
- mut_shift - siła mutacji
- knapsack_capacity - pojemność plecaka
- num_of_elements - ilość przedmiotów
- values - lista zawierająca wartości przedmiotów
- weights - lista zawierająca wagi przedmiotów
- penalty_rate - współczynnik, przez który mnożona jest wyliczona kara

Opis funkcji:

- get_data
  - Funkcja, która z otrzymanych danych wyznacza listę wag, listę wartości przedmiotów oraz pojemność plecaka
- solve_knapsack_problem
  - Funkcja rozwiązująca problem plecakowy dla zadanych danych
  - Jest ona głównym koordynatorem całego procesu obliczeń

- get_best_from_population
    - Funkcja ta tworzy końcową populację, z której wybierane jest najlepsze rozwiązanie
- is_stop_condition
    - Funkcja sprawdzająca, czy wszystkie wartości w wektorze prawdopodobieństw są równe 0 lub 1. Taka sytuacja oznacza, że algorytm wyliczył rozwiązanie
- selection
    - Funkcja wybierająca N najlepszych rozwiązań z populacji początkowej M elementowej
- calculate_quality
    - Funkcja obliczająca jakość rozwiązania w następujący sposób:
        - Jeśli rozwiązanie jest dopuszczalne, to zwracana jest suma wartości zapakowanych przedmiotów
        - Jeśli nie to jakość rozwiązania jest pomniejszana o wyliczoną karę
            - Wyjątkiem jest sytuacja, gdy wyliczana jest jakość rozwiązania końcowego, w takim przypadku wykonywana jest funkcja *return_valid_solution*
- return_valid_solution
    - Funkcja ta doprowadza rozwiązanie do stanu dopuszczalnego, usuwając kolejne elementy, do momentu, w którym suma wag elementów znajdujących się w plecaku nie będzie przekraczać jego pojemności
- penalty_function
    - Funkcja kary, która wyliczana jest jako logarytm naturalny z nadwyżki wag w plecaku(różnica sumy wag uwzględnionych przedmiotów i pojemności plecaka)
    - Kara jest mnożona przez współczynnik, co sprawia, że rozwiązanie nie będzie lepsze niż najlepsze rozwiązanie dopuszczalne
- mutation
    - Funkcja odpowiedzialna za mutacje rozwiązania
    - Mutacja polega na przybliżeniu wartości w wektorze prawdopodobieństw losowo do 0 lub 1.

astar.py

Implementacja algorytmu A*, heurystyczna metoda znajdowania najkrótszej ścieżki w drzewie. Przedmioty w plecaku są reprezentowane przez drzewo binarne. Wysokość drzewa odpowiada liczbie przedmiotów w problemie plecakowym, a liczba potencjalnych rozwiązań (liści drzewa) to 2 do potęgi równej liczbie przedmiotów. Metoda zaczyna od korzenia i generuje kolejne węzły w drzewie aż dojdzie do pierwszego liścia. Dodatkowo została zaimplementowana klasa Node(), reprezentująca węzeł w drzewie.

Atrybuty klasy:

- knapsack_capacity - pojemność plecaka
- num_of_elements - ilość przedmiotów
- dataset - posortowana pod względem ilorazu wartości do wagi lista przedmiotów

Opis funkcji:

- __init__
  - Inicjalizacja modelu, podanie listy przedmiotów i maksymalnej pojemności plecaka
- _sort_dataset
  - Zwraca posortowaną listę przedmiotów
- _get_price_to_weight_ratio
  - Kryterium sortowania przedmiotów, zwraca iloraz wartości i wagi
- _generate_children
  - Tworzy potomków obecnego punktu roboczego
- _set_g
  - Ustawia:
    - g-wartość plecaka w danym node
    - weight - wagę plecaka w danym node
- _set_h
  - Ustawia wartość h - przewidywana heurystyczną metodą wartość plecaka po dołożeniu kolejnych przedmiotów
    - Metoda heurystyczna - dokładamy po kolei przedmioty do plecaka jeśli się zmieszczą. Przedmioty są uszeregowane malejąco pod względem współczynnika wartość / waga
- _set_f
  - Ustawia wartości g i h dla podanego node'a, f to wartość funkcji decyzyjnej która określa "jak dobry" jest dany wierzchołek - obliczana wzorem g + h
- solve
  - Rozwiązuje problem plecakowy, zwraca wartość najlepszego znalezionego plecaka
- get_knapsack
  - Zwraca listę przedmiotów wybranych do docelowego plecaka

dynamic.py

Implementacja algorytmu programowania dynamicznego rozwiązującego problem plecakowy. Podczas implementacji wagi przedmiotów są mnożone razy 100, w celu umożliwienia obliczania przykładów, gdzie wartości wag mają 2 miejsca po przecinku.

Atrybuty klasy:

- knapsack_capacity - pojemność plecaka
- num_of_elements - ilość przedmiotów
- values - lista zawierająca wartości przedmiotów
- weights - lista zawierająca wagi przedmiotów

Opis funkcji:

- get_data
  - Funkcja, która z otrzymanych danych wyznacza listę wag, listę wartości przedmiotów oraz pojemność plecaka
- solve_knapsack_problem
  - Funkcja rozwiązująca problem plecakowy dla zadanych danych

heuristic.py

Implementacja heurystycznej metody

Atrybuty klasy:

- knapsack_capacity - pojemność plecaka
- dataset - posortowana pod względem ilorazu wartości do wagi lista przedmiotów
- packed - lista przedmiotów spakowanych do plecaka

Opis funkcji:

- __init__
  - Inicjalizacja modelu, podanie listy przedmiotów i maksymalnej pojemności plecaka
- _get_price_to_weight_ratio
  - Kryterium sortowania przedmiotów, zwraca iloraz wartości i wagi
- _sort_elements
  - Sortuje listę przedmiotów
- solve
  - Rozwiązuje problem plecakowy, zwraca wartość najlepszego znalezionego plecaka
- get_knapsack
  - Zwraca listę przedmiotów wybranych do docelowego plecaka

generate_data.py

Metody generujące listę przedmiotów o zadanej długości i podanej maksymalnej wadze. Wartości są losowane z rozkładem jednostajnym w podanym przedziale. Dostępne są trzy poziomy korelacji wartości z wagą przedmiotu.

Opis funkcji:

- generate_uncorrelated
  - W ogóle nie powiązane wartości, niezależnie wylosowane
- generate_semicorrelated
  - Do wartości przedmiotu dodawana jest waga
- generate_correlated
  - Wartość przedmiotu jest równa wadze + połowie maksymalnej wagi

Opis projektu

Interpretacja problemu plecakowego

Problem plecakowy, wielokrotnie poruszany na wykładach z optymalizacji, polega na maksymalizacji wartości przedmiotów które wybieramy. Istnieje jedno ograniczenie, wybrane przedmioty muszą mieścić się w plecaku. Zagadnienie nie pozwala na dzielenie przedmiotów na części. Często przedstawiany jako problem złodzieja okradającego sklep. Pragnie on zmieścić przedmioty o jak największej wartości do swojego plecaka.

pi - wartość i-tego przedmiotu
wi - waga i-tego przedmiotu
xi - informacja o tym czy wybieramy przedmiot
W - ograniczenie (maksymalna waga) plecaka

$$max \sum_{i=1}^{n} x_i p_i$$

$$\sum_{i=1}^{n} x_i w_i \leqslant W$$

$$x_i \in \{0,1\}$$

W matematyce problem plecakowy możemy przedstawić w postaci maksymalizacji funkcji zysku. Szukamy optymalnego wektora x o binarnych wartościach [x1, x2, x3, …, xn]. 0 oznacza że nie wybieramu danego przedmiotu, 1 dla zabranych.

Metoda heurystyczna - sortowanie przedmiotów według współczynnika

Do rozwiązania problemu plecakowego można użyć metody heurystycznej. Wyznacznikiem czy dany przedmiot opłaca się zabrać może być stosunek wartości do wagi. Innymi słowami jak duża wartość znajduje się w jednostce wagi. Poniżej przykład.

w - waga przedmiotu,
p - wartość przedmiotu,
W = maksymalna waga

założenia: W=12
Chcemy wybrać przedmioty o maksymalnej wartości

i	1	2	3	4	5	6
p_i	20	5	10	5	6	3
w_i	6	2	5	3	4	3
p_i/w_i	3.33	2.5	2	1.66	1.5	1

W powyższej tabeli umieszczono przedmioty posortowane w kolejności malejącej pod względem atrybutu p/w. Po kolei dokładamy przedmioty do pustego plecaka, w momencie gdy przedmiot się nie zmieści pomijamy go i próbujemy kolejny. Kończymy gdy przejdziemy przez całą listę, bądź gdy wypełnimy plecak plecak. Poniżej kolejne kroki:

1) Do plecaka wkładamy przedmiot 1. Łączna waga: 6; Łączna wartość: 20
2) Do plecaka wkładamy przedmiot 2. Łączna waga: 8; Łączna wartość: 25
3) Do plecaka nie możemy włożyć przedmiotu nr. 3 ponieważ 8 + 5 > 12
4) Do plecaka wkładamy przedmiot 4. Łączna waga: 11; Łączna wartość: 30
5) Do plecaka nie zmieszczą się przedmioty 5 i 6. Kończymy z plecakiem o wartości 30

W plecaku otrzymaliśmy przedmioty 1, 2 i 4. Ich łączna wartość to 30. Nie jest to maksymalna do uzyskania wartość, ale w stosunkowo prosty sposób pozwala zbliżyć się do optymalnego wyniku. Gdybyśmy wyjęli przedmiot 4 i w jego miejsce włożyli przedmiot 5 to waga plecaka byłaby równa ograniczeniu, a wartość plecaka wyniosłaby 31.

Ta metoda nie byłaby efektywna dla dużej liczby przedmiotów i dla stosunkowo niewielkiego plecaka. Duże elementy mogą zapchać plecak i nie będziemy go w stanie wypełnić tak jak w powyższym przykładzie.

Rozwiązanie sztucznej inteligencji

Do rozwiązania problemu plecakowego z pomocą przyjdzie nam uczenie maszynowe i algorytmy przeszukiwania przestrzeni. Wszystkie możliwości spakowania przedmiotów do plecaka potraktujemy jako bogatą dziedzinę zadania. Spośród niej musimy znaleźć punkt optymalny - to znaczy zestawienie przedmiotów które posiadają największą wspólną wartość i mieszczą się w ograniczeniu wagi.

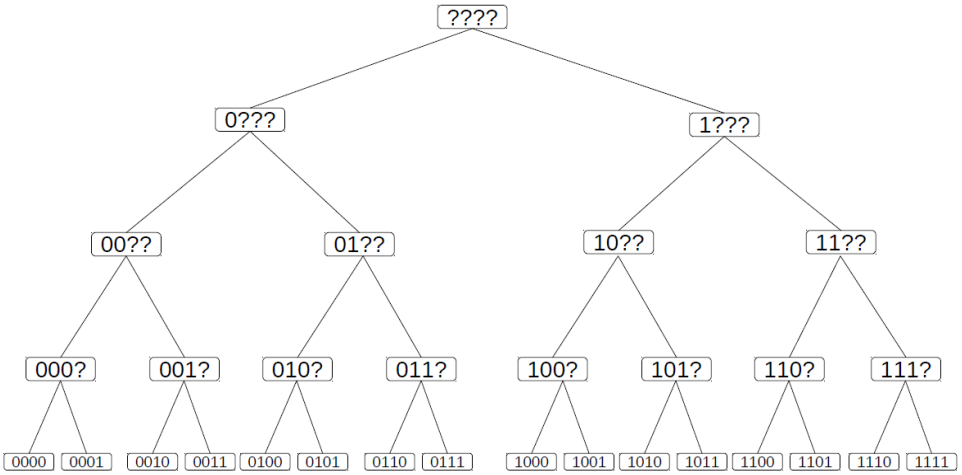
Zestaw przedmiotów będziemy reprezentować poprzez ciągi bitowe $x_i = \{0, 1\}$. Gdzie:

- ➔ 1 oznacza że przedmiot został wybrany do plecaka,
- ➔ 0 przeciwnie - pozostawiamy przedmiot.

Wówczas dla n-przedmiotów mamy 2^n kombinacji różnych sposobów spakowania ich do plecaka. Do przeszukania problemu w którym występuje ponad 30 przedmiotów nie jesteśmy w stanie użyć standardowych metod takich jak porównanie wszystkich możliwości.

Natomiast można do tego zagadnienia wykorzystać algorytmy do przeszukiwania łańcuchów znaków na przykład algorytmy genetyczne. Populacją początkową może być dowolnie wybrany podzbiór przedmiotów. Z każdą iteracją algorytmu poprzez mutację, krzyżowanie oraz selekcję będziemy przybliżać się do optymalnego zestawu przedmiotów.

Również możemy skorzystać z algorytmów przeszukujących struktury drzewiaste np. A*.



Gdy przedstawimy przestrzeń w postaci binarnego drzewa, tak jak powyżej, gdzie korzeniem będzie niezdefiniowany zestaw przedmiotów, a kolejne poziomy będą miały sprecyzowany pierwszy bit oznaczający czy zabieramy przedmiot o danym numerze czy nie. Głębokość takiego drzewa jest równa liczbie przedmiotów n, natomiast liczba liści to 2^n.

Opis wykorzystywanych algorytmów

Algorytm PBIL (Population-Based Incremental Learning)

Algorytm Population-Based Incremental Learning należy do rodziny algorytmów ewolucyjnych, a dokładnie do klasy Estimation of Distribution Algorithm. Algorytmy tej klasy cechują się tym, że mechanizm ewolucyjny realizowany jest przy pomocy modelu probabilistycznego. Dużym uproszczeniem jest fakt, że rozkład prawdopodobieństw każdego z bitów jest niezależny, co pozwala na osobny proces uczenia oraz generowania punktu dla każdej zmiennej.
Rozkład prawdopodobieństwa w PBIL reprezentowany jest jako wektor zawierający prawdopodobieństwo wystąpienie 1 w danym bicie.

$$p^t = [p_1^t, p_2^t, p_3^t, \ldots, p_n^t]$$, gdzie:

t - numer iteracji, n - rozmiar chromosomu

Algorytm PBIL- schemat działania:

1. Tworzymy populację P^t o rozmiarze M, bazując na p^t
2. Wybieramy z P^t N najlepszych rozwiązań tworząc zbiór O^t
3. Następnie aktualizowany jest wektor prawdopodobieństw zgodnie ze wzorem:

$$p^{t+1} = (1 - lr) * p^t + lr * \frac{1}{N} \sum_{x \epsilon O^t} x$$, gdzie:

lr - współczynnik uczenia

4. Mutacja wyliczonego p^{t+1}
5. Jeśli warunek stopu nie jest spełniony przejście do punktu 1.

Jako warunek stopu można zastosować maksymalna liczbę iteracji. Dodatkowo algorytm powinien zakończyć się w sytuacji prawie rozstrzygniętej, czyli takiej gdzie wartość każdego bitu będzie w określonym sąsiedztwie 0 lub 1.

Przykład obliczeń

1. p^t = [0.5, 0.5, 0.5, 0,5]
2. Zbiór O^t = {[0, 1, 1, 0], [1, 1, 1, 0], [0, 0, 1, 0]}
3. Lr = 0.2
4. p_1^{t+1} = 0.8 * 0.5 + 0.2 * ⅓ = 0.467
5. p_2^{t+1} = 0.8 * 0.5 + 0.2 * ⅔ = 0.533
6. p_3^{t+1} = 0.8 * 0.5 + 0.2 * 1 = 0.6
7. p_4^{t+1} = 0.8 * 0.5 + 0.2 * 0 = 0.4

Uzyskany rezultat jest zgodny z przewidywaniami, prawdopodobieństwo wystąpienia jedynki wzrosło w bitach, dla których większość osobników ze zbioru O^t posiada ten bit ustawiony na 1. Natomiast analogicznie zmalało, gdy większość osobników na tym bicie ma wartość 0.

IP (Integer Programing)

Jako algorytm porównawczy zastosujemy metodę programowania dynamicznego, która pozwoli na uzyskanie optimum globalnego.

Zasada działania algorytmu

Tworzymy tablicę **A** o wymiarach **n+1** $\times$ **W+1**, gdzie **n** - liczba przedmiotów, **W** - pojemność plecaka. Komórka **(i,j)** w naszej tablicy będzie przechowywało największą możliwą wartość, przy założeniu, że pojemność plecaka jest nie większa niż **j** oraz bierzemy pod uwagę pierwsze **i** przedmiotów. Dodatkowo tablica **c[n]** zawiera informacje o wartościach przedmiotów, a tablica **w[n]** o wagach przedmiotów.
Komórki o indeksach **(0,j)** oraz **(i, 0)** dla każdego **i** oraz **j** wypełniamy 0. Następnie schemat działania algorytmu wygląda w następujący sposób:

- $A(i,j) = A(i-1,\ j)$, jeśli wiemy, że **i-ty** przedmiot nie zmieści się do plecaka o pojemności **j**
- $A(i,j) = max(A(i-1, j),\ A(i-1,\ j-w[i]) + c[i])$, jeśli **i-ty** przedmiot zmieści się do plecaka o pojemności **j**

Rozwiązanie znajduje się w A**(n, W)** .

Przykład

- c = [3, 4, 6]
- w = [2, 3, 4]
- W = 6
-

i/j	**0**	**1**	**2**	**3**	**4**	**5**	**6**
0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3
2	0	0	3	4	4	7	7
3	0	0	3	4	6	7	9

- Otrzymany rezultat to 9
  - Możemy to łatwo zweryfikować:
    - Bierzemy tylko przedmiot nr 1, wartość plecaka będzie równa 3
    - Bierzemy tylko przedmiot nr 2, wartość plecaka będzie równa 4
    - Bierzemy tylko przedmiot nr 3, wartość plecaka będzie równa 6
    - Bierzemy tylko przedmioty nr 1 i 2, wartość plecaka będzie równa 7
    - Bierzemy tylko przedmioty nr 1 i 3, **wartość plecaka będzie równa 9**
    - Bierzemy tylko przedmioty nr 2 i 3, za duża suma wag
    - Bierzemy wszystkie przedmioty, za duża suma wag
- **Przykładowe wyliczenia:**
  - Dla komórki (3,6) korzystamy ze wzoru:
    $A(i,j) = max(A(i-1,j),\ A(i-1,\ j-w[i]) + c[i])$,
    który zwraca wartość $A(i-1,\ j-w[i]) + c[i]$
  - Dla komórki (2, 2) korzystamy ze wzoru: $A(i,j) = A(i-1,\ j)$

- Dla komórki (3, 5) korzystamy ze wzoru:
  $A(i,j) = max(A(i-1,j), A(i-1, j-w[i]) + c[i])$,
  który zwraca wartość $A(i-1,j)$

Opis zbiorów danych

Eksperymenty planujemy przeprowadzić na zbiorach danych wygenerowanych w sposób przedstawiony w poniższej tabeli. Dla wyróżnionych trzech stopni korelacji danych:

	Poziom korelacji	Waga	Wartość
1	Nieskorelowane	U[1, v]	U[1, v]
2	Częściowo skorelowane	U[1, v]	waga + U[-v/2, v/2]
3	Skorelowane	U[1, v]	waga + v/2

v - maksymalna wartość wagi, dla naszych badań przyjmiemy wartość v = 20
U[a, b] - dane losowane rozkładem ciągłym z przedziału od a do b

Wyniki eksperymentów

Parametry PBIL

Eksperymenty przeprowadziliśmy w notatniku jupyter o nazwie `experiments.ipynb`. Badaliśmy wpływ poniższych parametrów algorytmu PBIL:

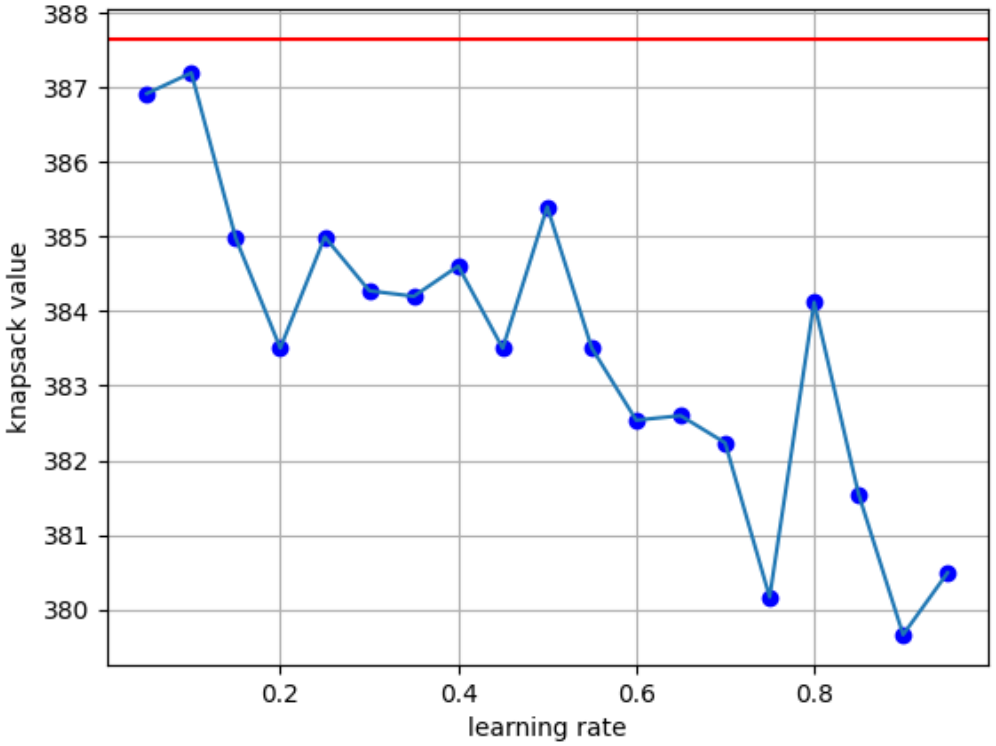
- M - liczebność populacji
- N - ilość potomków po selekcji
- lr - poziom uczenia

Poniżej zestawienie średnich rezultatów modelu w zależności od parametrów M i N, z zastrzeżeniem, że wartości M oraz N z tabeli należy pomnożyć przez liczbę przedmiotów które pakujemy do plecaka. Eksperyment wykonany dla 50 przedmiotów i lr=0.3, N <= M

Dynamic model solution: 385.58

	N=0.25	N=0.5	N=0.75	N=1	N=1.5	N=2	N=3
M=0.5	355.388	228.848	0.000	0.000	0.000	0.000	0.000
M=1	373.748	372.722	359.106	226.538	0.000	0.000	0.000
M=1.5	382.184	378.296	377.185	371.479	226.183	0.000	0.000
M=2	379.142	382.225	381.214	378.715	366.464	234.878	0.000
M=2.5	382.613	382.237	382.598	380.389	375.851	370.342	0.000
M=3	382.707	383.029	383.476	382.611	381.017	378.079	249.084
M=5	383.675	383.855	383.573	384.470	384.687	384.182	381.326
M=7.5	384.917	384.695	384.521	384.888	385.206	384.431	384.204
M=10	385.028	385.076	385.297	385.168	384.977	384.812	383.854

Wartość plecaka w zależności od parametru learning rate, czerwona linia to wartość zwrócona przez metodę programowania dynamicznego.



Wyniki na powyższym wykresie, jak i w tabeli parametrów M i N zostały sporządzone poprzez uśrednienie wyników algorytmu PBIL z dziesięciu uruchomień.

Wnioski

Wraz ze wzrostem czynnika M, rosną wyniki algorytmu PBIL, jednak dla wartości 10 i więcej obliczenia wykonują się znacznie dłużej i dla eksperymentów z 500 lub 1000 przedmiotów czas wykonania algorytmu drastycznie by urósł. Dlatego na potrzeby kolejnych eksperymentów wykorzystujemy parametr M=7.5.

Njlepsze rezultaty otrzymaliśmy dla wartości N około 1. Dla zbyt dużej liczby potomków algorytm prawdopodobnie jest niewystarczająco eksploracyjny, ponieważ dużo słabych osobników przechodzi do kolejnej populacji. Do kolejnych eksperymentów przyjęliśmy N=1.

Parameter learning rate w zakresie od 0.2 do 0.5 dawał stabilnie dobre rezultaty, dlatego do następnych badań przyjęliśmy wartość lr z tego przedziału, lr=0.3.

Ilość przedmiotów i maksymalna waga plecaka

Dla najlepszych znalezionych parametrów algorytmu porównaliśmy go z pozostałymi zaimplementowanymi metodami, oraz sporządziliśmy zestawienie wyników dla różnej korelacji pomiędzy wagą oraz wartością, dla rozmiarów zbiorów przedmiotów oraz dla różnych maksymalnych dopuszczalnych wag plecaka. Bezpośrednio porównaliśmy algorytm PBIL do metody programowania dynamicznego i wyznaczyliśmy o ile procent gorszy wynik otrzymaliśmy za pomocą PBIL.

W kolumnach zestawienie wartości plecaka dla konkretnej liczby przedmiotów, zbiory danych wygenerowaliśmy zgodnie z opisem zbiorów danych. W - maksymalna waga

Dane nieskorelowane

W = 2 * maksymalna_waga

	10	20	30	50	100	500
heuristic	72.53	86.12	132.27	143.480000	171.550000	382.900000
astar	72.53	88.05	132.27	143.480000	174.540000	386.580000
dynamic	72.53	88.05	132.27	143.480000	174.540000	386.580000
pbil	72.53	88.05	132.27	100.870000	115.140000	31.250000
pbil to dynamic diff [%]	0.00	0.00	0.00	29.697519	34.032314	91.916292

W = połowie sumy wag wszystkich przedmiotów

	10	20	30	50	100	500
heuristic	76.630000	151.230000	262.710000	411.860000	823.01	4157.810000
astar	79.990000	152.160000	262.710000	411.860000	823.01	4158.820000
dynamic	79.990000	152.160000	262.710000	411.860000	823.01	4158.930000
pbil	76.630000	151.788000	261.032000	411.500000	823.01	4156.242000
pbil to dynamic diff [%]	4.200525	0.244479	0.638727	0.087408	0.00	0.064632

Dane częściowo skorelowane

W = 2 * maksymalna_waga

	10	20	30	50	100	500
heuristic	45.270000	75.390000	64.98000	90.260000	106.30000	186.880000
astar	46.230000	76.590000	73.54000	90.260000	110.60000	188.520000
dynamic	46.230000	76.590000	73.54000	90.260000	110.60000	188.520000
pbil	45.708000	76.214000	69.45600	72.920000	85.34000	42.470000
pbil to dynamic diff [%]	1.129137	0.490926	5.55344	19.211168	22.83906	77.471886

W = połowie sumy wag wszystkich przedmiotów

	10	20	30	50	100	500
heuristic	6.210000e+01	121.66000	234.330000	399.620000	853.150000	3846.820000
astar	6.210000e+01	121.66000	234.330000	402.030000	854.050000	3847.170000
dynamic	6.210000e+01	121.66000	234.330000	402.030000	854.050000	3847.500000
pbil	6.210000e+01	121.45600	232.954000	399.876000	853.348000	3841.362000
pbil to dynamic diff [%]	-1.144191e-14	0.16768	0.587206	0.535781	0.082197	0.159532

Dane skorelowane

W = 2 * maksymalna_waga

	10	20	30	50	100	500
heuristic	93.340000	129.16	115.900000	144.710000	219.530000	289.010000
astar	99.850000	129.16	119.830000	150.000000	219.980000	0.000000
dynamic	99.850000	129.16	119.830000	150.010000	220.020000	290.040000
pbil	99.610000	129.16	76.970000	79.990000	66.770000	66.710000
pbil to dynamic diff [%]	0.240361	0.00	35.767337	46.676888	69.652759	76.999724

W = połowie sumy wag wszystkich przedmiotów

	10	20	30	50	100	500
heuristic	93.990000	240.290000	347.290000	592.420000	1220.230000	6023.090000
astar	105.970000	241.740000	355.000000	596.000000	1220.960000	0.000000
dynamic	105.970000	241.740000	355.010000	596.000000	1221.010000	6034.230000
pbil	105.408000	235.206000	354.442000	585.626000	1210.836000	6023.718000
pbil to dynamic diff [%]	0.530339	2.702904	0.159995	1.740604	0.833245	0.174206

Wnioski - porównanie metod

Metoda dynamicznego programowania we wszystkich przeprowadzonych testach dawała najlepsze rezultaty. Algorytm A* praktycznie dla wszystkich przypadków dawał jednakowy rezultat co programowanie dynamiczne, tylko dla danych silnie skorelowanych dawał lekko gorsze wyniki przy mniejszych liczbach przedmiotów. Dla dużej liczby przedmiotów silnie skorelowanych algorytm A* nie zwracał żadnych rezultatów przez kilkadziesiąt minut, więc przerwaliśmy dla niego ten test (stąd wartość zero w tabeli). Heurystyczna metoda dla mniejszej liczby przedmiotów dawałą wyniki na poziomie A*, dla liczby przedmiotów 100 i więcej nieznacznie odstawała.

Wnioski - algorytm PBIL

Algorytm niezależnie od poziomu korelacji danych dawał bardzo dobre rezultaty dla “pojemnego” plecaka. Miał nieznaczne odstające wyniki dla małej liczby przedmiotów. Dla plecaka u niewielkiej pojemności algorytm zwracał porównywalne rozwiązania dla zbiorów o co najwyżej 30 przedmiotach. Dla większej liczby odnotowaliśmy wartość plecaka otrzymanego metodą PBIL od 30 do nawet 90 procent mniejszą niż metodą programowania dynamicznego.

Wykorzystane narzędzia oraz biblioteki

Biblioteki pythonowe

- numpy
- math
- random
- copy
- pandas

Narzędzia

- Jupyter Notebook

Źródła

1) Prezentacje wykładowe z przedmiotu POP prof. J. Arabasa
2) Artykuł ze strony podanej w literaturze: DOI:10.3390/app11199136
3) https://en.wikipedia.org/wiki/Population-based_incremental_learning
4) https://pl.wikipedia.org/wiki/Problem_plecakowy