Dokumentacja końcowa projektu z POP

Zadanie projektowe nr 8

Krupa Olga

Szydłowski Kamil

Polecenie

Twoim zadaniem jest opracowanie metody, która pozwoli Ci umieścić w magazynie o skończonej pojemności c jak największą liczbę kontenerów. Wyjściem programu powinna być wizualizacja rozkładu kontenerów w magazynie.

Opis problemu i jego sposobu rozwiązania

Opis problemu:

W ramach zadania przyjmujemy postać magazynu oraz kontenerów jako prostopadłościan. Wymiary magazynu (wysokość x szerokość x głębokość) zostają podane przez użytkownika. Kontenery również zostają podane przez użytkownika w postaci wymiarów (wysokość x szerokość x głębokość) oraz ilości danego typu kontenerów.

Rozwiązaniem danego zadania jest umieszczenie jak największej ***liczby*** kontenerów w magazynie, biorąc pod uwagę maksymalizację zajętej przestrzeni.

Założenia i ograniczenia:

1. Kontenery mogą się obracać we wszystkich osiach.
2. Niedopuszczalne jest takie rozmieszczenie kontenerów w magazynie, w których występowałyby zjawiska niefizyczne, np. unoszenie się kontenera w powietrzu.
3. Ponieważ nie znamy rozkładu ciężkości kontenera, ze względów bezpieczeństwa nie dopuszczamy, aby jakakolwiek część kontenera wystawała poza inny.
4. Kontenery są niepodzielne.

Opis sposobu rozwiązania:

Na niniejszy problem składają się w rzeczywistości dwa podproblemy (pierwszy z nich rozwiązujemy przy użyciu algorytmu A*, a drugi – algorytmu ewolucji różnicowej z modyfikacją):

1. **Maksymalizacja liczby kontenerów**

A) przestrzeń przeszukiwań: drzewo z węzłami zawierającymi wektory;

B) sposób reprezentacji: wektory z ‘0’, ‘1’ lub ‘?’, gdzie i-ta pozycja oznacza decyzję dotyczącą i-tego kontenera (‘0’ oznacza, że dany kontener nie zostanie umieszczony w magazynie, ‘1’ - zostanie umieszczony, ‘?’ - decyzja na danym etapie nie została jeszcze podjęta);

C) funkcja zysku: suma ‘1’, czyli liczba kontenerów umieszczonych na danym etapie w magazynie;

D) funkcja heurystyczna: maksymalna liczba kontenerów, które mają szansę zostać umieszczone w magazynie. Aby maksymalizować ich liczbę, sumujemy liczbę tych kontenerów, które są najmniejsze i których objętość nie przekracza pojemności dostępnej w danym momencie w magazynie (funkcja jest nadmiernie optymistyczna kosztem założenia o niepodzielności kontenerów).

2. **Rozkład kontenerów w magazynie z uwzględnieniem założeń i ograniczeń**

A) Do tego celu używamy algorytmu ewolucji różnicowej z modyfikacją.

B) osobnik: wszystkie dane kontenery (reprezentowane przez tensory trzeciego rzędu) wraz z przypisanymi pozycjami w przestrzeni i wymiarami w znaczącej kolejności

B) populacja: zbiór wszystkich osobników w danej iteracji. Stały rozmiar.

C) selekcja: losowy wybór rozkładem jednostajnym trzech różnych osobników z populacji - kandydatów

D) mutacja:

1) utworzenie zmutowanej pozycji poprzez zsumowanie w każdym wymiarze pozycji kontenera z pierwszego kandydata z różnicą pozycji kontenera z drugiego i trzeciego kandydata, która to różnica jest przemnożona przez parametr F

2) permutacja dwóch losowych wymiarów (obrót) kontenera z prawdopodobieństwem P (modyfikacja oryginalnego algorytmu ewolucji różnicowej)

E) krzyżowanie: losowy wybór pozycji w każdym wymiarze. Wybieramy pozycję zmutowanego kontenera z prawdopodobieństwem CR. Permutacji wymiarów nie krzyżujemy (zawsze wybieramy zmutowaną permutację wymiarów).

F) funkcja celu: suma jednostek sześciennych, w których kontenery nachodzą na siebie, wystają poza magazyn oraz tych jednostek sześciennych, które pod kontenerem są puste aż do kolejnego kontenera lub podłogi (z = 0)

G) Selekcja osobników do następnej iteracji: wybieramy te nowe osobniki, których wartość funkcji celu jest wyższa od wartości funkcji celu dotychczasowego osobnika (tego na podstawie, którego zmutowaliśmy nowego osobnika)

Głównym algorytmem w rozwiązaniu jest algorytm A* do maksymalizacji liczby kontenerów. Służy on do znalezienia wektora o maksymalnej ilości 1, czyli maksymalnej liczby kontenerów które możemy umieścić w magazynie.

Działanie algorytmu A*:

Algorytm A* rozpoczynamy od węzła startowego złożonego z samych '?'. Następnie tworzone są 2 listy: open_list oraz closed_list, służące do przechowywania węzłów oczekujących na odwiedzenie oraz węzłów odwiedzonych. Węzeł startowy dodajemy do listy open_list.

Następnie poszukiwany jest węzeł o największej wartości funkcji f – suma funkcji zysku oraz funkcji heurystycznej. Wybrany węzeł ustawiamy jak aktualny, a następnie przenosimy go do lisy closed_list.

Kolejnym krokiem jest znalezienie wszystkich sąsiadów, czyli wszystkie możliwe wariancję wybrania kolejnego kontenera (ustawienie nowego kontenera -> wartość '?' na '1'). Dla każdego sąsiada dokonujemy sprawdzenia, czy dany sąsiad znajduje się już w liście odwiedzonych węzłów (closed_list) lub w liście węzłów oczekujących na sprawdzenie (open_list). Jeśli sąsiad znajduje się w open_list lub closed_list, przechodzimy do kolejnego sąsiada, jeśli nie, obliczamy wartości funkcji *g h* i *f* oraz sprawdzamy pojemność magazynu. Funkcja *runHeuristics* (sprawdzenie pojemności magazynu) służy do sprawdzenia, czy dane kontenery nie przekroczą pojemnościowo magazynu. Nie sprawdza ona jednak czy dane kontenery możemy ustawić w taki sposób, aby mieściły się w magazynie. Jeśli pojemność magazynu nie została przekroczona, możemy dodać nasz węzeł do listy open_list. Gdy kontenery przekraczają pojemność magazynu, ustawiamy nowo dodany kontener na '0', co oznacza, że dany kontener nie zostanie umieszczony w magazynie. Po sprawdzeniu wszystkich sąsiadów przechodzimy do wybrania z listy open_list nowego aktualnego węzła. Algorytm wykonujemy do czasu, aż wszystkie węzły z open_list zostaną odwiedzone.

Algorytm zwraca listę closed_list.

Działanie algorytmu ewolucji różnicowej:

1. Inicjacja populacji bazowej o zadanym rozmiarze.
2. Obliczenie funkcji celu dla wszystkich osobników z populacji bazowej.
3. Dopóki nie przekroczono zadanej maksymalnej liczby iteracji lub nie znaleziono minimum globalnego (wszystkie zadane kontenery rozłożone zgodnie z ograniczeniami).

1) Dla każdego osobnika w populacji

A) Wybór trzech kandydatów do mutacji

B) Mutacja

C) Zaokrąglenie pozycji uzyskanych podczas mutacji i przesunięcie ich do przestrzeni przeszukiwań (w tym wypadku ograniczonej rozmiarami magazynu).

D) Krzyżowanie zmutowanego osobnika z oryginalnym osobnikiem

E) Obliczenie wartości funkcji celu dla obecnego i nowego osobnika

F) Zamiana osobnika w populacji na nowy, o ile wartość funkcji celu jest wyższa.

2) Oblicz wartości funkcji celu dla nowo utworzonej populacji

4. Zwróć najlepszego osobnika i informację o tym, czy rozkład spełnia ograniczenia.

Działanie programu:

Głównym algorytmem w rozwiązaniu byłby algorytm A* do maksymalizacji liczby kontenerów. Natomiast do podjęcia decyzji o miejscu umieszczenia kontenera służy w algorytmie A* heurystyka, która zwraca twierdzącą odpowiedź na pytanie, czy można umieścić dane kontenery w magazynie, jeśli nie jest przekroczona pojemność magazynu. Natomiast po zakończeniu działania algorytmu A* do rozmieszczenia kontenerów służy algorytm ewolucji różnicowej. Iteruje on się po zwróconych węzłach w kolejności nierosnącej liczby umieszczanych w magazynie kontenerów. Dzięki zastosowanej heurystyce w algorytmie A* mamy pewność, że w pierwszej kolejności rozpatrzymy te węzły, w których jest trochę za dużo kontenerów, aby spełnić ograniczenia lub dokładnie tyle, aby udało się je rozmieścić. Iteracja trwa do pierwszego poprawnego rozmieszczenia jak największej liczby kontenerów w magazynie.

Do reprezentacji kontenerów i magazynu wykorzystujemy instancje klasy Tensor, które przechowują informacje o położeniu w przestrzeni (współrzędne wierzchołka najbliżej punktu (0, 0, 0) w kartezjańskim układzie współrzędnych) oraz wymiarach w poszczególnych osiach. Metody klasy pozwalają obliczyć, w ilu jednostkach przestrzennych jeden tensor nachodzi na drugi (kontener na kontener), ile jednostek przestrzennych jednego tensora wystaje poza drugi (kontenera poza magazyn), losowo permutować dwa wymiary (obrócić kontener) oraz stwierdzić, czy dany tensor zawiera zadany punkt przestrzenny (używane do minimalizacji odległości tensora (kontenera) od tensora poniżej lub podłogi magazynu (z=0).

Warto zauważyć, że uwzględnianie ograniczeń 2. i 3. odbywa się w algorytmie ewolucji różnicowej w funkcji celu poprzez minimalizację odległości tensora (kontenera) od tensora poniżej lub podłogi magazynu (z=0).

<u>Wizualizacja rozwiązania</u>

Do wizualizacji rozwiązania została wykorzystana biblioteka:

matplotlib

Na podstawie wysokości, szerokości i długości kontenera oraz jego położenia zostaje wykreślony kontener w przestrzeni (magazynie).

Wykres przyjmuje wartości na osiach x, y, z jako szerokość, wysokość oraz długość magazynu podanego przez użytkownika.

Różnice między wstępnymi założeniami a rozwiązaniem końcowym:

W dokumentacji wstępnej zakładaliśmy wykorzystanie dwóch algorytmów A*. Algorytm A* do maksymalizacji liczby kontenerów został zaimplementowany według wstępnych założeń. Algorytm A* do rozkładu kontenerów w magazynie został zastąpiony algorytmem ewolucji różnicowej. Rozwiązanie tego problemu za pomocą algorytmu A* byłoby w rzeczywistości zakamuflowanym użyciem algorytmu przeszukiwania siłowego.

Opis przeprowadzonych eksperymentów numerycznych

Do sprawdzenia poprawności działania programu używamy przeglądu zupełnego (bruteforce – metoda siłowa). Dla każdej permutacji kolejności umieszczania kontenerów w magazynie, próbujemy umieszczać je po kolei w każdy możliwy sposób. Jeśli otrzymany wynik (stanowiący liczbę kontenerów umieszczonych w magazynie o maksymalnej pojemności kontenerów nie przekraczającej pojemność magazynu) nie będzie równy (lub zbliżony) wynikowi działania naszego programu, będzie to oznaczać jego niepoprawne działanie.

Wyniki eksperymentów numerycznych

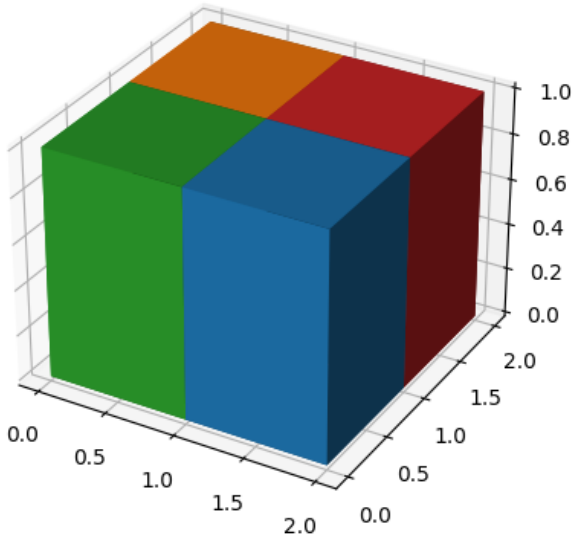
Do wszystkich zaprezentowanych eksperymentów zostało użyte ziarno równe 10 w generatorze liczb pseudolosowych używanym w algorytmie ewolucji różnicowej.

Parametry algorytmu ewolucyjnego:

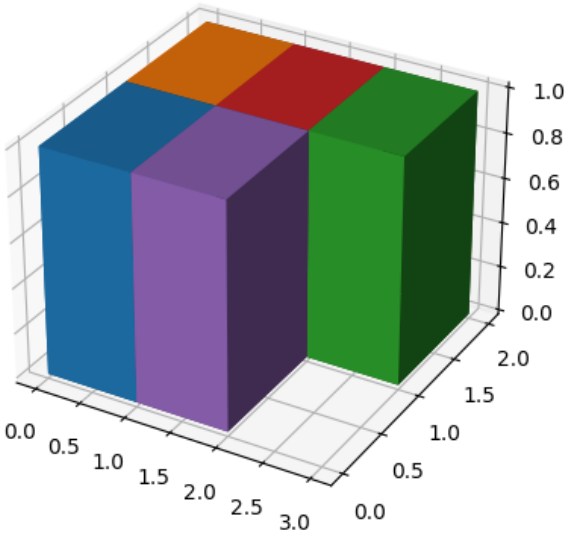
1. Rozmiar populacji: 50
2. Maksymalna liczba iteracji: 500
3. F = 0,5
4. CR = 0,7
5. P = 0,5

Wymiary magazynu	Liczba kontenerów	Wymiary kontenerów	Liczba kontenerów po rozłożeniu - bruteforce	Liczba kontenerów po rozłożeniu - A* + DE	Nr wizualizacji – uzyskanej przy użyciu A* + DE
2x2x1	4	1x1x1 (wszystkie)	4	4	1
3x2x1	5	1x1x1 (wszystkie)	5	5	2
2x2x2	3	2x2x2 (jeden) 1x1x1 (dwa)	2	2	3
5x5x5	4	3x3x3 (dwa) 1x1x1 (dwa)	3	2	4
1x5x1	6	2x2x2 (trzy)	3	3	5

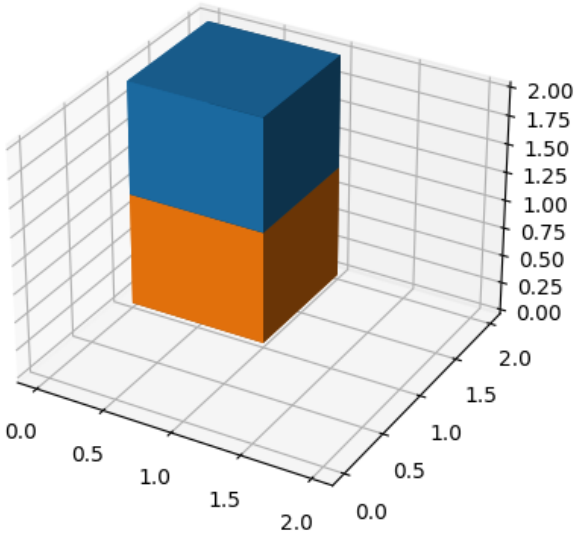
		1x2x1(jeden) 2x1x1(jeden) 1x1x1(jeden)			
5x5x5	5	5x5x5 (jeden) 4x4x4 (jeden) 3x3x3 (jeden) 2x2x2 (jeden) 1x1x1 (jeden)	3	1	6



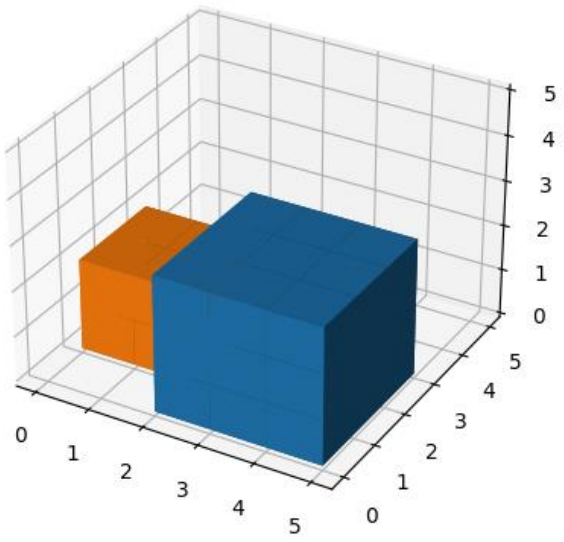
Wizualizacja nr 1



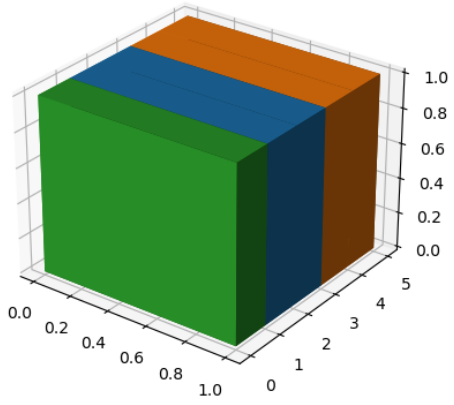
Wizualizacja nr 2



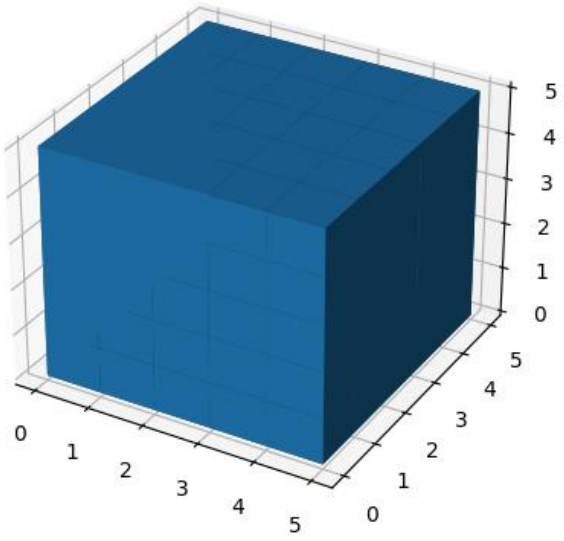
Wizualizacja nr 3



Wizualizacja nr 4



Wizualizacja nr 5



Wizualizacja nr 6

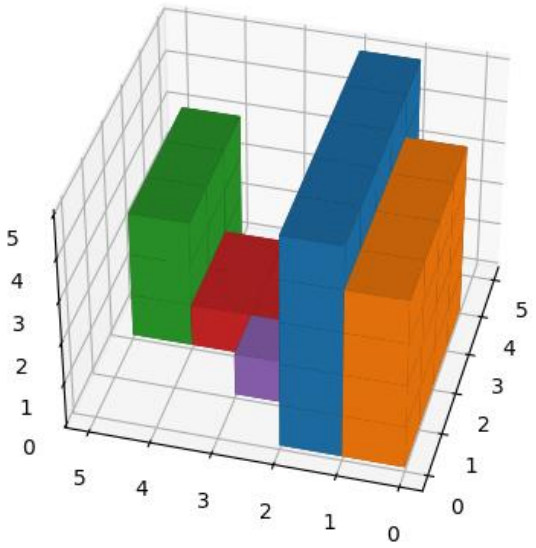
Dyskusja wyników

Przeprowadzone eksperymenty numeryczne pozwalają stwierdzić, że w większości przypadków nasz program z algorytmem A* oraz DE działa poprawnie. Rozmieszcza albo taką samą liczbę kontenerów jak bruteforce (eksperymenty z wizualizacjami nr 1, 2, 3, 5) albo zbliżoną (eksperyment z wizualizacją nr 4). W nielicznych przypadkach (eksperyment z wizualizacją nr 6) wartości te różnią się mocno od siebie. Wynika to z działania algorytmu DE – pomimo wskazania kontenerów do rozłożenia przez A* nie udało mu się poprawnie rozłożyć większej liczby kontenerów w magazynie niż 1.

Przykłady wizualizacji z użyciem A* i DE

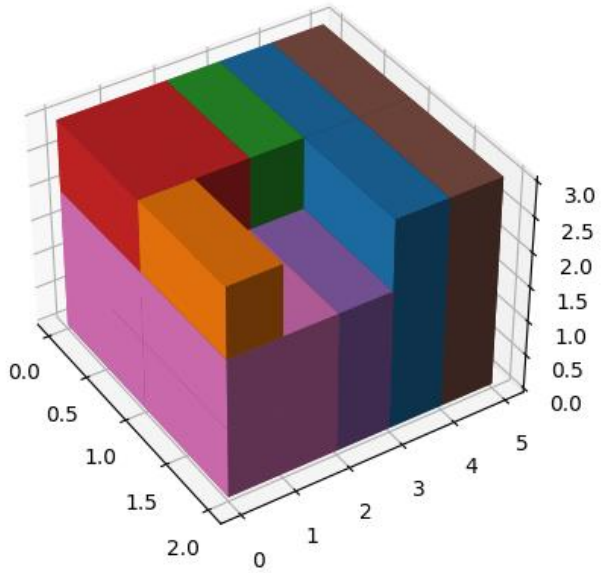
1. Magazyn: 5x5x5

Kontenery: 5x5x1, 4x4x1, 3x3x1, 2x2x1, 1x1x1



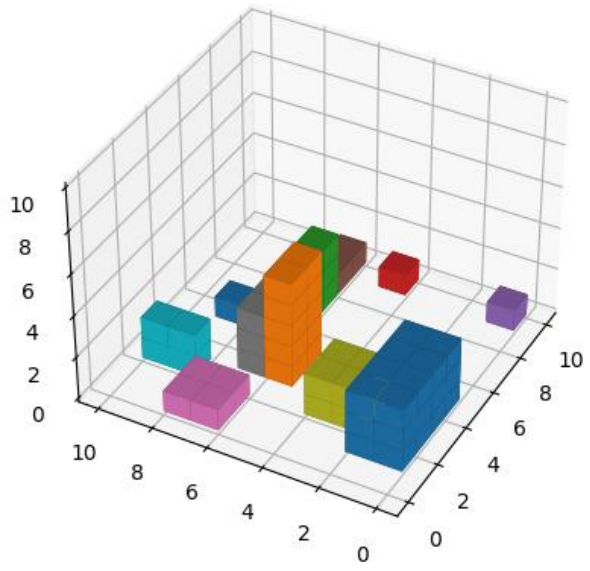
2. Magazyn: 2x5x3

Kontenery: 2x4x3, 7x4x5, 2x1x3, 1x1x1, 1x1x1, 1x1x2, 1x2x2, 2x3x1, 2x2x2, 2x1x2, 3x1x1



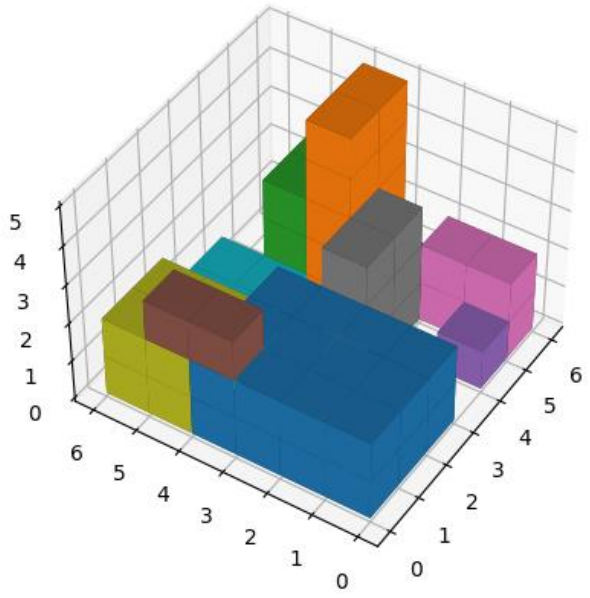
3. Magazyn: 10x10x10

   Kontenery: 2x4x3, 7x4x5, 2x1x3, 1x1x1, 1x1x1, 1x1x2, 1x2x2, 2x3x1, 2x2x2, 2x1x2, 3x1x1



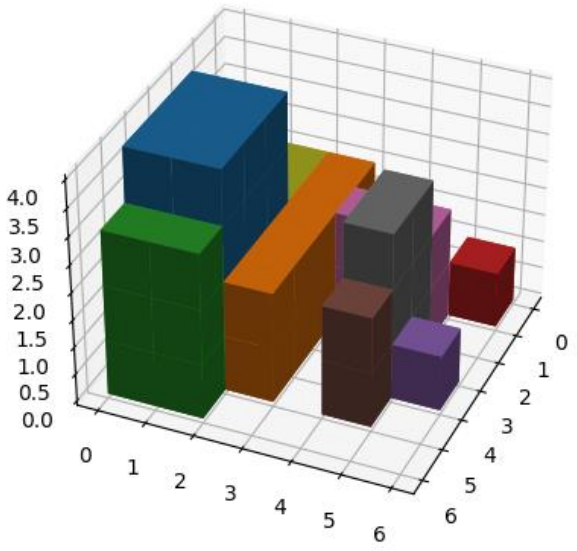
4. Magazyn: 6x6x5

   Kontenery: 2x4x3, 7x4x5, 2x1x3, 1x1x1, 1x1x1, 1x1x2, 1x2x2, 2x3x1, 2x2x2, 2x1x2, 3x1x1



5. Magazyn: 6x6x4

   Kontenery: 2x4x3, 7x4x5, 2x1x3, 1x1x1, 1x1x1, 1x1x2, 1x2x2, 2x3x1, 2x2x2, 2x1x2, 3x1x1



Technologie

1. Język programowania: Python (w wersji 3.9)
2. Biblioteki: `matplotlib, numpy, random, copy`

Zawartość projektu:

1. a_star.py - implementacja algorytmu A*
2. diff_evolution.py - implementacja algorytmu ewolucji różnicowej i klasy Tensor
3. get_input.py - główny program ,,dla użytkownika”
4. experiments.py - program do przeprowadzenia eksperymentów z rozmieszczeniem przy użyciu algorytmu A* i DE ze stałym ziarnem generatora liczb pseudolowych
5. ordering_functions.py - funkcje łączące rozmieszczanie przy użyciu obu algorytmów
6. bruteforce.py - implementacja algorytmu przeszukiwania wyczerpującego do porównania wyników uzyskanych przy użyciu algorytmu A* i DE.