Dokumentacja końcowa projektu POP

Maksymalne upakowanie kontenerów w magazynie 3D z wykorzystaniem algorytmu ewolucyjnego

Autorzy: Jakub Antas, Jerzy Muszyński

22 stycznia 2026

1 Opis problemu

Celem projektu jest rozwiązanie wariantu problemu **3D Bin Packing**, polegającego na rozmieszczeniu zbioru prostopadłościennych kontenerów o zadanych wymiarach wewnątrz pojedynczego magazynu o skończonej pojemności, w taki sposób, aby maksymalizować wykorzystanie jego objętości.

Problem ten należy do klasy problemów NP-trudnych, co oznacza, że dla większych instancji nie istnieją znane algorytmy deterministyczne gwarantujące znalezienie rozwiązania optymalnego w czasie wielomianowym. Z tego względu w projekcie zastosowano algorytm ewolucyjny, który pozwala uzyskiwać dobre jakościowo rozwiązania w rozsądnym czasie obliczeniowym.

2 Dane wejściowe

Danymi wejściowymi są:

- wymiary magazynu (W_x, W_y, W_z),
- lista kontenerów, z których każdy opisany jest trójką wymiarów (l, w, h).

Dane mogą być wczytywane z pliku CSV, co umożliwia łatwe testowanie różnych instancji problemu oraz zapewnia powtarzalność eksperymentów.

3 Reprezentacja rozwiązania

Jedno rozwiązanie (osobnik algorytmu ewolucyjnego) reprezentowane jest jako lista kontenerów. Każdy kontener opisany jest przez:

$$(x, y, z, dx, dy, dz, inserted)$$

gdzie:

- (x, y, z) – współrzędne położenia kontenera w magazynie,
- (dx, dy, dz) – wymiary kontenera po uwzględnieniu rotacji,
- $inserted \in \{0, 1\}$ – informacja, czy dany kontener został umieszczony w magazynie.

Rotacje kontenerów realizowane są poprzez permutację wymiarów (l, w, h). Kontenery oznaczone jako niewstawione nie są brane pod uwagę podczas obliczania funkcji celu ani sprawdzania kolizji.

4 Warunki poprawności rozwiązania

Rozwiązanie uznawane jest za poprawne, jeśli spełnia wszystkie poniższe warunki:

1. Każdy wstawiony kontener mieści się w granicach magazynu:

$$0 \leq x,\ x + dx \leq W_x, \quad 0 \leq y,\ y + dy \leq W_y, \quad 0 \leq z,\ z + dz \leq W_z$$

2. Żadne dwa wstawione kontenery nie nachodzą na siebie w przestrzeni trójwymiarowej.
3. Każdy kontener jest podparty – tzn. znajduje się na podłodze magazynu ($z = 0$) lub spoczywa bezpośrednio na innym kontenerze.

5 Funkcja celu

W projekcie zastosowano trzy warianty funkcji celu:

- **strict** – rozwiązania niepoprawne otrzymują wartość 0,
- **partial** – akceptowane są tylko poprawne fragmenty rozwiązania,
- **penalized** – rozwiązania niepoprawne są karane proporcjonalnie do liczby naruszeń.

Podstawową miarą jakości rozwiązania jest suma objętości wszystkich poprawnie umieszczonych kontenerów.

6 Algorytm ewolucyjny

Zastosowany algorytm ewolucyjny składa się z następujących etapów:

1. Inicjalizacja populacji (strategia konstrukcyjna lub losowa),
2. Ocena osobników przy użyciu funkcji celu,
3. Selekcja (turniejowa lub progowa),
4. Krzyżowanie jednolite,
5. Mutacja typu *Ruin and Recreate*, polegająca na częściowym usunięciu kontenerów i ich ponownym wstawieniu,
6. Elitaryzm – zachowanie najlepszych osobników,
7. Kryterium stopu oparte na liczbie generacji lub braku poprawy.

7 Eksperymenty numeryczne

Przeprowadzono serię eksperymentów porównujących algorytm ewolucyjny z losowym przeszukiwaniem przestrzeni rozwiązań (Random Search).

Dla zapewnienia powtarzalności wszystkie eksperymenty wykonywano z ustalonymi ziarnami generatora liczb losowych (seeds).

7.1 Eksperyment 1 – Tuning hiperparametrów

Pierwszy eksperyment miał charakter systematycznego strojenia hiperparametrów operatorów algorytmu ewolucyjnego metodą grid search (faza A). Dla ustalonej instancji problemu (zdefiniowane pudełka oraz magazyn) uruchomiono algorytm GA wielokrotnie, testując wszystkie kombinacje wartości pięciu kluczowych parametrów:

- `ratio_to_remove` (intensywność „ruin-and-recreate”),
- `p_mut_move` (prawdopodobieństwo dużej mutacji/przepakowania),
- `p_mut_resupport` (resupport),
- `init_constructive_ratio` (udział inicjalizacji konstruktywnej),
- `p_crossover` (krzyżowanie).

Pozostałe elementy budżetu obliczeniowego i konfiguracji (m.in. `pop_size=300`, `generations=300`, selekcja turniejowa, elitism, stałe mutacje rotacji/obecności) utrzymano stałe, aby izolować wpływ badanych operatorów na jakość rozwiązania.

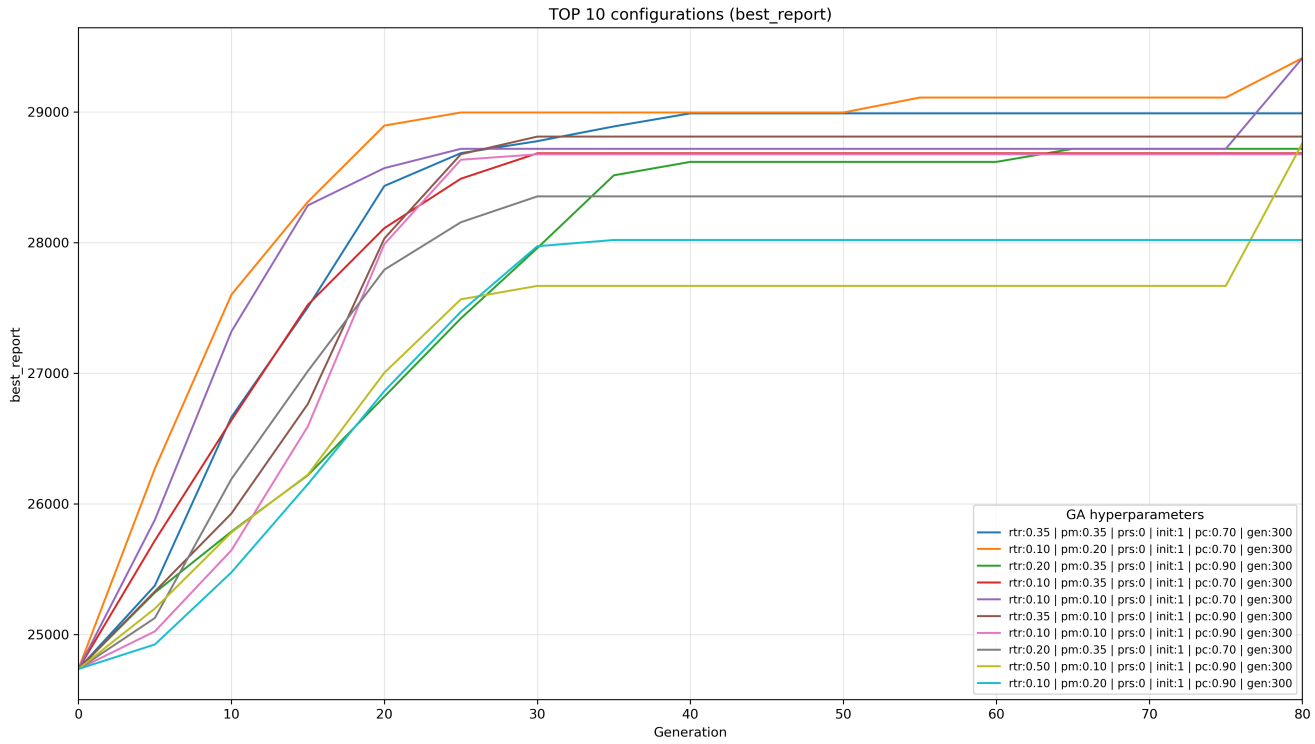
Wynikiem eksperymentu jest zestaw logów i podsumowań umożliwiających analizę porównawczą konfiguracji:

(i) pliki `runs/convergence/A/*.csv` zawierające przebiegi konwergencji (m.in. `best_report`, `avg_report`, `feasible_rate`) w funkcji generacji dla każdej konfiguracji i ziarna losowego,

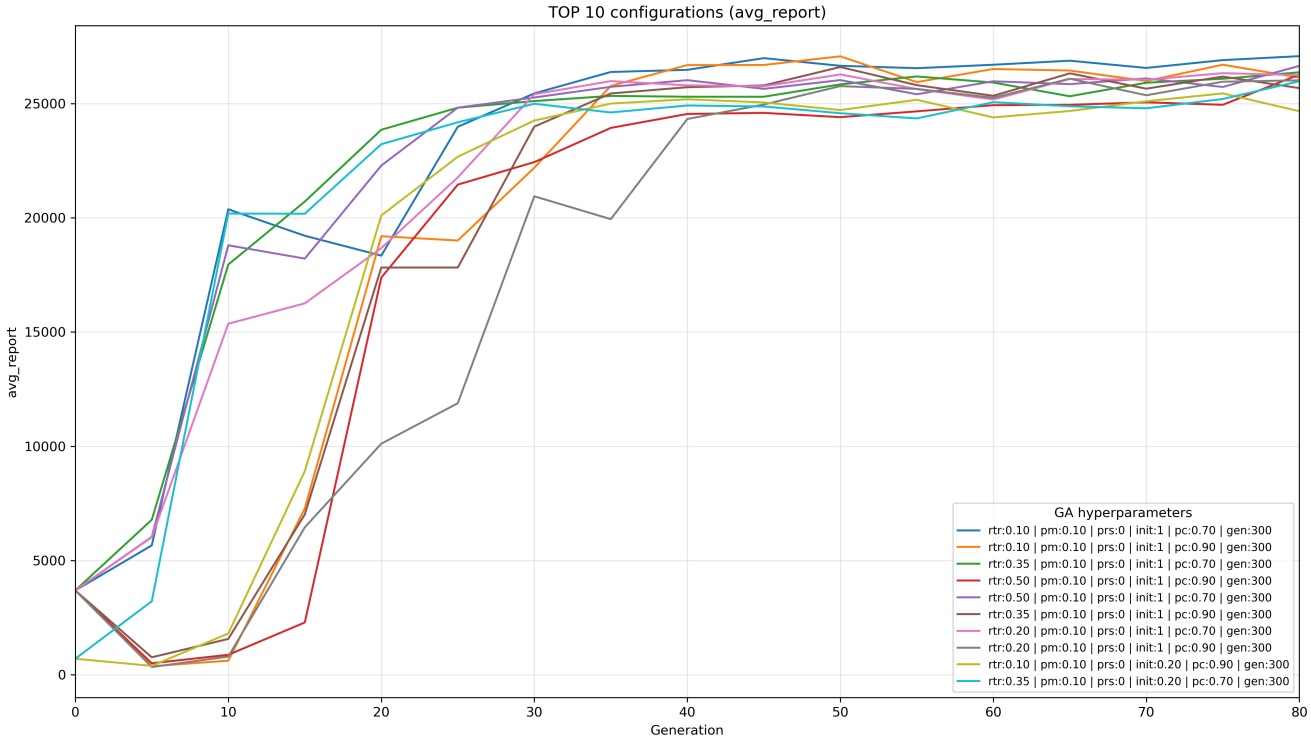
(ii) pliki `runs/summary.csv` / `runs/summary_A.csv` zawierające zagregowane metryki końcowe (np. najlepsza wartość funkcji celu, wykorzystanie objętości, czas wykonania) wraz z pełnym opisem hiperparametrów.

7.1.1 Analiza wykresów zbieżności

Poniższe wykresy prezentują porównanie najlepszych (TOP 10) oraz najgorszych (BOTTOM 10) konfiguracji hiperparametrów. Analiza ta pozwala zrozumieć, jak dobór parametrów wpływa na zdolność algorytmu do poprawy wyniku w czasie.

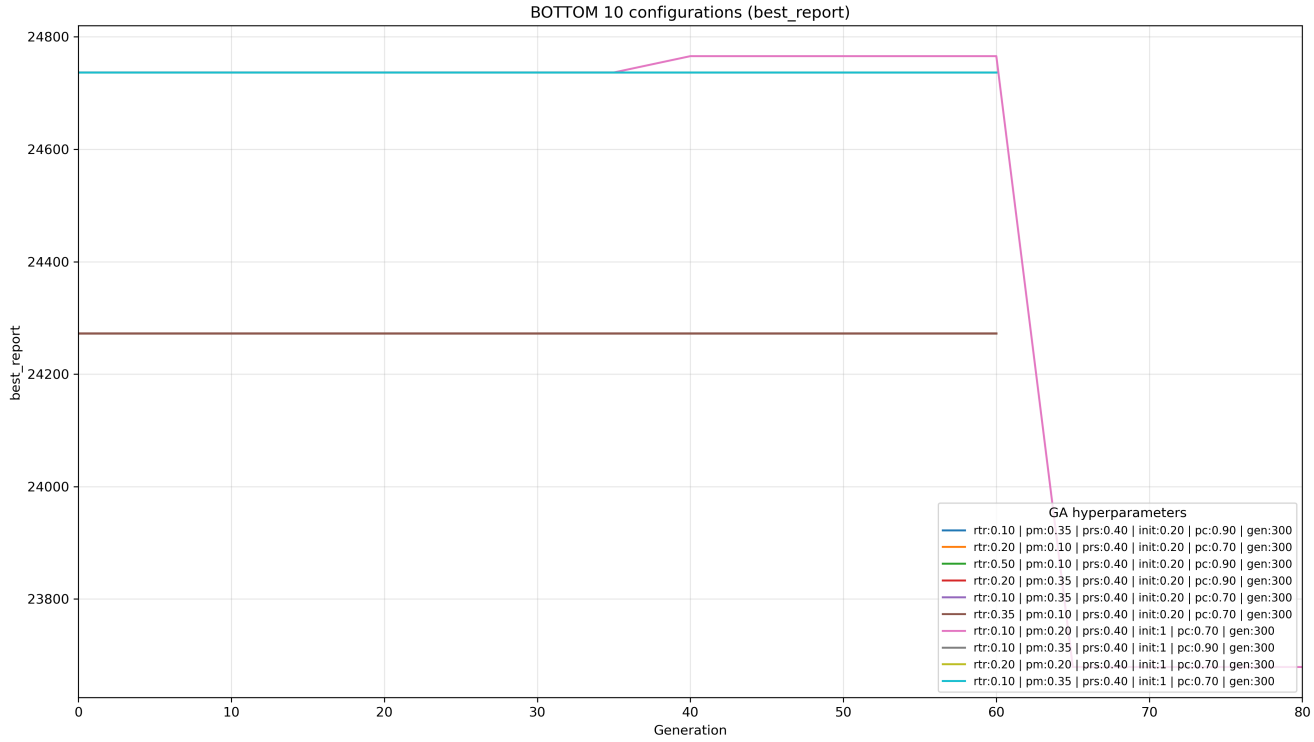


Rysunek 1: **TOP 10 – Najlepszy osobnik.** Wykres pokazuje, jak rośnie jakość najlepszego rozwiązania w kolejnych generacjach dla najlepiej dobranych parametrów. Widać wyraźny, szybki wzrost na początku (faza eksploracji) i stabilizację na wysokim poziomie w późniejszej fazie.

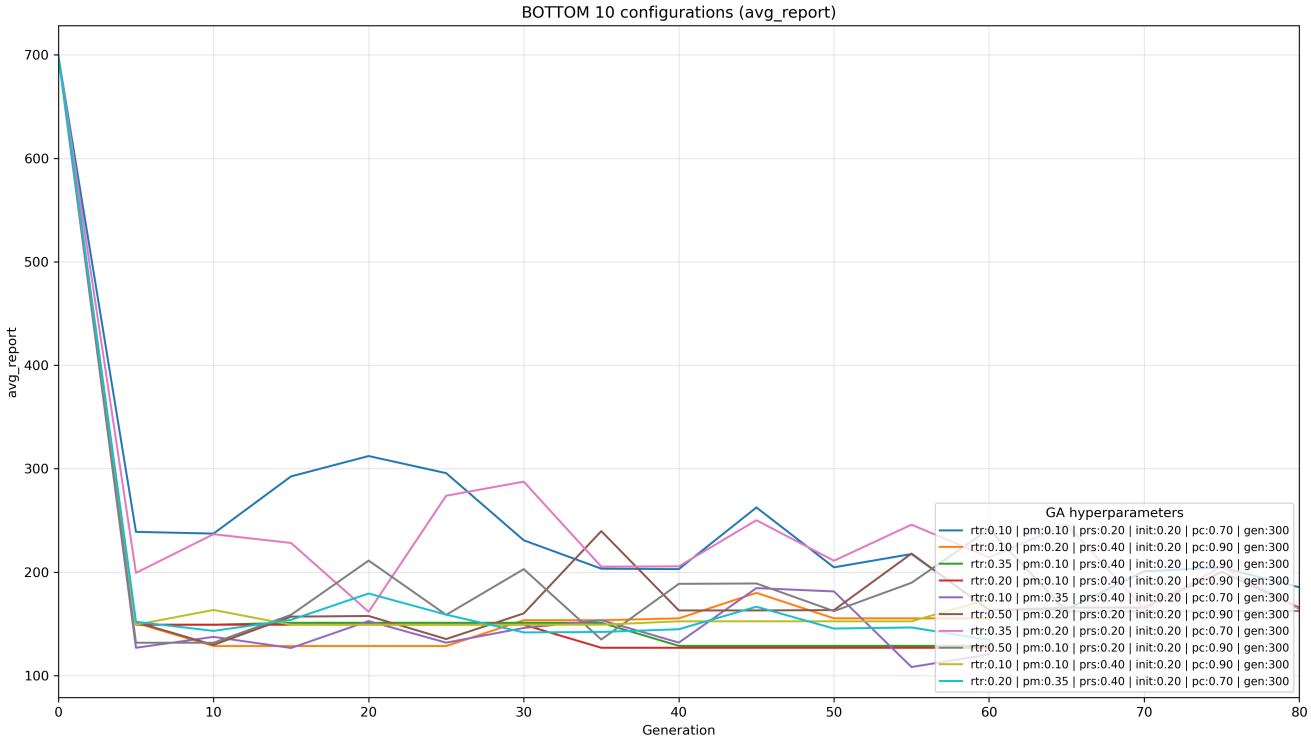


Rysunek 2: **TOP 10 – Średnia populacji.** Wykres średniej jakości całej populacji dla najlepszych konfiguracji. Rosnąca średnia świadczy o tym, że algorytm skutecznie eliminuje słabe rozwiązania, a cała populacja „podąża” za liderami.

W przeciwieństwie do powyższych, wykresy dla najgorszych konfiguracji (poniżej) pokazują stagnację.



Rysunek 3: **BOTTOM 10 – Najlepszy osobnik.** Dla źle dobranych parametrów algorytm nie potrafi znaleźć lepszego rozwiązania. Linie są płaskie, co oznacza, że algorytm utknął w minimum lokalnym lub parametry mutacji są zbyt destrukcyjne.



Rysunek 4: **BOTTOM 10 – Średnia populacji.** Wykres pokazuje chaos lub bardzo niską jakość populacji. W niektórych przypadkach średnia spada (widać gwałtowny zjazd w dół na początku), co sugeruje, że algorytm generuje bardzo dużo niepoprawnych rozwiązań, które są karane przez funkcję celu.

7.2 Eksperyment 2 – Porównanie GA z RS

Drugim eksperymentem było porównanie algorytmu ewolucyjnego (GA) z metodą bazową Random Search (RS). W ramach tego punktu analizujemy zarówno przebieg zbieżności algorytmu w kolejnych generacjach, jak i końcową jakość rozwiązań w porównaniu do metody losowej.

7.2.1 Porównanie GA z RS na różnych magazynach

Przeprowadziliśmy tu ewaluacje najlepszych konfiguracji algorytmu ewolucyjnego (GA), wyłonionych w pierwszym etapie strojenia parametrów, w porównaniu z metodą bazową Random Search (RS). Dziesięć najwyżej ocenionych konfiguracji GA zostało przetestowanych na dwóch wariantach magazynu reprezentujących malejący poziom trudności ($20 \times 20 \times 20$, $35 \times 35 \times 35$). Dla każdej konfiguracji oraz metody bazowej przeprowadzono wiele uruchomień z różnymi ziarnami losowymi, a uzyskane wyniki zostały zagregowane w celu ograniczenia wpływu losowości.

Magazyn 35x35x35: W tym przypadku algorytm ewolucyjny osiągnął znaczącą przewagę. Średnie wypełnienie magazynu dla GA wyniosło ok. 67-68%, podczas gdy Random Search (RS) zatrzymał się na poziomie 54%.

Config ID	GA Fitness	RS Fitness	GA Wypełnienie	RS Wypełnienie
0f093ffa	29110	23408	67.9%	54.6%
9de0dfab	28989	23408	67.6%	54.6%
53886113	28846	23408	67.3%	54.6%
c491eb65	28811	23408	67.2%	54.6%
303e75bc	28738	23408	67.0%	54.6%
c8ca4b93	28717	23408	66.9%	54.6%
5efe19be	28717	23408	66.9%	54.6%
2fc14040	28717	23408	66.9%	54.6%
72a98dfd	28683	23408	66.8%	54.6%
d2595fe1	28675	23408	66.8%	54.6%

Tabela 1: Wyniki dla magazynu $35 \times 35 \times 35$. Porównanie najlepszych konfiguracji GA z Random Search.

Magazyn 20x20x20: Jest to instancja trudniejsza ("ciasna"), co widać po wynikach algorytmu losowego, który w wielu przypadkach nie był w stanie znaleźć żadnego poprawnego rozwiązania (Fitness = 0). Najlepsze konfiguracje GA potrafiły jednak osiągnąć wypełnienie na poziomie ponad 60%.

Config ID	GA Fitness	RS Fitness	GA Wypełnienie	RS Wypełnienie
53886113	5350	0	67%	0%
c8ca4b93	4988	0	62%	0%
9de0dfab	4969	0	62%	0%
0f093ffa	0	0	0%	0%

Tabela 2: Wyniki dla magazynu $20 \times 20 \times 20$. Widać wyraźną porażkę metody losowej (RS Fitness = 0) przy trudniejszych instancjach.

Config ID	Init Ratio	P_Crossover	P_Mut_Move	P_Mut_Resupport	Ratio_Remove
0f093ffa	1.0	0.7	0.20	0.0	0.10
9de0dfab	1.0	0.7	0.35	0.0	0.35
53886113	1.0	0.7	0.20	0.0	0.50
c491eb65	1.0	0.9	0.10	0.0	0.35
303e75bc	1.0	0.7	0.10	0.0	0.20
c8ca4b93	1.0	0.7	0.20	0.0	0.35
5efe19be	1.0	0.9	0.35	0.0	0.20
2fc14040	1.0	0.7	0.10	0.0	0.10
72a98dfd	1.0	0.7	0.35	0.0	0.10
d2595fe1	1.0	0.9	0.10	0.0	0.10

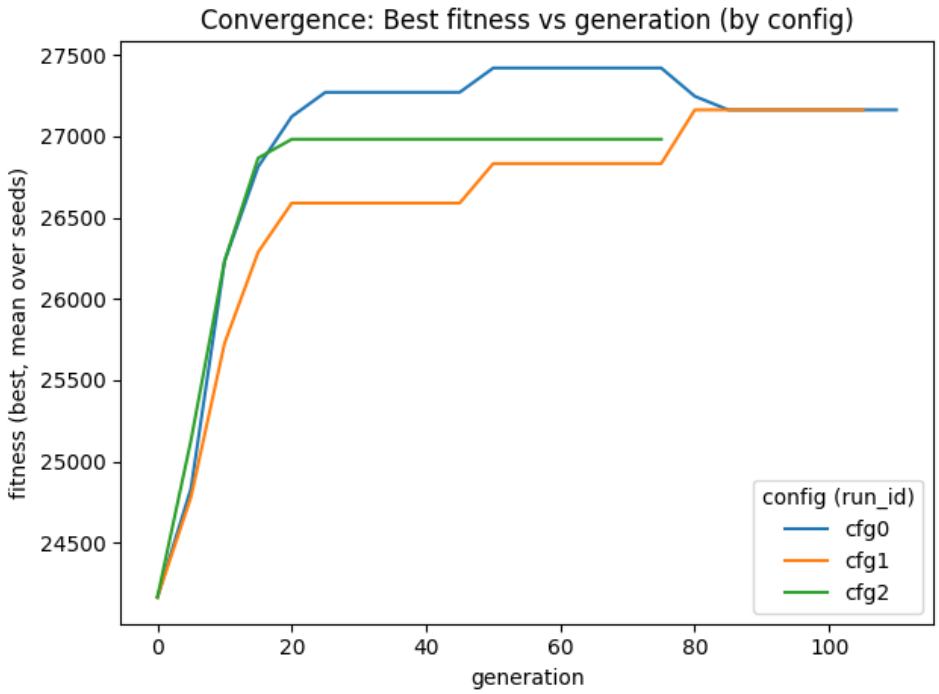
Tabela 3: Zestawienie parametrów dla wymienionych wyżej konfiguracji algorytmu ewolucyjnego

7.2.2 Porównanie 3 najlepszych konfiguracji GA z RS (średnia po 3 seedach)

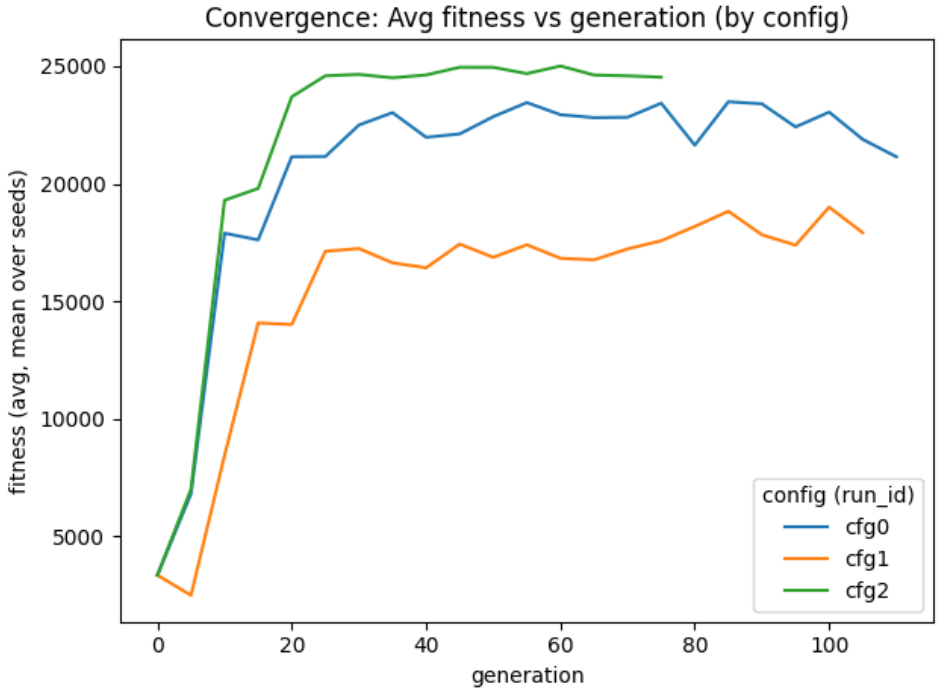
W tym podpunkcie wybrano trzy najlepsze konfiguracje hiperparametrów algorytmu GA (oznaczone dalej jako cfg0, cfg1, cfg2) i porównano je z metodą Random Search. Każdą konfigurację uruchomiono dla trzech różnych ziaren generatora liczb losowych (seeds). Następnie wyniki zostały uśrednione, aby ograniczyć wpływ losowości i uzyskać bardziej stabilny obraz działania algorytmów.

Analizowano następujące charakterystyki:

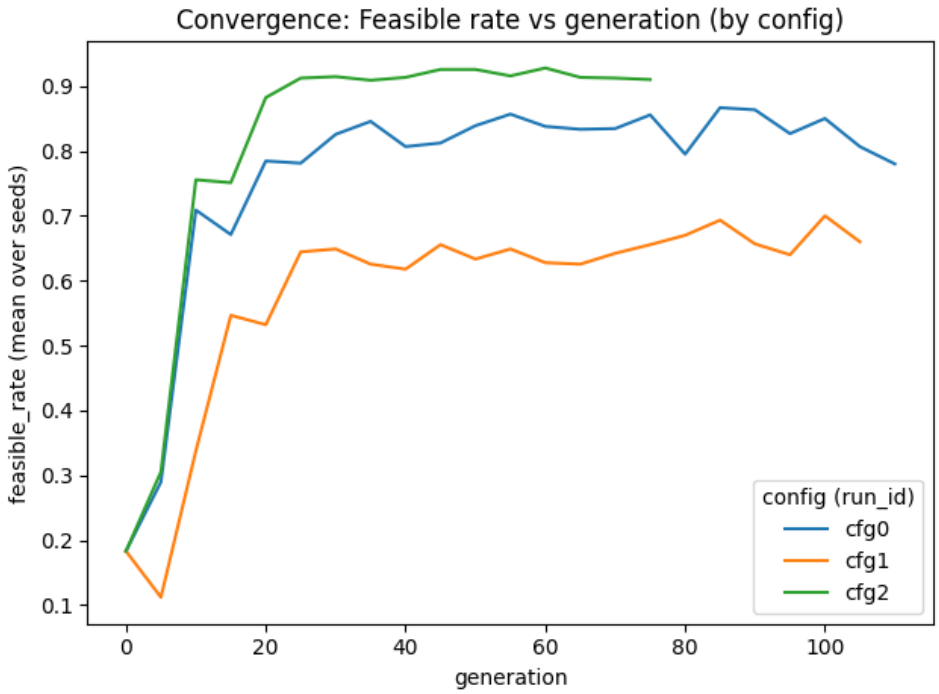
- najlepsza wartość funkcji celu w generacji (fitness best vs generation),
- średnia wartość funkcji celu w populacji (fitness avg vs generation),
- odsetek rozwiązań poprawnych w populacji (feasible rate vs generation),
- porównanie końcowego wykorzystania magazynu GA vs RS (best utilization).



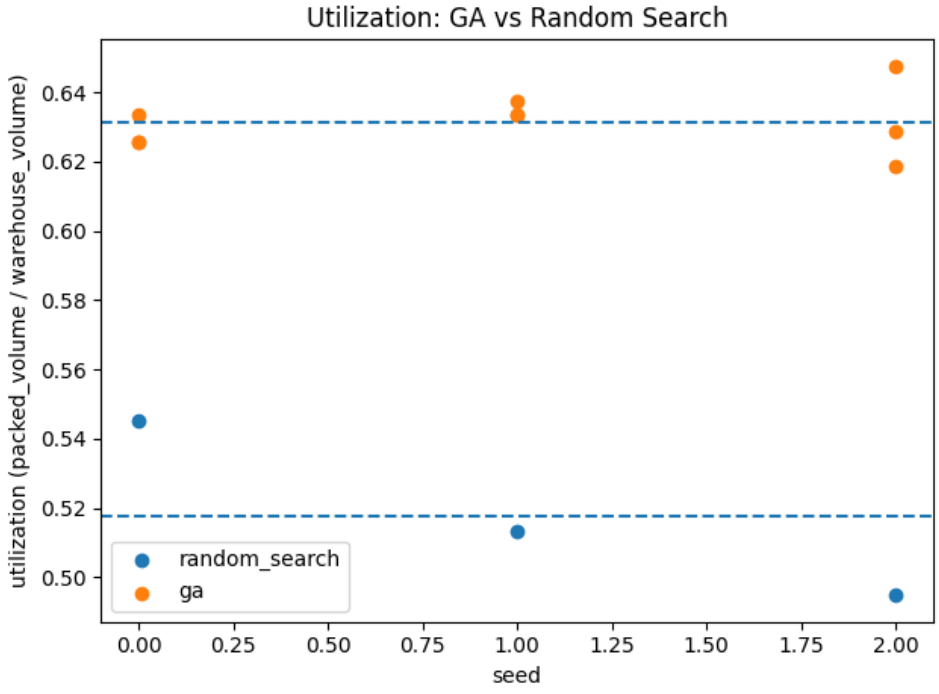
Rysunek 5: Najlepsza wartość funkcji celu w kolejnych generacjach dla cfg0–cfg2 (średnia po 3 ziarnach).



Rysunek 6: Średnia wartość funkcji celu w populacji dla cfg0–cfg2 (średnia po 3 ziarnach).



Rysunek 7: Odsetek rozwiązań poprawnych w populacji (feasible rate) dla cfg0–cfg2 (średnia po 3 ziarnach).



Rysunek 8: Porównanie GA i Random Search: wykorzystanie objętości magazynu (best utilization).

Na podstawie powyższych wykresów można stwierdzić, że GA systematycznie poprawia jakość rozwiązań w kolejnych generacjach (widoczny wzrost wartości best fitness). W porównaniu do Random Search algorytm ewolucyjny osiąga wyraźnie lepsze wykorzystanie objętości magazynu, jednak wymaga większego nakładu obliczeniowego.

Config ID	Init Ratio	P_Crossover	P_Mut_Move	P_Mut_Resupport	Ratio_Remove	Kolor funkcji na Rys. 1
cfg0	1.0	0.7	0.20	0.0	0.10	pomaranczowa
cfg1	1.0	0.7	0.35	0.0	0.35	ciemno niebieska
cfg2	1.0	0.7	0.10	0.0	0.10	fioletowa

Tabela 4: Zestawienie parametrów dla wymienionych wyżej konfiguracji algorytmu ewolucyjnego

8 Analiza wyników

Algorytm ewolucyjny we wszystkich testowanych instancjach osiągał istotnie lepsze wyniki niż losowe przeszukiwanie przestrzeni rozwiązań. Zaobserwowano również wpływ parametrów takich jak intensywność mutacji oraz wielkość populacji na szybkość zbieżności algorytmu.

Wykresy zbieżności pokazują systematyczny wzrost jakości rozwiązań w kolejnych generacjach oraz stabilizację procesu optymalizacji.

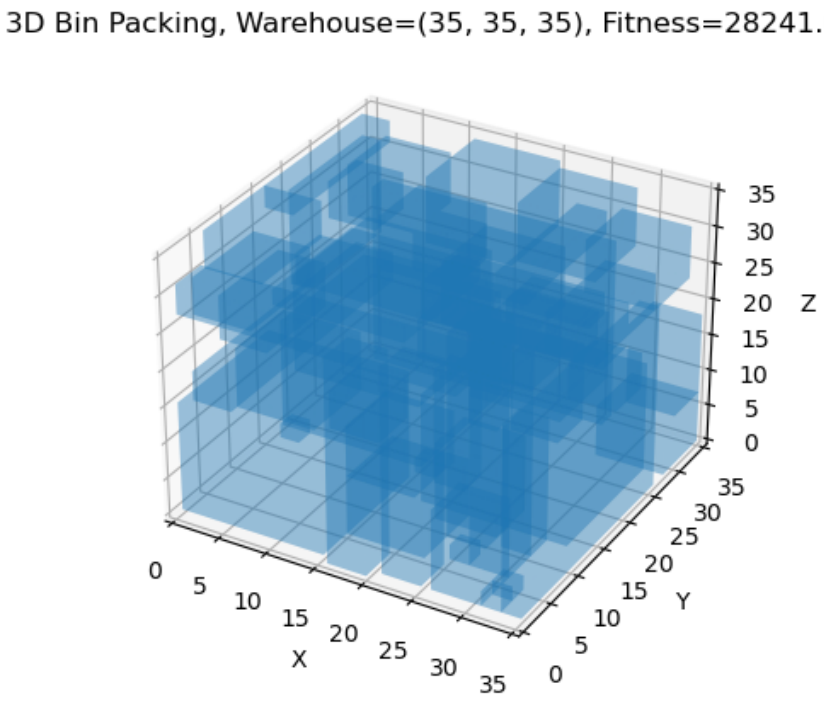
9 Wizualizacja rozwiązań

Liczby w tabelach to nie wszystko. Aby potwierdzić, że algorytm działa poprawnie w praktyce, zaimplementowano moduł wizualizacji 3D (przy użyciu biblioteki `matplotlib`). Pozwala on zobaczyć, jak dokładnie ułożone są kontenery wewnątrz magazynu.

Wizualizacja pełni dwie funkcje:

- **Weryfikacja poprawności** – pozwala szybko sprawdzić "na oko", czy pudełka nie lewitują w powietrzu i czy nie wchodzą jedno w drugie.
- **Ocena upakowania** – widać wyraźnie, gdzie powstają luki (marnowana przestrzeń) i jak algorytm radzi sobie z układaniem warstw.

Poniższy rysunek przedstawia najlepsze rozwiązanie znalezione dla dużego magazynu ($35 \times 35 \times 35$). Półprzezroczyste niebieskie bloki to umieszczone kontenery.



Rysunek 9: Wizualizacja 3D dla magazynu $35 \times 35 \times 35$. Algorytm osiągnął wynik (fitness) 28241, dla jednego z seedów tworząc zwarte bloki ładunku.

10 Różnice względem dokumentacji wstępnej

W porównaniu do dokumentacji wstępnej:

- dodano warunek podparcia kontenerów,
- wprowadzono zaawansowaną mutację typu Ruin and Recreate,
- zastosowano kilka wariantów funkcji celu,
- przeprowadzono kompleksowe eksperymenty porównawcze,

11 Podsumowanie

Projekt pokazuje, że algorytmy ewolucyjne stanowią skuteczne narzędzie do rozwiązywania złożonych problemów optymalizacyjnych, takich jak 3D Bin Packing. Uzyskane wyniki potwierdzają ich przewagę nad metodami losowymi oraz dużą elastyczność w dostosowaniu do dodatkowych ograniczeń problemu.