AMHE - Sprawozdanie końcowe

Bartosz Latosek,
Mateusz Krakowski

Kwiecień 2025

Spis treści

1 Temat

*Rozwiązać problem plecakowy dla danych skorelowanych i nieskorelowanych używając algorytmu **PBIL** (Population-Based Incremental Learning), porównując z wybraną metaheurystyką. Wymagana dokładną analiza statystyczna przedstawionych wyników. Zalecane konsultacje u prowadzącego przed przystąpieniem do pracy nad projektem.*

2 Opis problemu

Problem plecakowy (znany również jako problem złodzieja) jest klasycznym zagadnieniem optymalizacyjnym. W problemie tym mamy do czynienia ze zbiorem n przedmiotów, gdzie każdy i-ty przedmiot posiada:

- wartość $v_i \in R^+$
- ciężar $w_i \in R^+$

Problem polega na znalezieniu wektora decyzyjnego $x = (x_1, x_2, \ldots, x_n)$, gdzie $x_i \in \{0, 1\}$, który maksymalizuje łączną wartość zabranych przedmiotów, nie przekraczając pojemności plecaka $C \in R^+$:

$$\text{Max} \sum_{i=1}^{n} v_i x_i \tag{1}$$

przy ograniczeniu:

$$\sum_{i=1}^{n} w_i x_i \leq C \tag{2}$$

gdzie:

- $x_i = 1$ oznacza, że przedmiot i-ty został spakowany
- $x_i = 0$ oznacza, że przedmiot i-ty został pominięty

Powyższy opis dotyczy tzw. wariancji dyskretnej problemu plecakowego, w której decyzja o umieszczeniu przedmiotu w plecaku jest binarna - albo umieszczamy cały przedmiot, albo nie. W literaturze można również znaleźć ciągłą wersję tego problemu, w której dopuszczalne jest umieszczenie części przedmiotu w plecaku, ale na potrzeby niniejszego projektu i ograniczeń, jakie nakłada wykorzystanie algorytmu PBIL można założyć, że analizowany będzie wyłącznie dyskretny wariant tego problemu.

3 Cel projektu

Celem projektu jest porównanie statystyczne działania dwóch algorytmów rozwiązujących zdefiniowany problem plecakowy. Pierwszym z analizowanych algorytmów będzie narzucony odgórnie algorytm **PBIL** (opisany w dalszych rozdziałach). Jako, że jest to odmiana algorytmu ewolucyjnego, jako algorytm konkurencyjny wybrany został **klasyczny algorytm ewolucyjny** (również opisany w późniejszych rozdziałach).

4 Dane

W niniejszym projekcie zdecydowano stworzyć i wykorzystać własny generator przedmiotów do problemu plecakowego, co związane było z późniejszym ułatwieniem analizy wniosków i badań statystycznych. Wykorzystanie funkcji generatora umożliwia parametryzację wykorzystanych danych pod kątem:

- **Wielkości zbioru danych**, przy czym n oznacza liczbę przedmiotów:
  - Małe - $n \leq 10$
  - Średnie - $10 < n \leq 50$
  - Duże - $n > 50$
- **Skorelowania danych** pochodzących ze zbioru dostępnych przedmiotów:
  - **Skorelowane** - wartości v_i i w_i generowane z rozkładu normalnego o zadanych parametrach:
$$w_i \sim \mathcal{N}(\mu_w, \sigma_w^2), \quad v_i \sim \mathcal{N}(\mu_v, \sigma_v^2) \tag{3}$$
  - **Nieskorelowane** - generowanie niezależnych wartości v_i i w_i z rozkładu jednostajnego:
$$v_i \sim U(v_{\min}, v_{\max}), \quad w_i \sim U(w_{\min}, w_{\max}) \tag{4}$$

5 Opis algorytmów

5.1 PBIL - Population-Based Incremental Learning

Algorytm PBIL (Population-Based Incremental Learning) to połączenie algorytmów genetycznych z uczeniem prawdopodobieństwa. Operuje na wektorze prawdopodobieństwa P:

$$p_l(x) = (p_l(x_1), p_l(x_2), \ldots, p_l(x_n))$$

, gdzie $p_l(x_i)$ określa prawdopodobieństwo wystąpienia wartości 1 na $i - tej$ pozycji genomu w generacji l.

Początkowy genom populacji inicjalizowany jest z wartością $p_1(x) = 0.5$, co gwarantuje generację losowych osobników do momentu optymalizacji genomu. W każdej iteracji algorytmu generowane jest M osobników populacji (wszystkie oparte na głównym genomie prawdopodobieństwa), z których następnie wybierane jest N najlepszych rozwiązań. Następnie, najlepsze osobniki są wykorzystywane do aktualizacji genomu prawdopodobieństwa zgodnie z regułą inspirowaną *Hebbem*:

$$p_{l+1}(x) = (1 - \alpha)p_l(x) + \alpha \frac{1}{N} \sum_{K=1}^{N} x_{k:M}^l$$

, gdzie $\alpha \in (0, 1]$ to prędkość uczenia (*ang. learning rate)*.

Po każdej aktualizacji genomu prawdopodobieństwa, generowane są nowe osobniki populacji i cykl powtarza się do momentu osiągnięcia warunku stopu (np. przekroczenia maks. liczby epok). W celu podsumowania przebiegu algorytmu został stworzony niniejszy pseudokod:

Algorithm 1: Population-Based Incremental Learning (PBIL)

Initialize probability vector P with 0.5 for all positions
while $l <= MAX\_EPOCHS$ **do**
  Generate M individuals based on P:
  foreach *individual* **do**
    foreach *gene position* i **do**
      Set gene to 1 with probability $P[i]$, otherwise set to 0
    end
  end
  Evaluate fitness of all M individuals
  Select the N best individuals based on fitness
  foreach *gene position* i **do**
    Compute mean value of gene i in the N best individuals

$$\text{mean\_value} = \frac{1}{N} \sum_{K=1}^{N} x^{l}_{k:M}$$

    Update $P[i]$ using:

$$P[i] \leftarrow (1 - \alpha)P[i] + \alpha \cdot \text{mean\_value}$$

  end
end
Return optimized probability vector P

5.2 Klasyczny algorytm ewolucyjny

Klasyczny algorytm ewolucyjny jest dobrze znanym podejściem optymalizacyjnym, szeroko stosowanym w różnych dziedzinach. W związku z tym, że jego szczegóły są dość powszechne, w ramach przypomnienia został zamieszczony poniższy pseudokod:

Algorithm 2: Classical Evolutionary Algorithm

Initialize population of size M with random individuals
while $l \leq MAX\_EPOCHS$ **do**
  Evaluate fitness of all M individuals
  Select N best individuals based on fitness (elitism)
  Generate offspring through crossover: **foreach** *pair of selected parents* **do**
    Perform crossover to create new offspring
  end
  Apply mutation: **foreach** *individual in offspring* **do**
    foreach *gene position* i **do**
      With probability p_{mut}, flip the gene value
    end
  end
  Replace the population with selected N best solutions + generated offspring
end
Return best solution found in population

- W ramach **krzyżowania** najlepszych osobników wykorzystane zostanie krzyżowanie z wykorzystaniem punktu przecięcia (stanowiącego środek genomu), zgodnie z poniższym przykładem:

  Rodzic 1: 1011 | 0010

  Rodzic 2: 0110 | 1101

  Punkt podziału: 4-ty bit

  Potomek 1: 1011 | 1101

  Potomek 2: 0110 | 0010

- Mutacje stanowić będzie zamiana bitu na $i - tej$ pozycji genomu potomstwa.

  Osobnik: 10**1**10010

  Mutacja na pozycji i=3

  Zmutowany Osobnik: 10**0**10010

W ramach wykonywanych eksperymentów, algorytm uczył się przez zdefiniowaną liczbę iteracji, a do dalszego porównania wykorzystany został najlepszy, kiedykolwiek znaleziony osobnik i jego przypisana wartość funkcji celu *(ang. fitness)*.

5.3 Funkcja celu

Obydwa algorytmy korzystały ze wspólnej funkcji celu z naliczaniem kar na podstawie wartości i wagi wybranych przedmiotów. Funkcja celu penalizuje przekroczenie dopuszczalnej pojemności plecaka, stosując do tego dynamicznie wyliczany współczynnik kary. Takie podejście sprawia, że wartość kary przyznawana za przekroczenie dopuszczalnej pojemności plecaka rośnie proporcjonalnie do liczby przedmiotów w zbiorze danych oraz do średnich wartości przedmiotów w plecaku (również zależnych od wielkości zbioru przedmiotów, co zostanie opisane później). Funkcja zysku to po prostu wartość wszystkich wybranych przedmiotów w plecaku.

6 Opis funkcjonalny

Do wykonania projektu wykorzystany został język **Python3** wraz ze środowiskiem **jupyternotebook** wykorzystywanym do analizy wyników. Wykorzystane biblioteki wraz z użytymi wersjami dostępne są w pliku *requirements.txt*. Kod został częściowo przetestowany za pomocą biblioteki *pytest*.

Projekt podzielony został na 3 główne części, a cały kod źródłowy został dołączony do projektu. Całość została napisana przy użyciu zasad programowania obiektowego i najlepszych praktych stosowanych powszechnie w projektach realizowanych w języku *python*.

6.1 Obiekty Statyczne - *objects*

Wśród obiektów statycznych znajdują się klasy danych pozwalające na łatwe definiowanie parametrów. Podmoduł zawiera: *AlgoritmParams* - definiujący parametry dotyczące egzekucji algorytmów takie jak wielkość populacji, czy liczba epok, *ComparerParams* - czyli parametry służace porównywaniu algorytmów - liczba powtórzeń, parametry dotyczące wielkości zbiorów danych, *Distributions* - klasy opisujące parametry dystrybucji danych (*UniformDistribution* dla nieskorelowanych i *NormalDistribution* dla skorelowanych) oraz *GeneratorParams* - czyli parametry przekazywane bezpośrednio do generatora danych np. rodzaj dystrybucji.

6.2 Algorytmy - *algorithms*

Podmoduł ten zawiera implementacje klasycznego algorytmu ewolucyjnego oraz algorytmu PBIL, wraz z klasą bazową zawierającą elementy wspólne, takie jak funkcję celu.

6.3 Obiekty funkcjonalne - *src*

Wśród tych obiektów funkcjonalych znajdują się *AlgorithmsComparer* - zawierający logikę służącą porównywaniu algorytmów w zdefiniowanych warunkach, *DataGenerator* służący generowaniu danych, *Logger* - czyli obiekt zapisujący dane pochodzące z porównywania algorytmów, m.in. czas egzekucji, zajętość pamięci i szereg danych związanych z wartością funkcji celu najlepszych rozwiązań.

6.4 Pozostałe

W plikach projektowych znajduje się folder z testami jednostkowymi napisanymi przy użyciu biblioteki *pytest*. Plik *main.py* służy pozyskiwaniu danych z porównania algorytmów, które następnie analizowane są w notatniku *analysis.ipynb*.

7 Opis eksperymentów

Celem eksperymentu jest porównanie efektywności algorytmu **PBIL** i **klasycznego algorytmu ewolucyjnego** w kontekście problemu plecakowego. Badanie będzie obejmowało zarówno aspekty jakościowe (np. najlepsze rozwiązania), jak i ilościowe (np. czas obliczeń, wykorzystanie pamięci). W szczególności, analizowane będą:

- **Jakość rozwiązań**: średnia jakość rozwiązania w różnych iteracjach algorytmu, najlepszy oraz najgorszy wynik uzyskany w każdej z rund eksperymentalnych.
- **Czas obliczeń**: czas wykonania obydwu algorytmów w zależności od liczby przedmiotów w problemie plecakowym.
- **Zajętość pamięci**: zużycie pamięci w zależności od wielkości problemu.
- **Złożoność czasowa**: wykazanie złożoności problemu przez analizę wzrostu zużycia czasu i pamięci w zależności od wielkości zbiorów danych

7.1 Procedura eksperymentalna

1. **Inicjalizacja**:
   - Inicjalizacja obu algorytmów (PBIL oraz klasycznego algorytmu ewolucyjnego).
   - Ustawienie takich samych parametrów dla obu algorytmów (liczba epok, parametry zbioru danych...)
2. **Powtarzanie eksperymentu**:
   - Każdy eksperyment będzie powtarzany dla różnych rozmiarów problemu (mały, średni i duży).
   - Każdy eksperyment zostanie powtórzony T razy (np. 100 razy), aby uzyskać wiarygodne dane statystyczne.
3. **Zbieranie wyników**:
   - Dla każdego przebiegu algorytmu zebrane będą następujące dane:

- **Czas obliczeń** (w sekundach)
- **Wykorzystanie pamięci** (w MB)
- **Średnia jakość rozwiązania** (średnia wartość z najlepszych rozwiązań po każdej iteracji)
- **Najlepszy i najgorszy wynik** (najlepsze i najgorsze rozwiązanie uzyskane w danym eksperymencie)

7.2 Analiza statystyczna wyników

- **Średnia i odchylenie standardowe**: Obliczymy średnią i odchylenie standardowe dla jakości rozwiązań uzyskanych przez oba algorytmy. Średnia poda nam ogólną tendencję, a odchylenie standardowe pomoże ocenić zmienność wyników.

$$\text{Średnia} = \frac{1}{T}\sum_{i=1}^{T} x_i$$

$$\text{Odchylenie standardowe} = \sqrt{\frac{1}{T}\sum_{i=1}^{T}(x_i - \text{średnia})^2}$$

- **Test statystyczny - Test t-Studenta**: Porównanie średnich wyników algorytmów PBIL i klasycznego algorytmu ewolucyjnego. Test t-Studenta zostanie wykorzystany, aby ocenić, czy różnice w jakości rozwiązań są statystycznie istotne.
- **Analiza wyników**: Porównanie wyników obu algorytmów na różnych instancjach problemu plecakowego (różna liczba przedmiotów n). Obliczenie najlepszego i najgorszego wyniku dla obu algorytmów. Analiza różnicy w jakości rozwiązań między algorytmami.

7.3 Pomiar złożoności czasowej i pamięciowej

- **Złożoność czasowa**: Mierzymy czas obliczeń algorytmów na różnych instancjach problemu plecakowego o różnych rozmiarach (mały, średni i duży). Zbadamy, jak czas obliczeń rośnie w miarę wzrostu liczby przedmiotów. Oczekujemy, że złożoność czasowa obu algorytmów będzie rosła wykładniczo.
- **Zajętość pamięci**: Będziemy rejestrować użycie pamięci przez oba algorytmy w trakcie obliczeń, aby ocenić różnice w wymaganiach pamięciowych.

7.4 Inicjalizacja parametrów

Ze względu na dużą liczbę parametrów i hiperparametrów w obydwu algorytmach, zdecydowano się maksymalnie ograniczyć potencjalne różnice w wynikach poprzez zbliżenie do siebie warunków, w jakich działały testowane algorytmy.

7.4.1 Czynniki stałe

Wśród stałych czynników umieściliśmy m.in. szczegóły implementacyjne klasycznego algorytmu ewolucyjnego, tj. metodykę krzyżowania i mutowania osobników. Jako, że tematem projektu jest porównanie ze sobą algorytmu **PBIL** z **klasycznym ewolucyjnym**, dobór metod wewnętrznych ma marginalne znaczenie w wynikowym porównaniu a dodatkowa próba testowania kilku wariancji klasycznego algorytmu ewolucyjnego może wprowadzić nieścisłości w wynikach. W związku z powyższym, ostatecznie zaimplementowane zostały opisane w sekcji opisu algorytmów klasyczne

wariancje krzyżowania i mutacji dostosowane do dyskretnego problemu plecakowego. Uściślając - krzyżowanie występuje pomiędzy losowo wybranymi N najlepszymi osobnikami danej populacji do momentu uzyskania M osobników kolejnej populacji, a mutacja następuje **zawsze** i dotyczy jednego, losowo wybranego bitu genomu.

Dodatkowym stałym czynnikiem jest to, że dane algorytmy są porównywane zawsze na dwóch wariantach zbioru danych - **skorelowanym** i **nieskorelowanym**.

7.4.2 Czynniki wspólnie zależne

By maksymalnie ograniczyć różnice w warunkach algorytmów, niektóre z parametrów zostały uzależnione od siebie wzajemnie, tak by skalowały się automatycznie ale jednocześnie pozostawały w stałej relacji między sobą. Zdecydowano, że bazowym parametrem będzie **wielkość zbioru danych**. Zauważono, że z tą wartością można powiązać **parametry rozkładów** wykorzystywane przy generowaniu danych oraz **maksymalną pojemność plecaka**. Dla przykładu, przyjmując wielkość zbioru danych np. **20**, zakładamy, że przedmioty nieskorelowane będą przyjmowały wagi od **1** do max, **20** jednostek. Sprawia to, że będą istniały zarówno przedmioty lekkie, jak i takie które są w stanie zająć cały plecak. Przyjmując wartości przedmiotów z tego samego przedziału, uzyskujemy zróżnicowany zbiór danych na którym można testować algorytmy.

W przypadku danych skorelowanych (z rozkładu normalnego) możemy przyjąć rozkład, którego średnia (dla wartości i wag) będzie stanowić połowa**wielkości zbioru danych**, a odchylenie standardowe arbitralne $0.25 \times$ **wielkość zbioru danych**. Ponownie uzyskujemy przedmioty skupione wokół połowy maksymalnej pojemności plecaka, ale z zapewnioną różnorodnością w postaci odchylenia standardowego.

Takie rozwiązanie sprawia, że parametry przedmiotów i maksymalna pojemność plecaka skalują się wraz ze wzrostem liczby przedmiotów, co ujednolica środowisko testowe dla obydwu algorytmów.

7.4.3 Parametry konfigurowalne

Wśród parametrów konfigurowalnych znajdują się:

- **liczba powtórzeń wykonania** - dla danego zestawu parametrów, pozwala na uzyskanie uśrednionych wyników a także wyniku nalepszego i najgorszego.
- **liczba osobników w populacji** - ile osobników należy wygenerować w danej epoce, ważne w kontekście eksploracji przestrzeni
- **liczba najlepszych osobników w populacji** - na podstawie ilu najlepszych osobników aktualizować populację / wektor p-stwa w przypadku PBIL
- **wartość współczynnika uczenia** - wykorzystywane przy aktualizacji wartości wektora p-stwa w przypadku PBIL
- **liczba epok** - ile epok powinno trwać przeszukiwanie przestrzeni w przypadku każdego z algorytmów
- **wartości kroku** - przedziały determinujące wielkość kroku między sprawdzaną wielkością zbioru danych (dla małych wartości skok np. co 1, dla większych co 5)

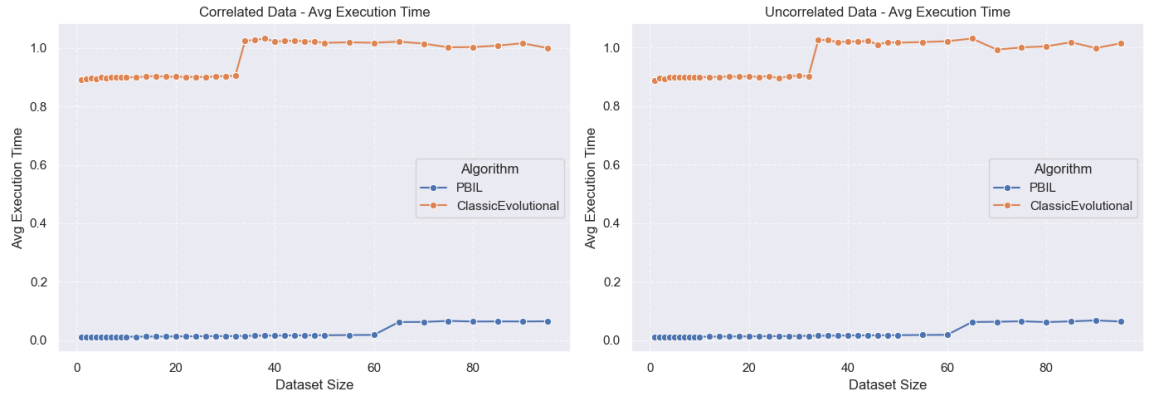
8 Wyniki

8.1 Przypadek ogólny

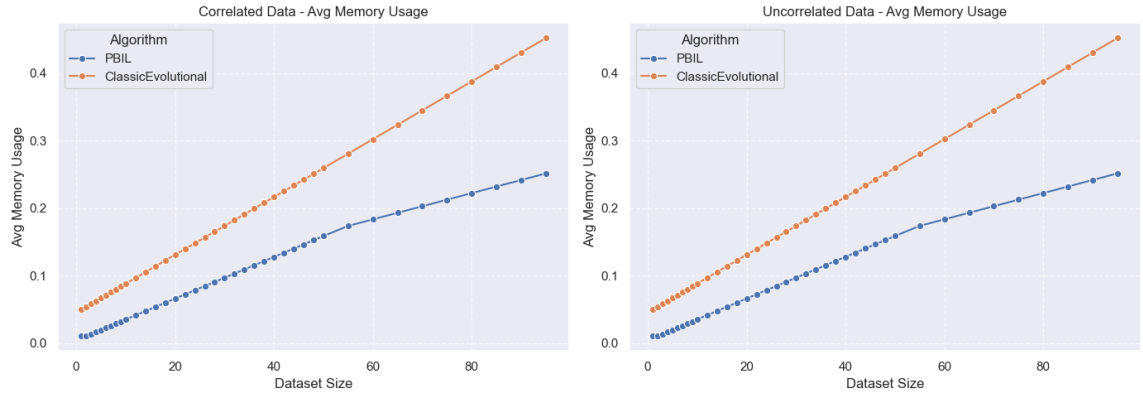
Niniejsze wyniki dotyczą porównania algorytmów dokonanego przy wartościach parametrów:

- **liczba powtórzeń wykonania** - 30
- **liczba osobników w populacji** - 150
- **liczba najlepszych osobników w populacji** - 20
- **wartość współczynnika uczenia** - 0.01
- **liczba epok** - 200
- **wartości kroku** - $[1, 10) = 1$; $[10, 50) = 2$; $[50, 100] = 5$

8.1.1 Czasy egzekucji i zajętość pamięci



Rysunek 1: Czasy egzekucji



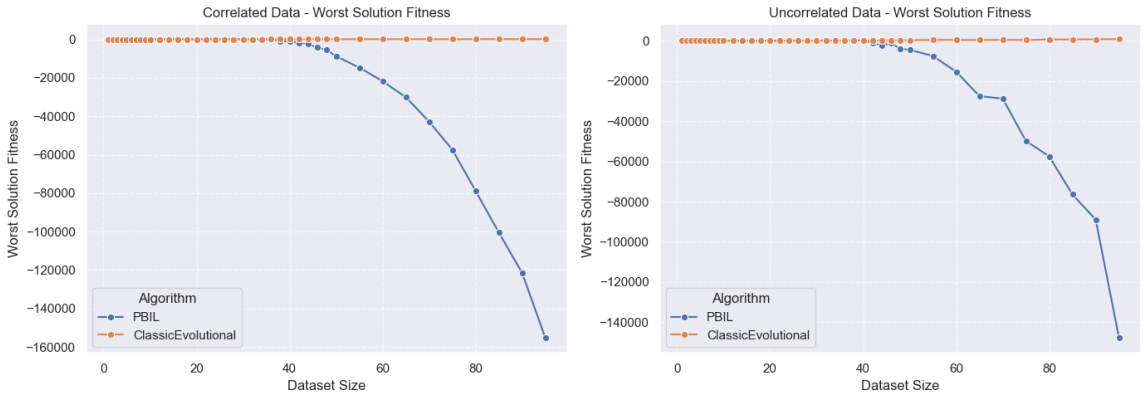
Rysunek 2: Zajętość pamięci

W obydwu przypadkach widzimy znaczną przewagę algorytmu **PBIL**. Dla każdej wielkości zbioru danych jest on rzędy wielkości szybszy i zajmuje mniej pamięci niż klasyczna wersja algorytmu ewolucyjnego.

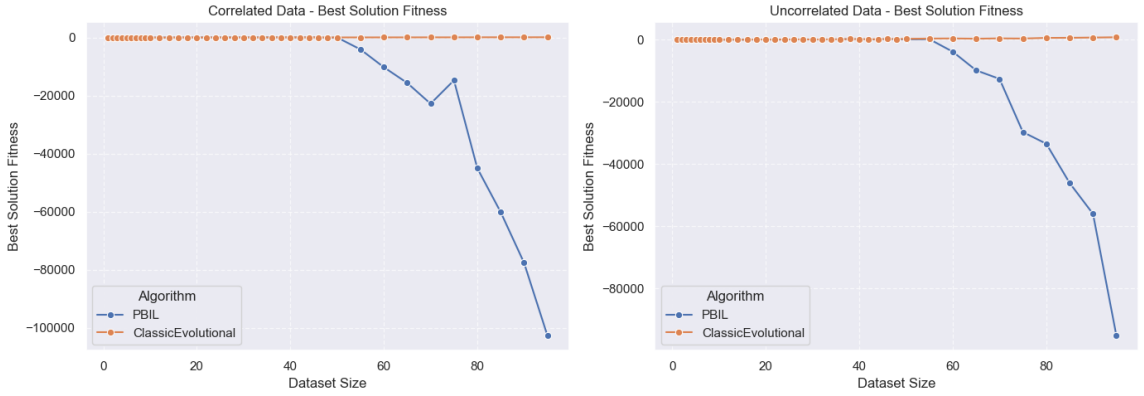
8.1.2 Uzyskane rozwiązania

W ramach przypomnienia, analizowane były rozwiązania uzyskane przy liczbie *30* powtórzeń algorytmów, przy zachowaniu reszty parametrów. Najlepsze rozwiązanie oznacza najlepsze rozwiązanie dla danej wielkości zbioru danych (innego w każdym z 30 powtórzeń) uzyskane w ciągu tych *30* powtórzeń. Analogicznie dla reszty danych.

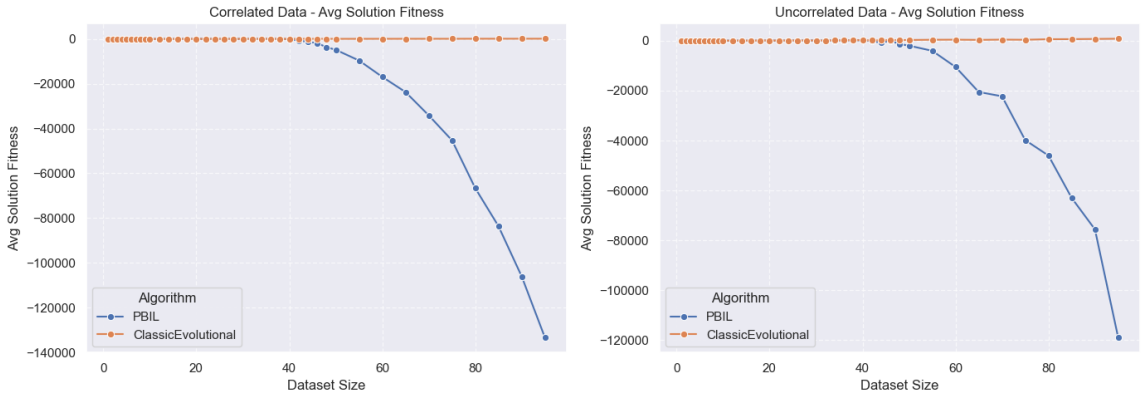
Jeżeli chodzi o skalę **y** na uzyskanych wykresach - algorytm PBIL radził sobie na tyle źle przy większych zbiorach danych, że miał problem ze znalezieniem chociaż jednego prawidłowego rozwiązania - wszystkie objęte były karą, na tyle dużą, że przysłoniła ona faktyczne, dobre rozwiązania klasycznego algorytmu.



Rysunek 3: Najgorsze rozwiązanie

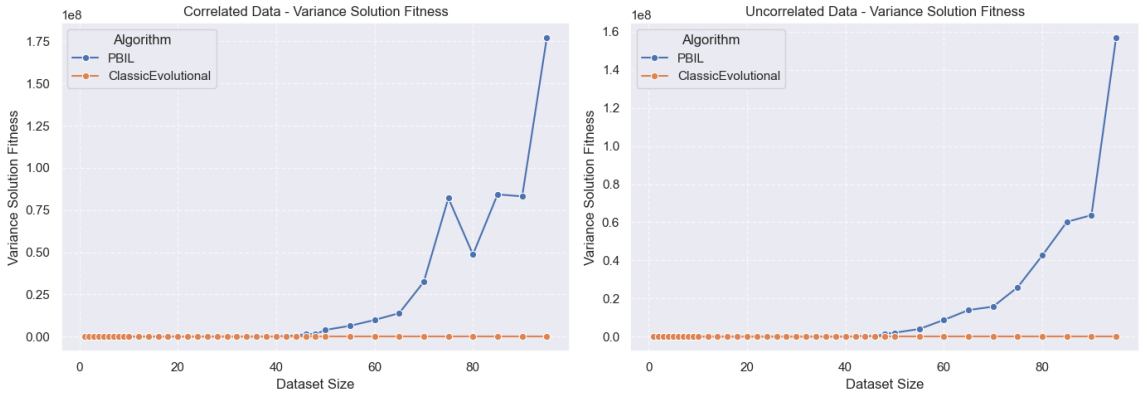


Rysunek 4: Najlepsze rozwiązanie



Rysunek 5: Średnie rozwiązanie

Wszystkie trzy powyższe wykresy są podobne w naturze - algorytm PBIL radzi sobie podobnie do algorytmu ewolucyjnego do ok. 35 możliwych przedmiotów. W późniejszych przypadkach, przy takiej samej wielkości populacji, liczbie epok i liczbie wybranych najlepszych osobników, jego jakość znacząco spada w stosunku do stałej jakości uzyskiwanej przez klasyczny algorytm ewolucyjny.



Rysunek 6: Wariancja

Na powyższym wykresie widać, że algortytm PBIL przy większych zbiorach danych zaczyna generować losowe (błędne) rozwiązania. Nie radzi sobie dobrze z przeszukiwaniem większych przestrzeni (przy takich samych parametrach, co klasyczna wersja algorytmu) i generuje rozwiązania znacznie od siebie różne - co wykazuje rosnąca wariancja.

Powyższe wykresy są wystarczające do stwierdzenia, że wyniki algorytmów dla wielkości zbiorów danych z przedziału [1, 100] znacząco się od siebie różnią i nie ma sensu wykonywać dodatkowo testu t-studenta w ramach potwierdzenia.

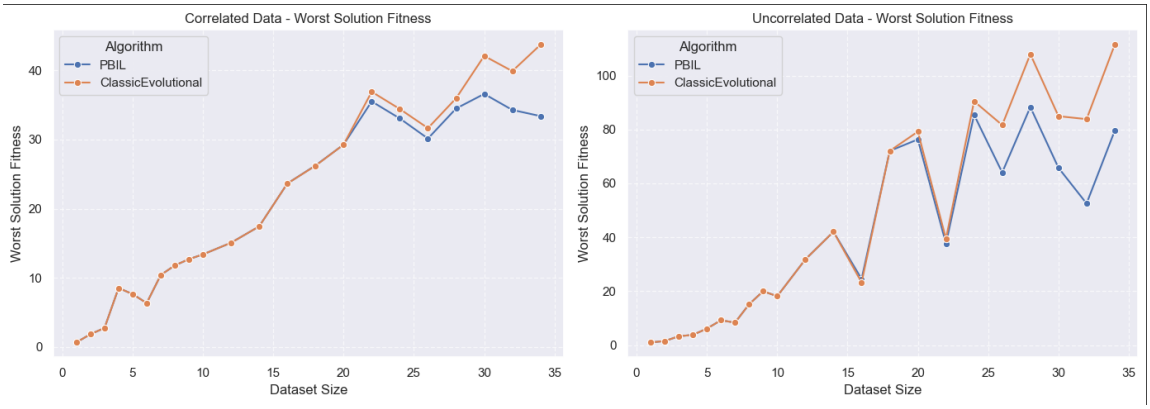
8.2 Poprawione uzyskane rozwiązania

W celu faktycznego zbadania różnic między algorytmami należy zawęzić zakres wielkości zbiorów danych do momentu, w którym działanie algorytmów się ze sobą pokrywa.

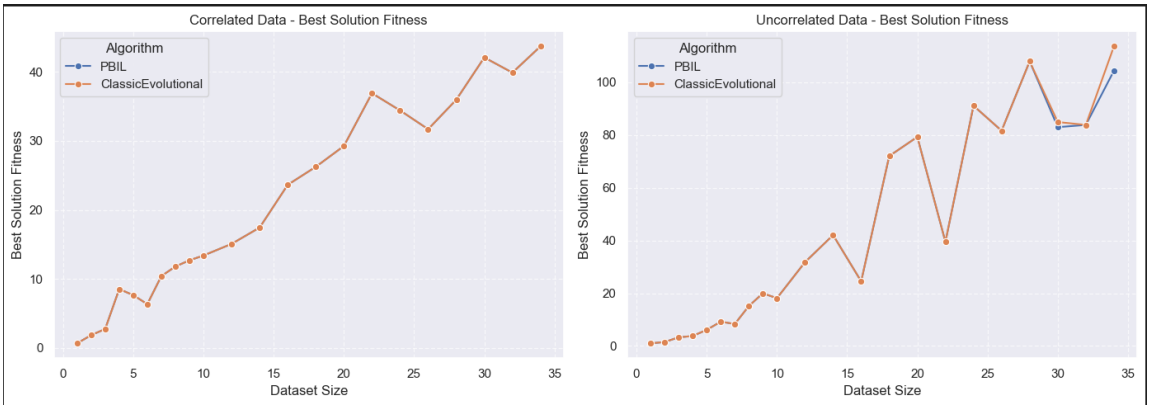
8.2.1 Czasy egzekucji i zajętość pamięci

Wyniki obrazują zawężone wykresy przedstawione w poprzedniej sekcji, więc nie ma sensu powielania uzyskanych wniosków.

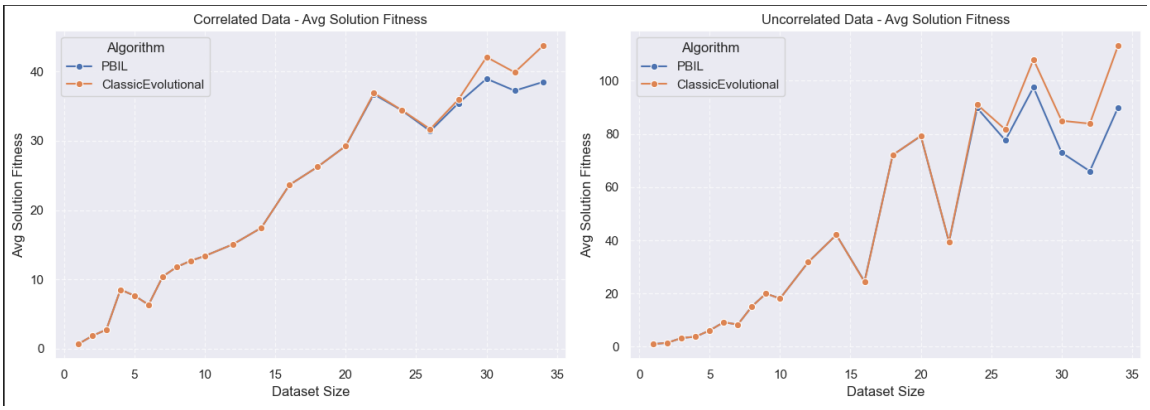
8.2.2 Uzyskane rozwiązania



Rysunek 7: Najgorsze rozwiązanie



Rysunek 8: Najlepsze rozwiązanie

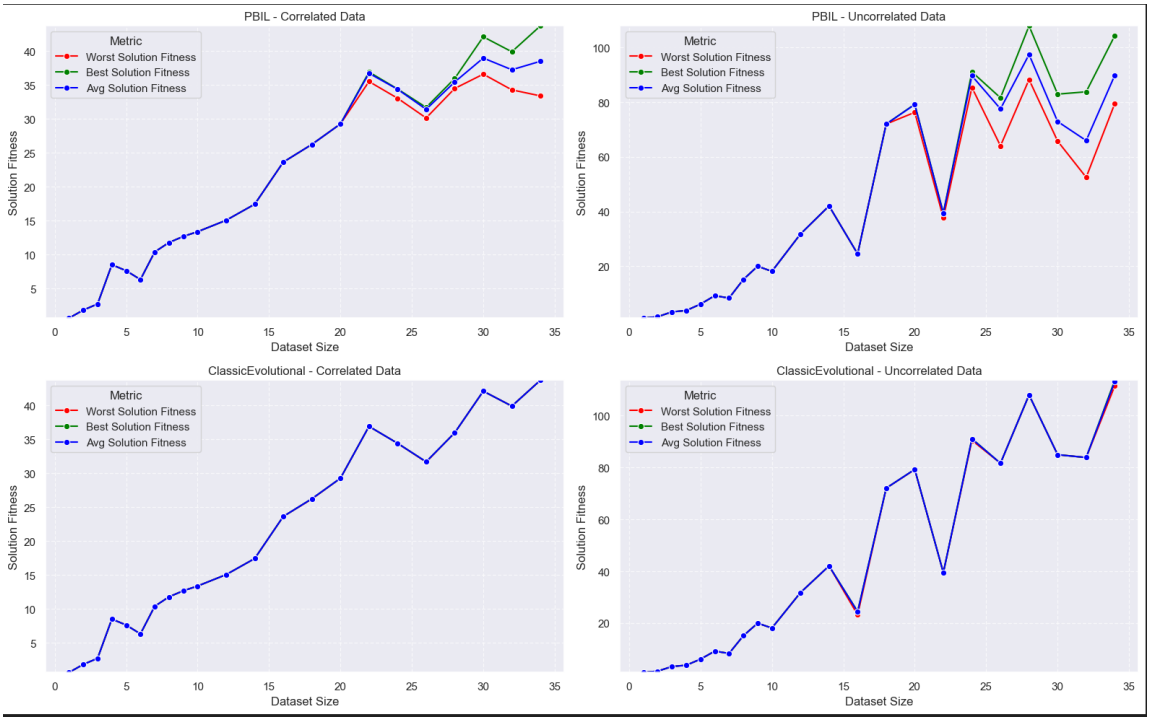


Rysunek 9: Średnie rozwiązanie

Przy górnym limicie wielkości zbiorów danych, wyniki uzyskane przez obydwa algorytmy są do siebie znacząco zbliżone. Na uwagę zasługuje wykres obrazujący najlepsze rozwiązania - wartości uzyskane przy pomocy algorytmu PBIL i ClassicEvolutional na danych nieskorelowanych pokrywają się w całości.

Różnice w najgorszych rozwiązaniach są trochę bardziej znaczące i przy większych zbiorach danych algorytm PBIL radzi sobie minimalnie gorzej od klasycznej wersji.

Zestawienie najlepszych, najgorszych i średnich rozwiązań prezentuje poniższy wykres:



Rysunek 10: Zestawienie najlepszych, najgorszych i średnich rozwiązań

Wykonany na powyższych, średnich rozwiązaniach **test t-studenta** (pochodzący z biblioteki *scipy*)

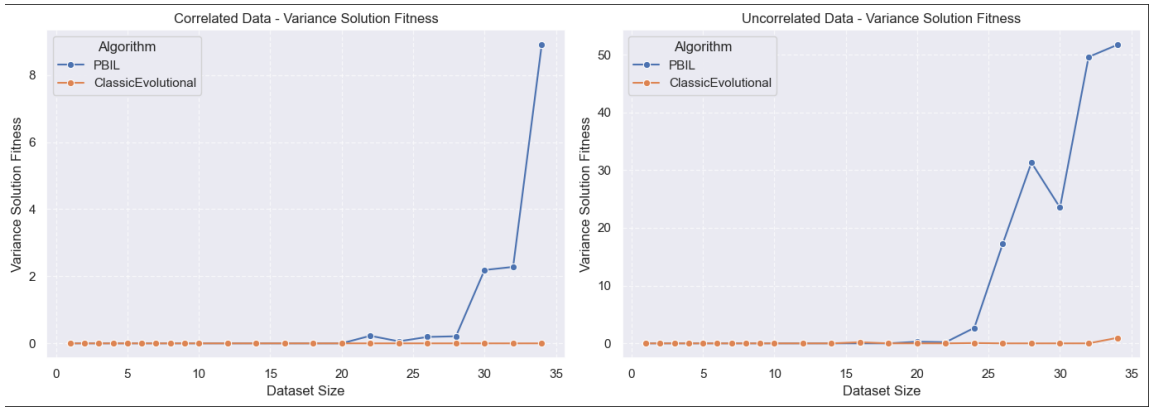
zwrócił wartości:

T-statistic: -0.1316, P-value: 0.8960

dla danych skorelowanych, oraz

T-statistic: -0.2850, P-value: 0.7770

dla nieskorelowanych. W obydwu przypadkach, wartości T są bliskie zeru, co oznacza, że wartości średnich są bardzo podobne. Wartości P-value również są znacznie większe od standardowego progu 0.05 i wyrażają prawdopodobieństwo uzyskania takich (lub bardziej ekstremalnych) wyników przy założeniu, że nie ma różnicy między średnimi. W tym przypadku, oznacza to, że nie mamy podstaw na odrzucenie hipotezy zerowej, która w tym przypadku oznacza, że **Nie ma statystycznie istotnej różnicy między wynikami PBIL i klasycznego algorytmu ewolucyjnego**.



Rysunek 11: Wariancja

Powyższa wariancja wynika ponownie z trudnościami algorytmu PBIL w uzyskaniu stabilnego rozwiązania przy większej liczbie przedmiotów. Wariancja klasycznego algorytmu ewolucyjnego zawsze wynosi 0, co oznacza że jest on w stanie niezawodnie znaleźć najlepsze rozwiązanie w każdym, z 30 powtórzeń.

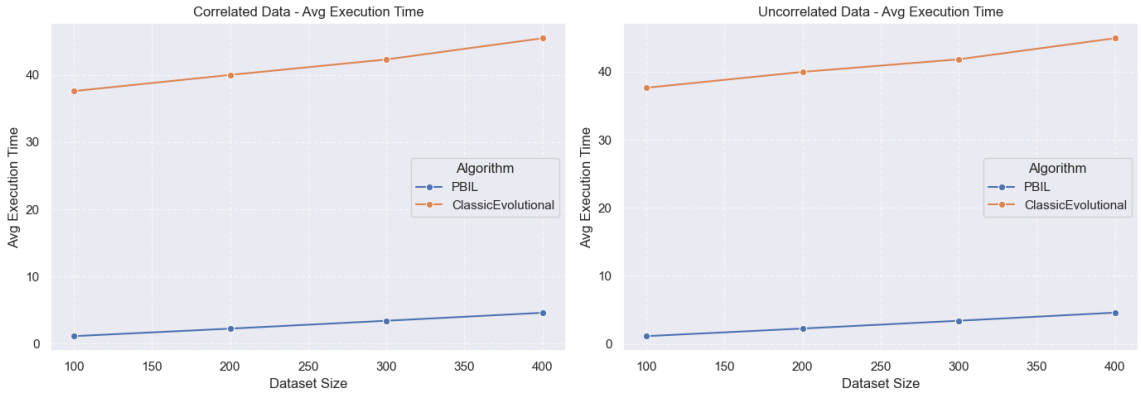
8.3 Przypadek ekstremalny

W ramach badań algorytmy zostały przetestowane również dla bardzo dużych zbiorów danych - zawierających nawet 400 przedmiotów. W związku z dużym nakładem obliczeniowym, porównanie zostało wykonane dla 4 wielkości zbiorów danych, przy następujących parametrach.

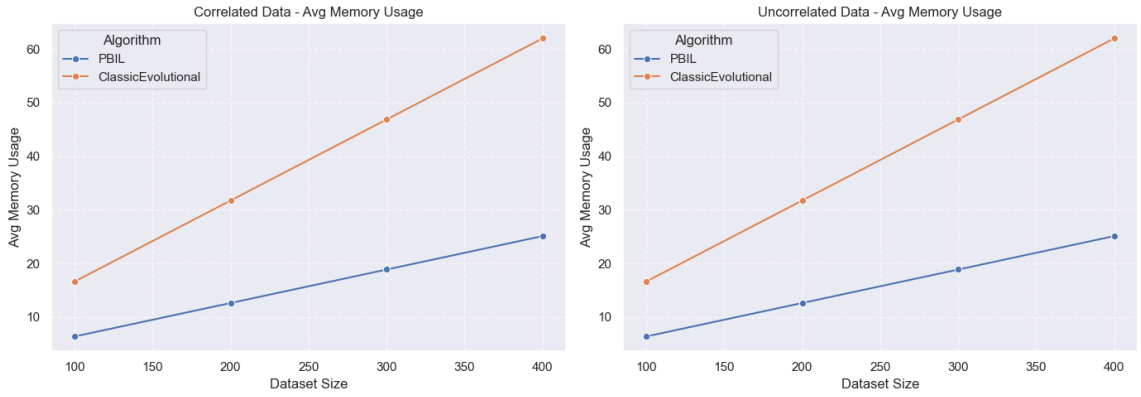
Niniejsze wyniki dotyczą porównania algorytmów dokonanego przy wartościach parametrów:

- **liczba powtórzeń wykonania** - 10
- **liczba osobników w populacji** - 5000
- **liczba najlepszych osobników w populacji** - 100
- **wartość współczynnika uczenia** - 0.002
- **liczba epok** - 200
- **wartości kroku** - 100 dla każdej wartości

8.3.1 Czasy egzekucji i zajętość pamięci



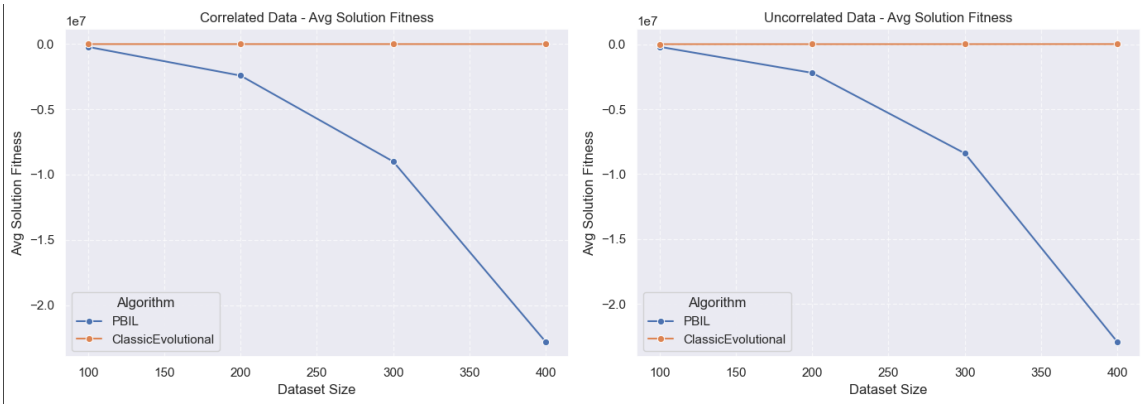
Rysunek 12: Czasy egzekucji



Rysunek 13: Zajętość pamięci

Uzyskane wyniki pokrywają się z tymi pochodzącymi z mniejszych zbiorów danych, ale naturalnie wartości są znacząco większe dla przypadku ekstremalnego. Algorytm PBIL dalej uzyskuje lepsze wyniki w czasie nie przekraczającym 5 sekund, podczas gdy algorytm ewolucyjny przekracza nawet wartości 40 sekund. Sytuacja wygląda analogicznie w kwestii zajętości pamięci.

8.3.2 Uzyskane rozwiązania



Rysunek 14: Średnie rozwiązanie

Ponownie, sytuacja wygląda analogicznie jak w przypadku mniejszych danych. Na pierwszy rzut oka wydaje się, że średnie wartości dla 100 przedmiotów pokrywają się, ale warto zwrócić uwagę, że skala y pokazuje wartości w skali *1e7*. Bezpośrednie wartości dla tej wielkości zbioru danych to **117.82** dla klasycznego algorytmu i aż **-210347.62** dla PBIL. Znaczy to, że w ekstremalnej skali obserwujemy to samo zachowanie, co w poprzednim paragrafie - algorytm PBIL (nawet przy tak dużych wartościach parametrów) nie jest w stanie znaleźć żadnego poprawnego rozwiązania.

Różnice w pozostałych przypadkach ponownie wykluczają sens przeprowadzania dodatkowych testów w celu wykazania różnic w jakości rozwiązań zwracanych przez obydwa algorytmy.

8.4 Podsumowanie

- Dla wielkości zbiorów danych z przedziału [1, 35] nie ma statystycznie istotnej różnicy między wynikami PBIL i klasycznego algorytmu ewolucyjnego, w związku z czym lepszy jest algorytm PBIL, który osiąga wyniki w krótszym czasie i przy mniejszych zasobach pamięciowych.
- W przypadku większych zbiorów danych, algorytm PBIL radzi sobie znacząco gorzej od klasycznego algorytmu ewolucyjnego i nie jest niezawodnym rozwiązaniem.
- Algorytm ewolucyjny osiąga wariancję równą 0 w przeważającej liczbie przypadków, przez co stanowi niezawodny algorytm do przeszukiwania przestrzeni problemu plecakowego - jest w stanie znaleźć to samo, optymalne rozwiązanie za każdym razem. Fakt ten dodatkowo podkreślają minimalne różnice między wartościami najgorszych, najlepszych i średnich rozwiązań.