Przeszukiwanie i Optymalizacja

Dokumentacja końcowa - 21Z

Realizacja w zespole: Bratosiewicz Konrad, Marczuk Jakub
Temat projektu: SK.POP.2 - Złodziej
Repozytorium projektu: https://github.com/kbratosi/POP-zlodziej

Treść zadania

Złodziej ukradł X gramów złota ze skarbca i wraca do domu pociągiem. Żeby uniknąć schwytania przez policję, musi zamienić złoto na banknoty, więc postanawia sprzedać złoto pasażerom pociągu. Zainteresowanych kupnem jest N pasażerów, każdy z nich zgadza się kupić Ai, gdzie i ∈ (1, 2, …, N) gramów złota za Vi, i ∈ (1, 2, …, N). Złodziej chce uciec przed policją, jednocześnie maksymalizując zysk.

Założenia projektu

- Implementacja algorytmu ewolucyjnego, który wskaże pasażerów, którym złodziej powinien sprzedać złoto oraz sumę wartości banknotów, którą zarobi.
- Implementacja algorytmu plecakowego opartego na programowaniu dynamicznym.
- Porównanie algorytmów pod względem złożoności obliczeniowej i jakości otrzymywanych wyników. Zbadanie wpływu parametrów algorytmu ewolucyjnego na otrzymywane wyniki.

Opis funkcjonalny

Struktura projektu

Projekt obejmuje:

1. implementację algorytmu plecakowego
2. moduł algorytmu ewolucyjnego
3. skrypty testujące
4. pliki konfiguracyjne, zawierające parametry dla algorytmu ewolucyjnego

Odczytywanie parametrów z pliku

W celu uruchomienia algorytmu ewolucyjnego wymagane jest podanie stosunkowo dużej liczby parametrów. Wpisywanie ich w odpowiedniej kolejności do terminala jest czasochłonne, a przy rosnącej liczbie zmiennych może doprowadzić do popełnienia błędu. Mając to na uwadze zostało zaimplementowane odczytywanie ich z pliku. Z uwagi na charakter danych zostały utworzone dwa pliki (podane nazwy mają charakter poglądowy - najważniejszy jest wykorzystany format rozszerzenia):

- **parameters.json** - przechowuje wszystkie niezbędne parametry z podanymi wartościami w popularnym i przyjaznym dla użytkownika formacie `.json`

```
{
   "max_weight":1000,
   "populations": 2000,
   "crossover_probability": 0.25,
   "mutation_probability": 0.01,
   "generations": 180
}
```

- **characteristic.csv** - możliwe jest również przekazanie do algorytmu ewolucyjnego list zmiennych A_i i V_i. Z uwagi na typ danych został wykorzystany format `.csv`

```
1, 2, 3, 4, 5, 7, 10, 2  // Ai
10, 8, 9, 4, 4, 13, 4, 9 // Vi
```

Skrypty testujące

Jednym z założeń projektowych było przeprowadzenie testów. Z tego względu zostały przygotowane specjalne skrypty automatyzujące cały proces. Pierwszą warstwą abstrakcji było stworzenie jednostkowego przypadku, jako skrypt dla interpretera języka Python. Wykonuje on następujące operacje: interpretacja przekazanych argumentów, uruchomienie algorytmu plecakowego i ewolucyjnego, obliczenie czasów wykonań i zapisanie rezultatów do pliku wynikowego. Kolejną warstwą abstrakcji stanowi skrypt dla interpretera powłoki systemu Unix - Shell. W tym przypadku przygotowany skrypt odpowiada za iteracyjne uruchomienie wcześniej omówionego programu z wyspecyfikowanymi dla danego scenariusza testowego danymi.

Zbieranie danych z przeprowadzonych testów również zostało zautomatyzowane i zaimplementowane na poziomie skryptu napisanego w języku Python. Ponownie został wykorzystany format `.csv`. Zapisywane są takie informacje jak: maksymalna waga, wielkość zbioru A_i ,rozmiar populacji, liczba generacji, prawdopodobieństwo krzyżowania, prawdopodobieństwo mutacji, wyniki działania algorytmu plecakowego i ewolucyjnego, flaga zgodności wyników i czasy działania poszczególnych algorytmów. Oprócz tego zapisywany jest w postaci logu zrzut danych wyświetlanych na konsoli w trakcie wykonywania programu.

Testowe zbiory danych

Pierwszym etapem testów przeprowadzanych w ramach porównania algorytmów jest generacja testowych zbiorów danych. Na podstawie ustalonej jako parametr generacji wielkości problemu tworzone są wektory generowanych losowo wag A_i oraz wartości V_i.
W zależności od scenariusza testowego masa ukradzionego przez złodzieja złota X jest podawana jako parametr testu bądź losowana. Losowanie każdej z wag i wartości jest realizowane z użyciem modułu `random` języka Python.
Po zakończonej generacji następuje uruchomienie algorytmów. Oba algorytmy otrzymują ten sam zestaw danych testowych jako parametry funkcji.

Algorytm plecakowy

Algorytm plecakowy jest algorytmem programowania dynamicznego. Rozwiązuje on zadanie znalezienia optymalnego zbioru pasażerów w celu maksymalizacji zysku poprzez rozbicie go na podproblemy. Istotą algorytmu plecakowego jest obliczenie optymalnego zbioru pasażerów dla każdej wartości masy ukradzionego złota od 1 do X. W celu skrócenia obliczeń wyniki poszczególnych podproblemów są zapisywane w macierzy obliczeniowej.

Niech Ai, gdzie i $\in$ (1, 2, …, N) będzie wagą elementów oraz Vi, i $\in$ (1, 2, …, N) - wartościami. Algorytm ma zmaksymalizować wartość elementów przy zachowaniu sumy ich wagi mniejszej bądź równej XS. Niech P(i,j) będzie największą możliwą wartością, która może być otrzymana przy założeniu wagi mniejszej bądź równej j i wykorzystaniu pierwszych i elementów.

Funkcja P(i,j) definiowana jest rekurencyjnie:

- P(0, j) = 0
- P(i, 0) = 0
- P(i, j) = P(i - 1, j), jeśli Ai > j
- P(i, j) = max(P(i - 1, j), P(i - 1, j - Ai) + Vi), jeśli Ai ≤ j

Pseudokod:

```
for i := 0 to n do
        P[i, 0] := 0
for j := 0 to X do
        P[0, j] := 0
// rozważanie kolejno i pierwszych przedmiotów
for i := 1 to n do
        for j := 0 to W do
                // czy element A[i] przekracza dostępną ilość złota
                if ( A[i] > j ) then          // tak - pomiń
                        P[i, j] := P[i - 1, j]
                else                  // nie - oblicz wartość, zapisz referencję na poprzedni element
                        P[i, j].score := max( P[i - 1, j], P[i - 1, j - A[i]] + V[i] )
                        P[i, j].prev := P[i - 1, j - A[i]]
```

Odczyt rozwiązania

Rozwiązaniem problemu jest wynik dla P(N, X). Na podstawie referencji na poprzednie pola możliwe jest odtworzenie sekwencji indeksów pasażerów, z którymi złodziej powinien przeprowadzić transakcję, aby zmaksymalizować zysk.

Implementacja

Algorytm został zaimplementowany w formie funkcji. Jej argumentami są tablica wag A, tablica wartości V oraz masa ukradzionego złota X. Funkcja zwraca optymalne rozwiązanie w formie wektora binarnego oraz sumy zarobku.

Algorytm ewolucyjny

Jest to metoda rozwiązywania problemów optymalizacyjnych inspirowana mechanizmami zachodzącymi w naturze - w tym konkretnym przypadku mechanizmy związane z ewolucją. W przypadku rozwiązywanego problemu bazuje ona na następujących ustaleniach:

Osobnik: wektor binarny długości N: 1 oznacza wykonanie transakcji, 0 - brak transakcji
Funkcja celu: suma wartości wszystkich transakcji, jeżeli nie przekroczono wagi sprzedawanego złota, bądź 0, jeżeli przekroczono sumaryczną wagę złota we wszystkich transakcjach
Selekcja: turniejowa (dla trzech osobników)
Krzyżowanie: utworzenie wektora z podciągu o losowej długości k jednego osobnika i długości N - k drugiego osobnika z prawdopodobieństwem Km.
Mutacja: zmiana wartości pojedynczego bitu w wektorze osobnika z prawdopodobieństwem Pm (sprawdzenie dla każdego bitu w wektorze).

Na początku działania algorytmu tworzona jest populacja o podanej jako argument liczbie osobników. Każdy z nich posiada generowany losowo własny genom o długości odpowiadającej liczbie parametrów (i) zadania. Następnie obliczana jest wartość funkcji celu dla stworzonej populacji. Kolejnym krokiem działania algorytmu jest wykonywanie operacji w pętli do momentu niespełnienia warunku o zdefiniowanej liczbie generacji. Wywoływane funkcje to (w kolejności):

1. wybór dwóch osobników z wykorzystaniem selekcji turniejowej
2. dokonanie krzyżowania z określonym prawdopodobieństwem pomiędzy zwycięzcami turnieju w celu utworzenia nowego osobnika
3. dokonanie mutacji genów każdego osobnika z określonym prawdopodobieństwem
4. obliczenie funkcji celu nowej populacji
5. zastąpienie poprzedniej populacji nową i inkrementacja generacji

W momencie zakończenia wykonywania operacji w pętli program wybiera najlepszego osobnika i zwraca jego genotyp, wartość funkcji celu i wagę proponowanego rozwiązania

Operacje 1-2 są powtarzane do momentu uzyskania populacji równie licznej co poprzednia

Z uwagi na charakterystykę działania algorytmu ewolucyjnego wymagane było jego dostrojenie, tzn. dobór odpowiednich parametrów w celu uzyskania jak najlepszych rozwiązań w jak najkrótszym czasie. W uproszczeniu sam proces polegał na inicjacji parametrów losowymi wartościami zgodnie z zaleceniami co do natury algorytmu, np: małe prawdopodobieństwo mutacji. Następnie zostały przeprowadzone testy badające wpływ zmiany wartości parametrów na otrzymywane rozwiązania, jak również czas działania algorytmu. Na podstawie zgromadzonych wyników zostały ustalone parametry do efektywnego rozwiązywania postawionego problemu.

Wykorzystane narzędzia

Język implementacji projektu: Python
IDE: Visual Studio Code
Repozytorium: git
Generacja liczb pseudolosowych: Python.random
Obsługa plików wejściowych/wyjściowych: Python.json, Python.csv

Testowanie

Pragnąc szczegółowo zbadać porównywane algorytmy do rozwiązywania postawionego problemu zostały przygotowane kompleksowe scenariusze testowe uwzględniające wszystkie główne parametry algorytmu ewolucyjnego. W toku testowania zrezygnowaliśmy z porównywania wektorów wynikowych - kilka różnych układów wektorów może dać jednakowy wynik, więc nie jest to warunek konieczny zgodności rozwiązań.

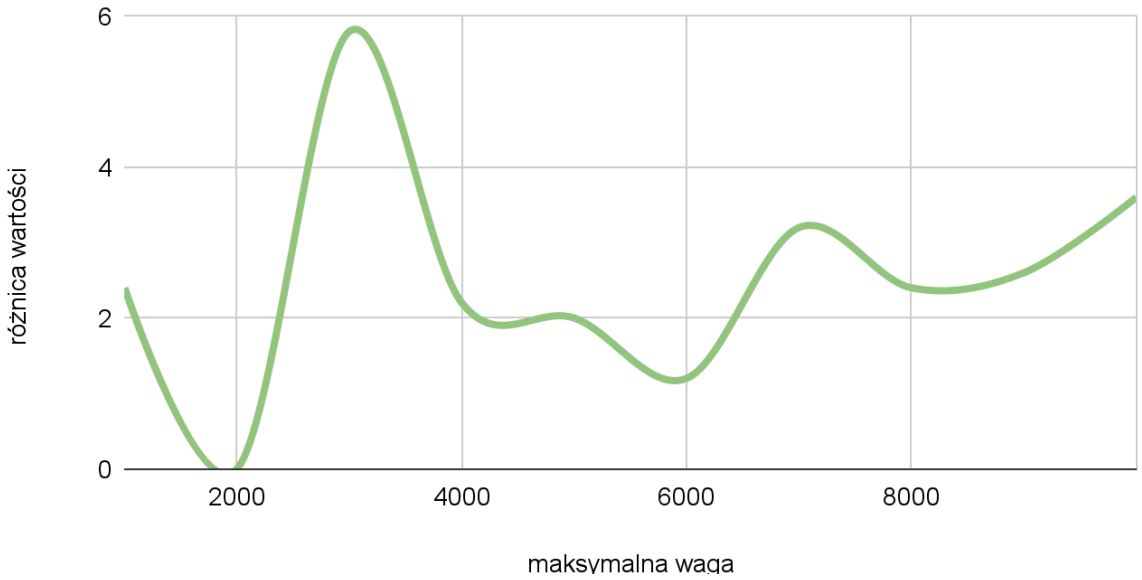
Interpretacja scenariuszy testowych opiera się na następujących zasadach:

- pogrubioną czcionką zaznaczone są badane parametry, których wartość była modyfikowana w zdefiniowanym zakresie w czasie wykonywania kolejnych iteracji testów
- linią pionową zaznaczone są kolejne przypadki, w których testowana była zmiana wyżej wymienionego parametru
- różnica wartości - kryterium oceny wyników
  różnica wartości = *wynik algorytmu plecakowego - wynik algorytmu ewolucyjnego*
- jako punkt odniesienia wykorzystany jest algorytm plecakowy
- każdy z przypadków danego scenariusza był wykonywany wielokrotnie w celu uśrednienia otrzymanego wyniku i tym samym zmniejszenia błędu
- w niektórych testach uzyskane wyniki pomiędzy scenariuszami różnią się o rząd wielkości. Mając to na uwadze, w celu zachowania dokładności i klarowności prezentowanych pomiarów zastosowane zostały dwie skale. Użycie skali logarytmicznej powodowało spadek czytelności wykresu.

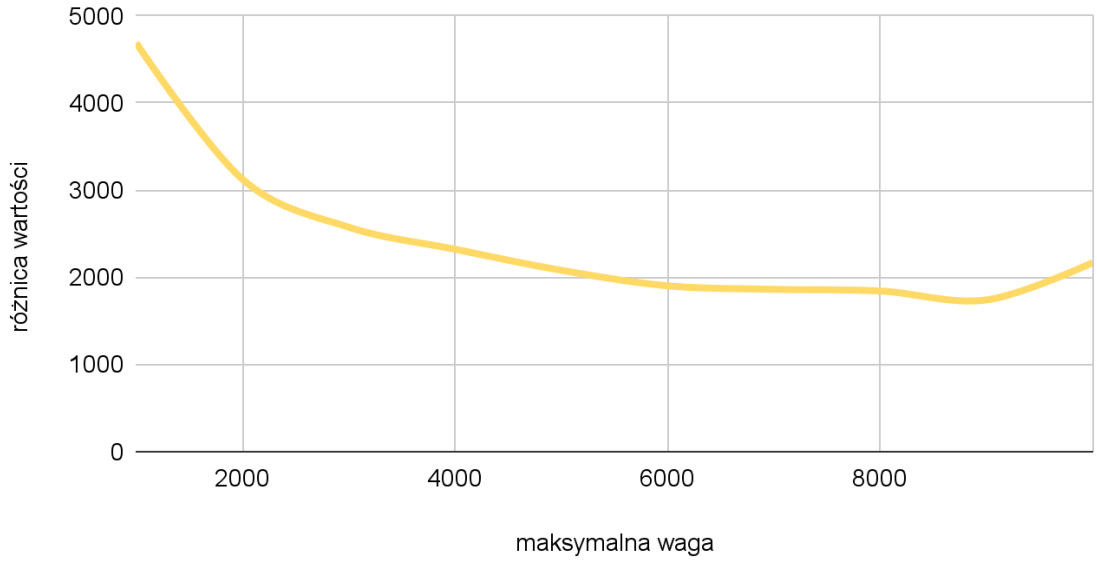
Test 1 - Badanie wpływu zmiany maksymalnej wagi X na otrzymywane wyniki i czas wykonania

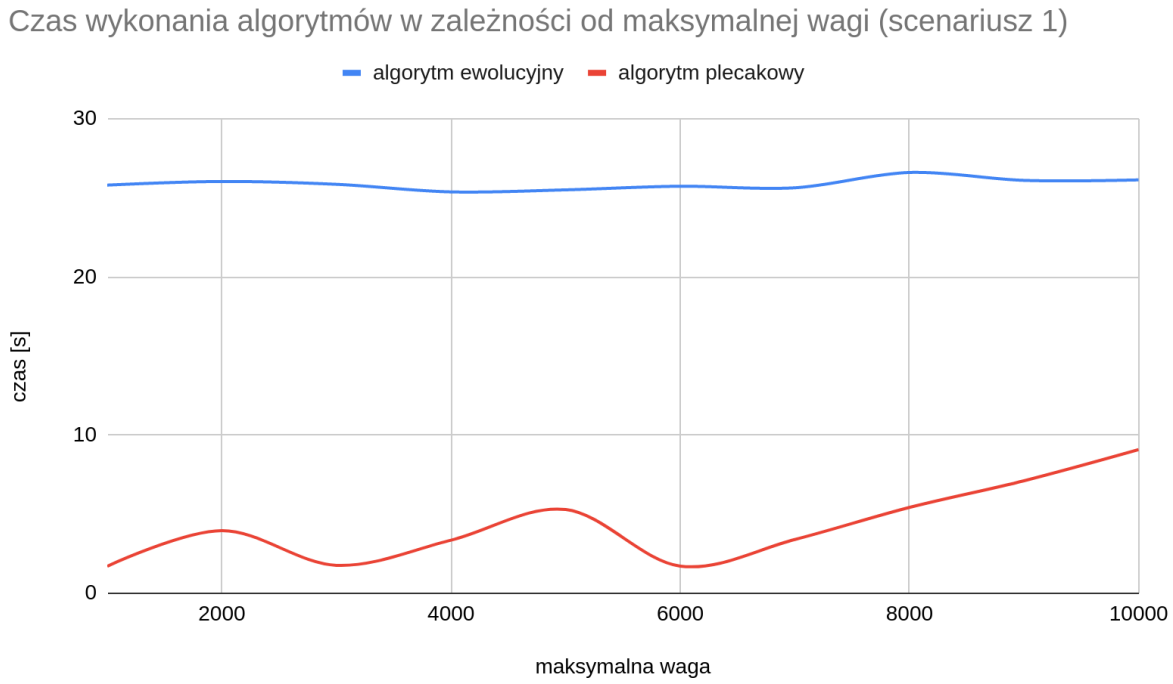
maksymalna waga	1k - 10k (krok co 1k)	
wielkość problemu N	100	500
wielkość populacji	2000	10000
liczba generacji	180	900
prawdopodobieństwo krzyżowania	0.25	
prawdopodobieństwo mutacji	00.1	

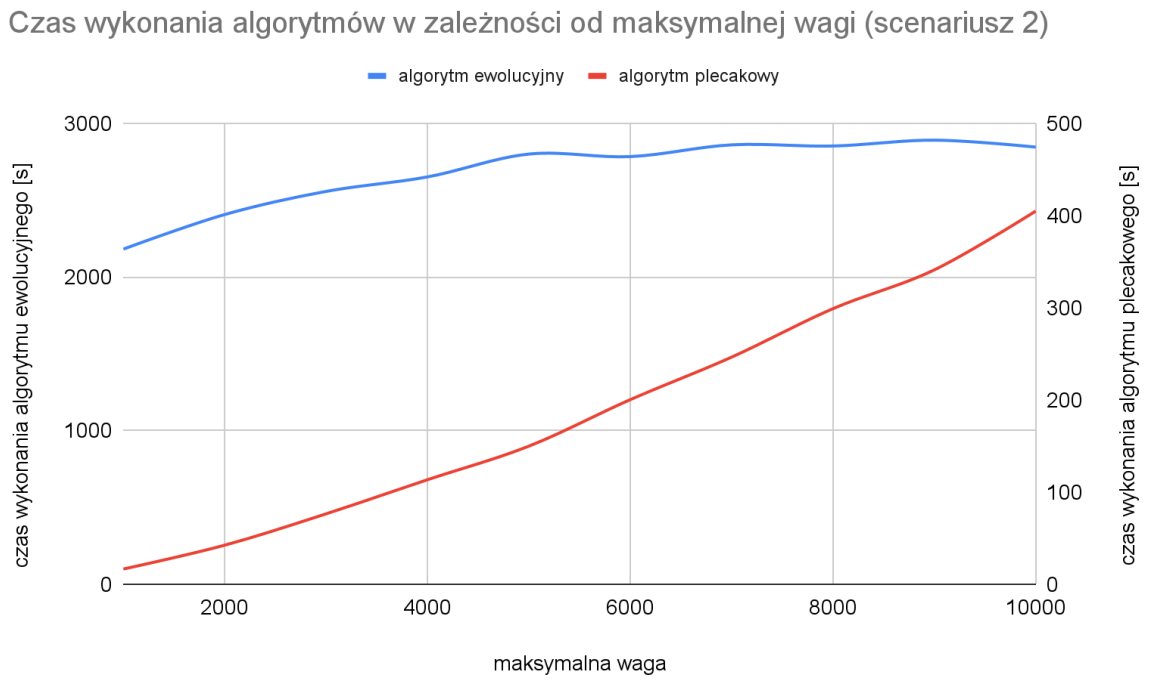
Różnica wartości w zależności od maksymalnej wagi (scenariusz 1)



Różnica wartości w zależności od maksymalnej wagi (scenariusz 2)





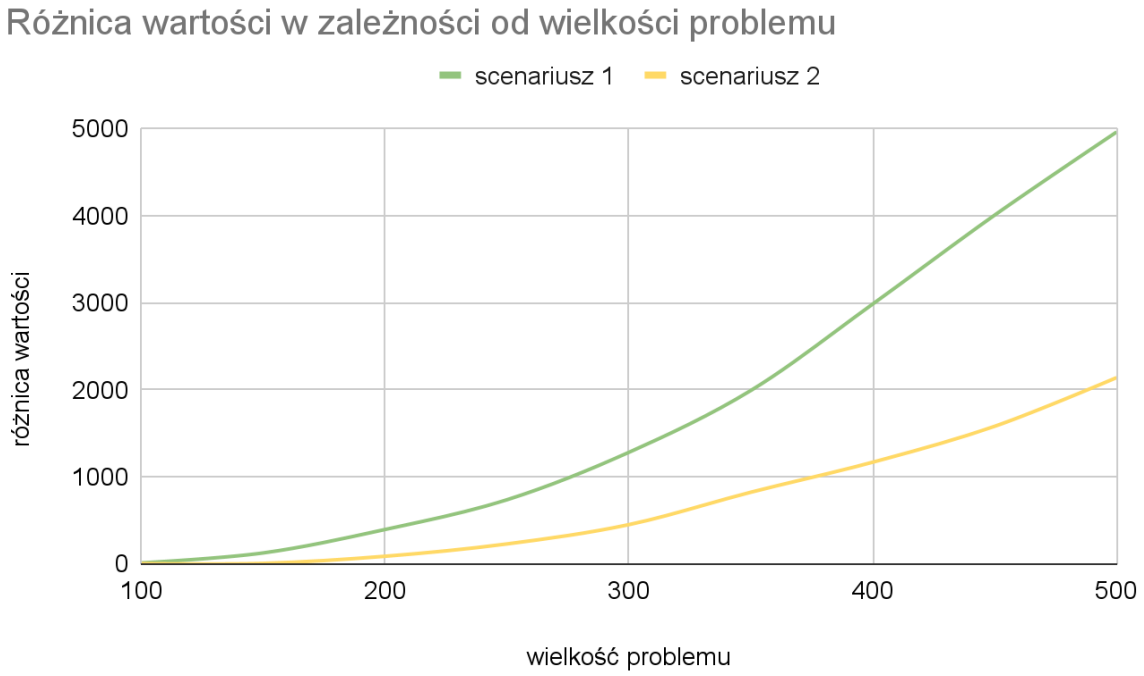


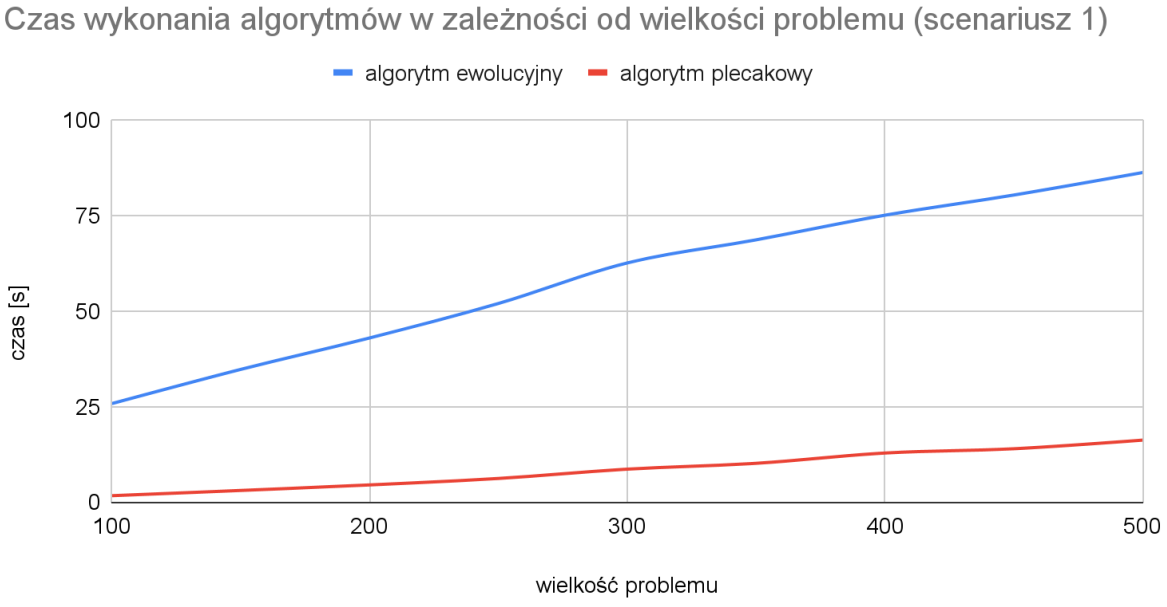
Wnioski:

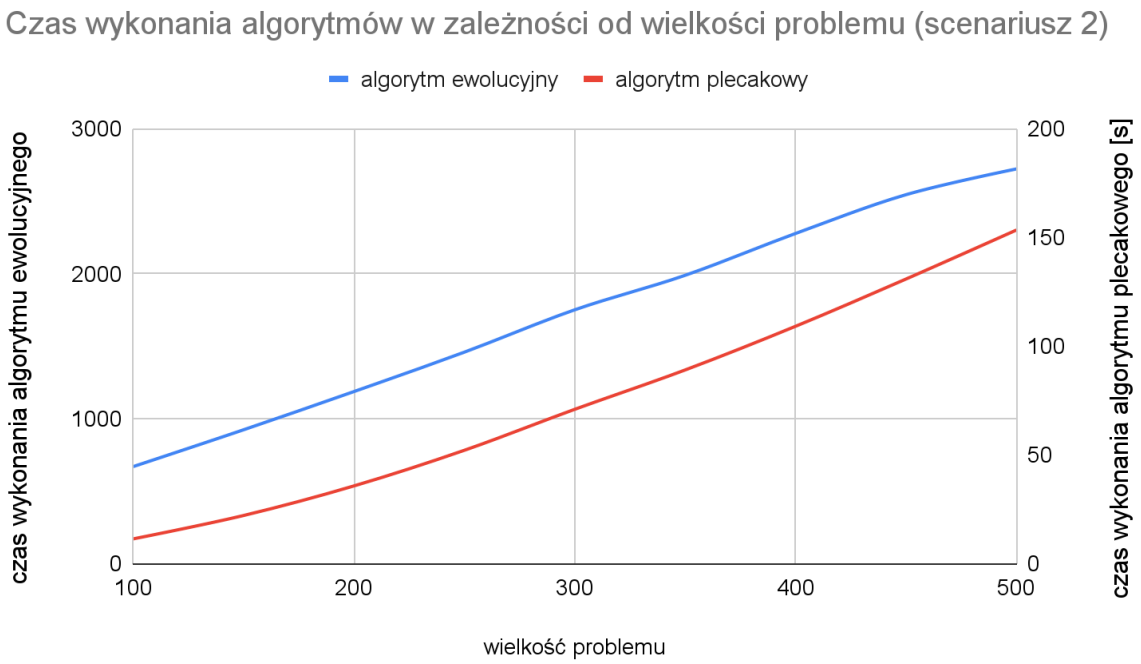
- wraz ze wzrostem maksymalnej wagi różnica wartości maleje. Szczególnie to widać w przypadku drugiego scenariusza. Różnica znacząco maleje do punktu dostrojenia dla algorytmu ewolucyjnego (maksymalna waga = 5000).
- czas wykonania algorytmu ewolucyjnego jest w przybliżeniu stały niezależnie od wartości maksymalnej wagi X. Z kolei czas wykonania algorytmu plecakowego rośnie kwadratowo w odniesieniu do maksymalnej wagi X.

Test 2 - Badanie wpływu zmiany wielkości problemu N na otrzymywane wyniki i czas wykonania

maksymalna waga	1000	5000
wielkość problemu N	100 - 500 (co 50)	
wielkość populacji	2000	10000
liczba generacji	180	900
prawdopodobieństwo krzyżowania	0.25	
prawdopodobieństwo mutacji	0.01	







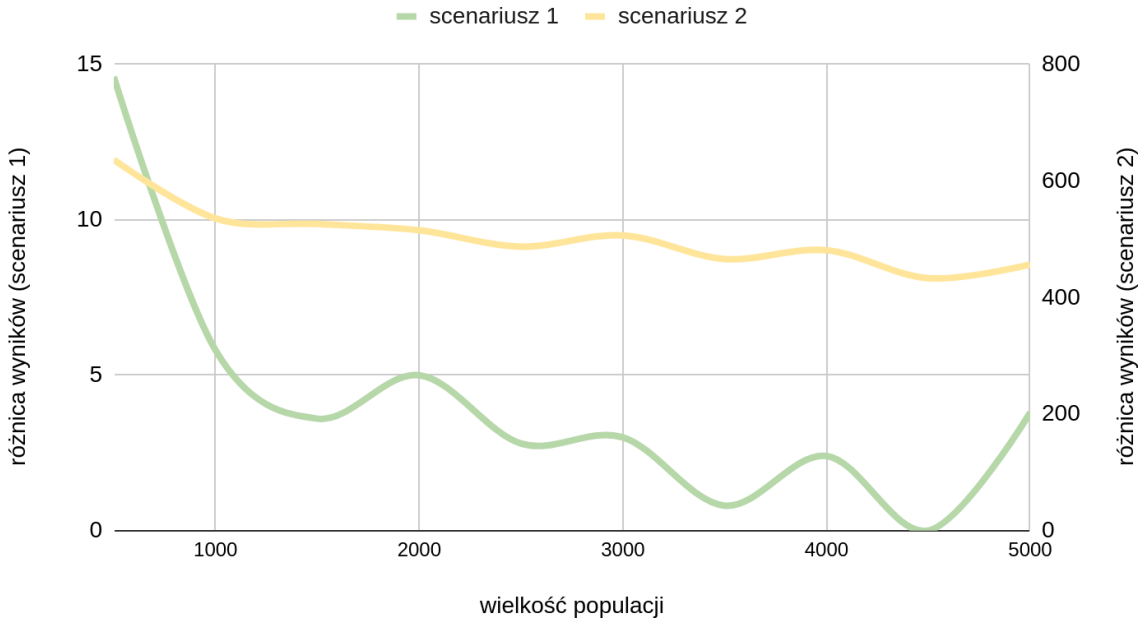
Wnioski:

- Wraz ze wzrostem wielkości problemu rośnie różnica otrzymanych wyników. Wynika to z niskiej skuteczności dostrojonego algorytmu ewolucyjnego do rozwiązywania problemów o zmiennym rozmiarze.
- Wzrost wielkości populacji oraz liczby generacji powoduje zmniejszenie różnicy wartości. Algorytm ewolucyjny może znaleźć lepsze rozwiązanie kosztem dłuższego czasu jego wykonania.
- Czas wykonania algorytmu ewolucyjnego rośnie liniowo względem wielkości problemu N.
- Czas wykonania algorytmu plecakowego rośnie kwadratowo względem wielkości problemu N.

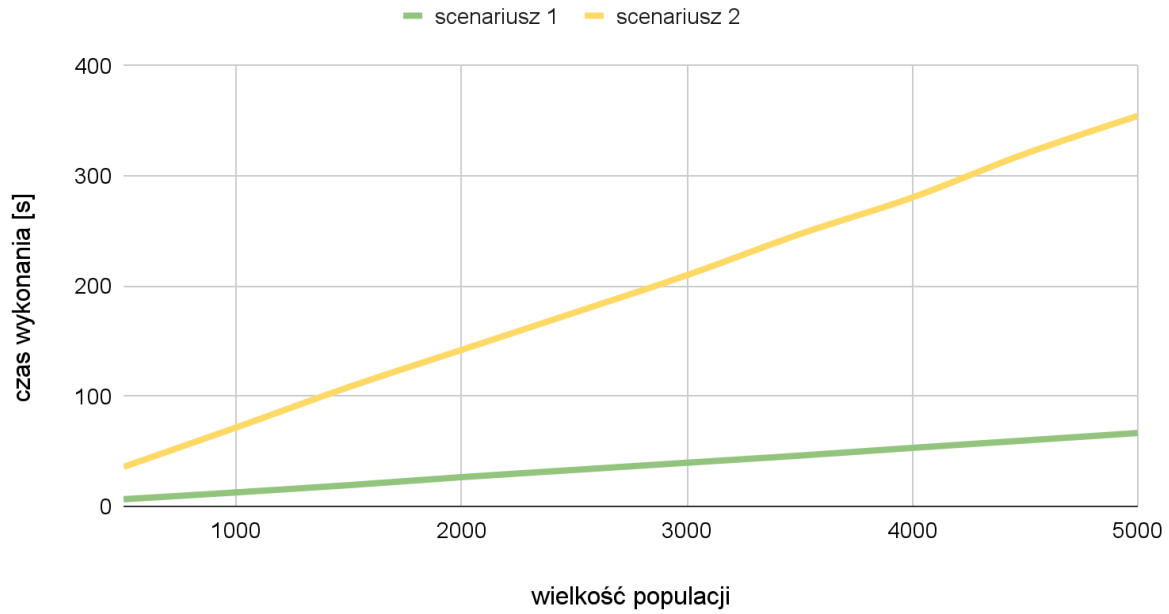
Test 3 - Badanie wpływu zmiany wielkości populacji na otrzymywane wyniki i czas wykonania

maksymalna waga	1000	2500
wielkość problemu N	100	250
wielkość populacji	500 - 5000 (co 500)	
liczba generacji	180	450
prawdopodobieństwo krzyżowania	0.25	
prawdopodobieństwo mutacji	0.01	

Różnica wyników w zależności od wielkości populacji



Czas wykonania algorytmu w zależności od wielkości populacji

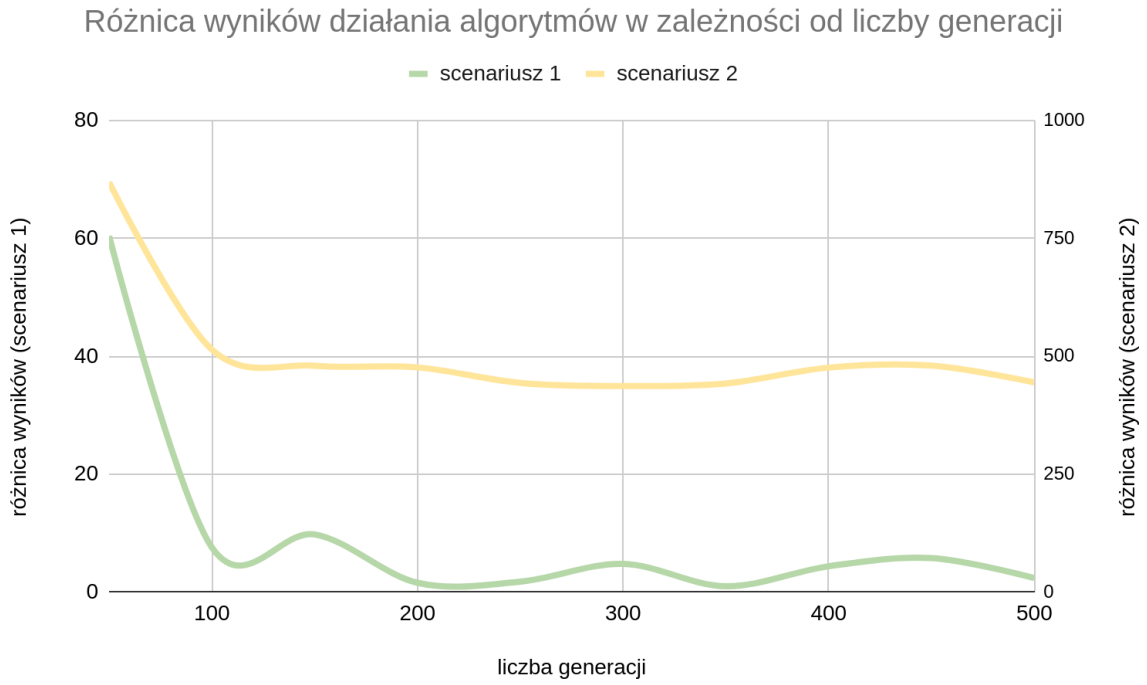


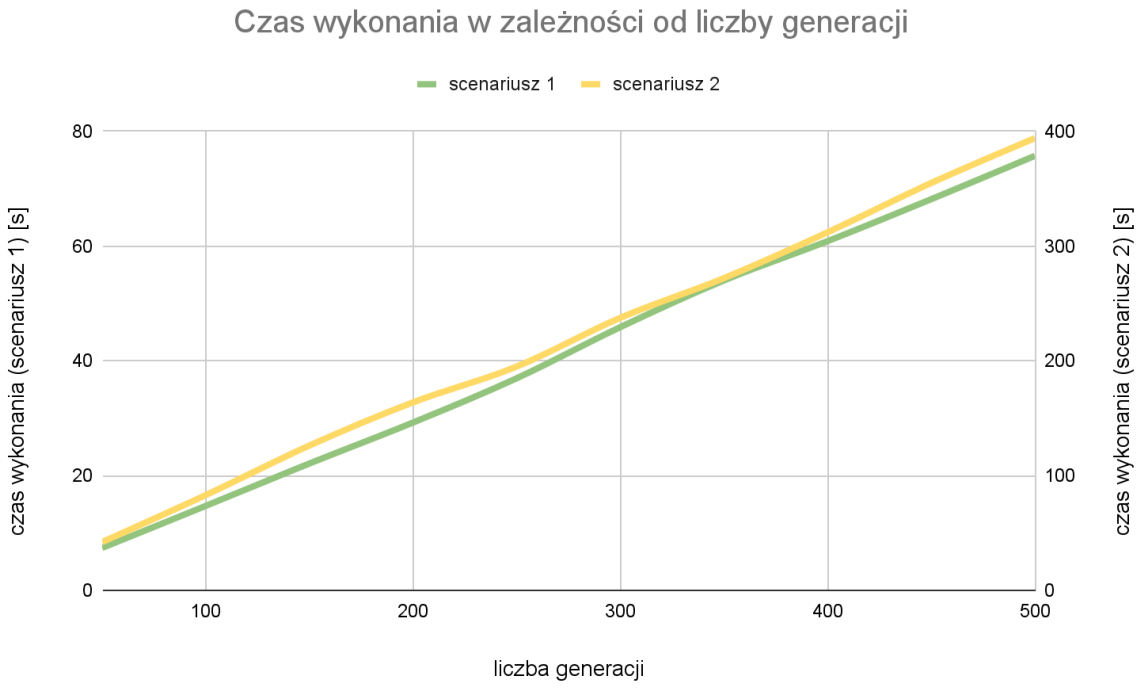
Wnioski:

- Wraz ze wzrostem populacji różnica wartości maleje. Większa liczba populacji skutkuje uzyskaniem lepszych jakościowo wyników, ale kosztem dłuższego czasu działania algorytmu
- Czas wykonania algorytmu rośnie liniowo względem wielkości populacji

Test 4 - Badanie wpływu liczby generacji na otrzymywane wyniki i czas wykonania

maksymalna waga	1000	2500
wielkość problemu N	100	250
wielkość populacji	2000	5000
liczba generacji	50 - 500 (co 50)	
prawdopodobieństwo krzyżowania	0.25	
prawdopodobieństwo mutacji	0.01	



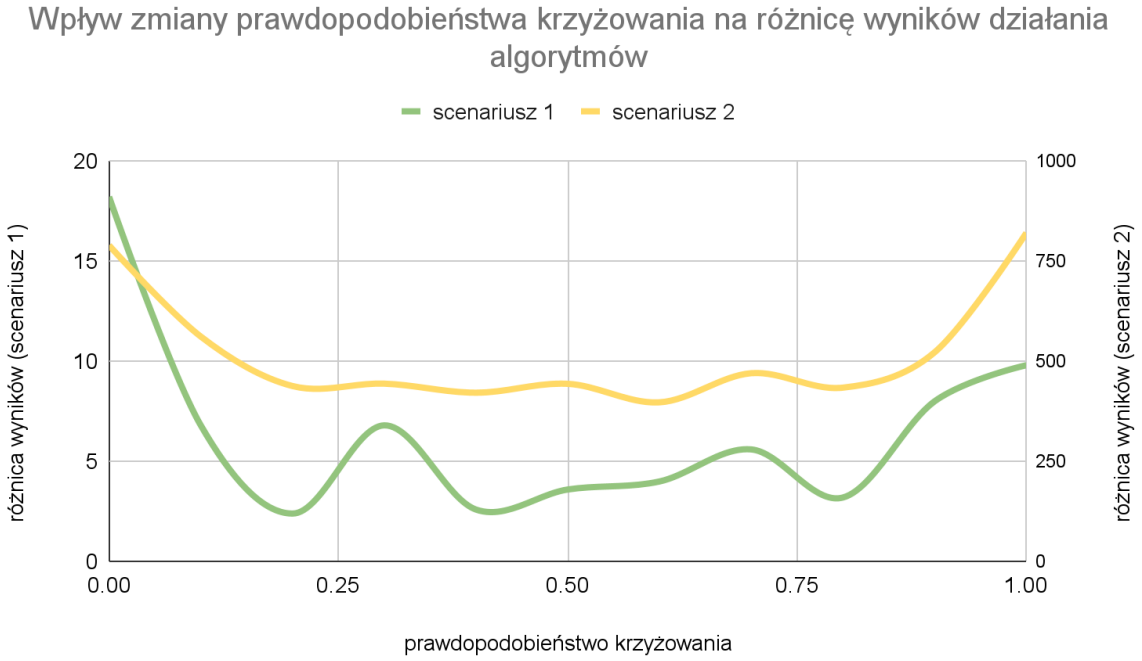


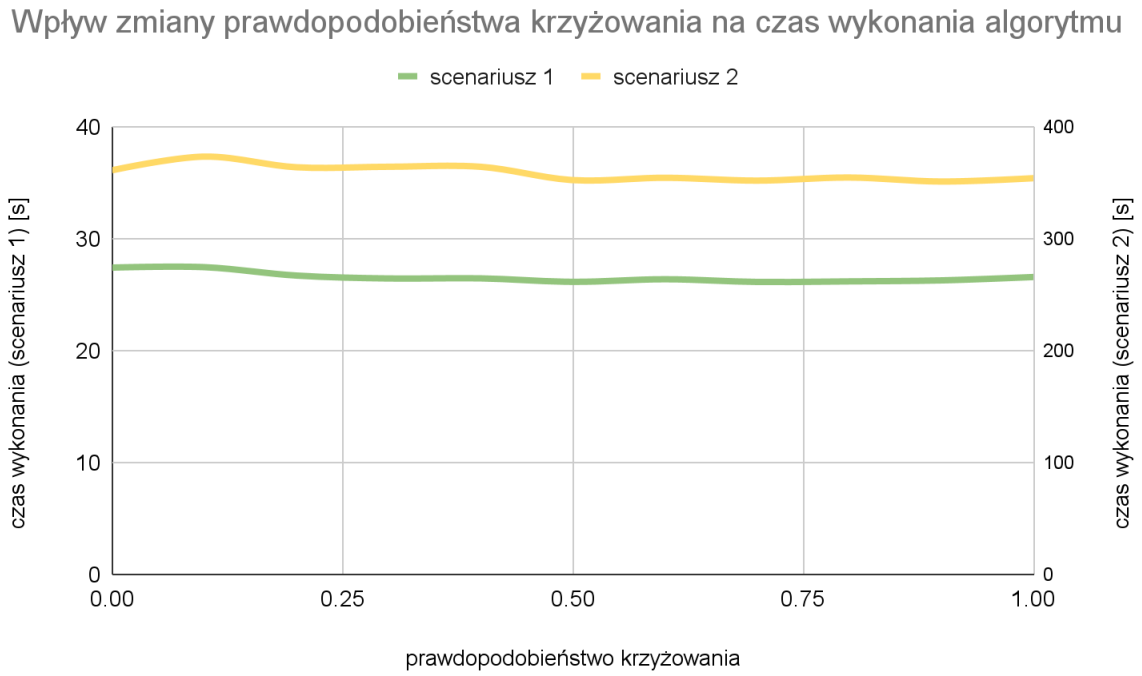
Wnioski:

- Wzrost liczby generacji powoduje poprawę wyników algorytmu ewolucyjnego. Po osiągnięciu pewnego pułapu dalsze zwiększanie liczby generacji nie wpływa na wzrost jakości.
- Dla małej liczby generacji algorytm charakteryzuje się najgorszą skutecznością.
- W scenariuszu 1 możemy zaobserwować okresowe pogorszenie wyników algorytmu ewolucyjnego. Było spowodowane przez mutację i krzyżowanie - algorytm znajdował optimum, a następnie, na skutek mutacji i krzyżowania, oddalał się od niego.
- Czas wykonania liniowo zależy od liczby generacji algorytmu ewolucyjnego.

Test 5 - Badanie wpływu prawdopodobieństwa krzyżowania na otrzymywane wyniki i czas wykonania

maksymalna waga	1000	2500
wielkość problemu N	100	250
wielkość populacji	2000	5000
liczba generacji	180	450
prawdopodobieństwo krzyżowania	0.0 - 1.0 (co 0.1)	
prawdopodobieństwo mutacji	0.01	



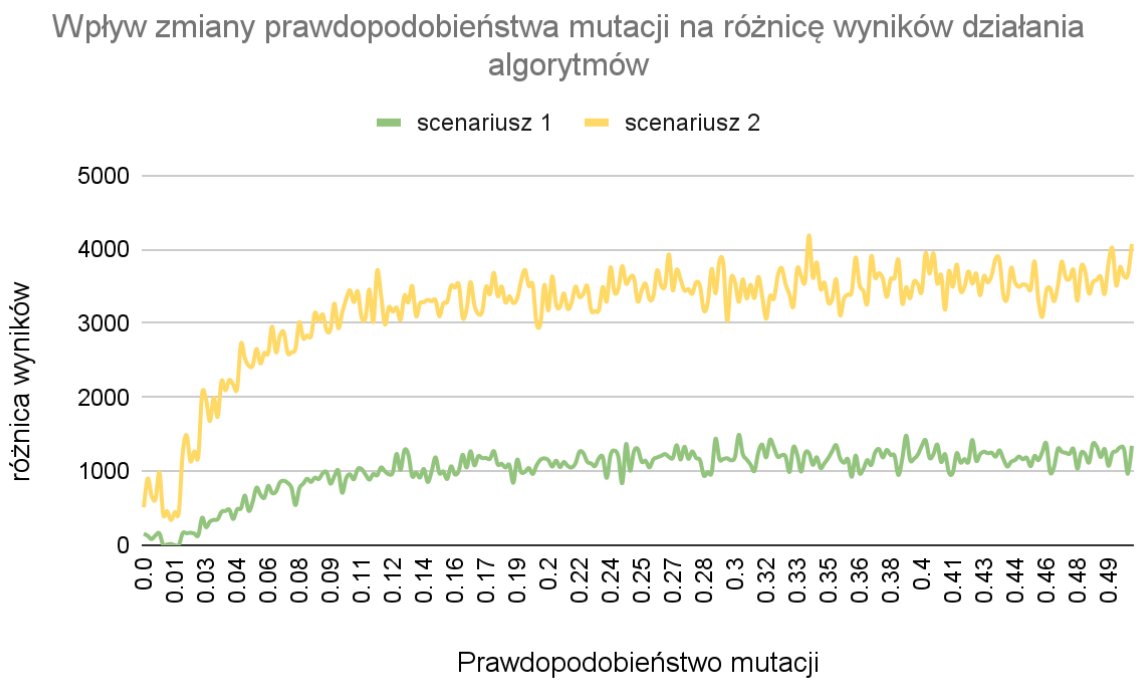


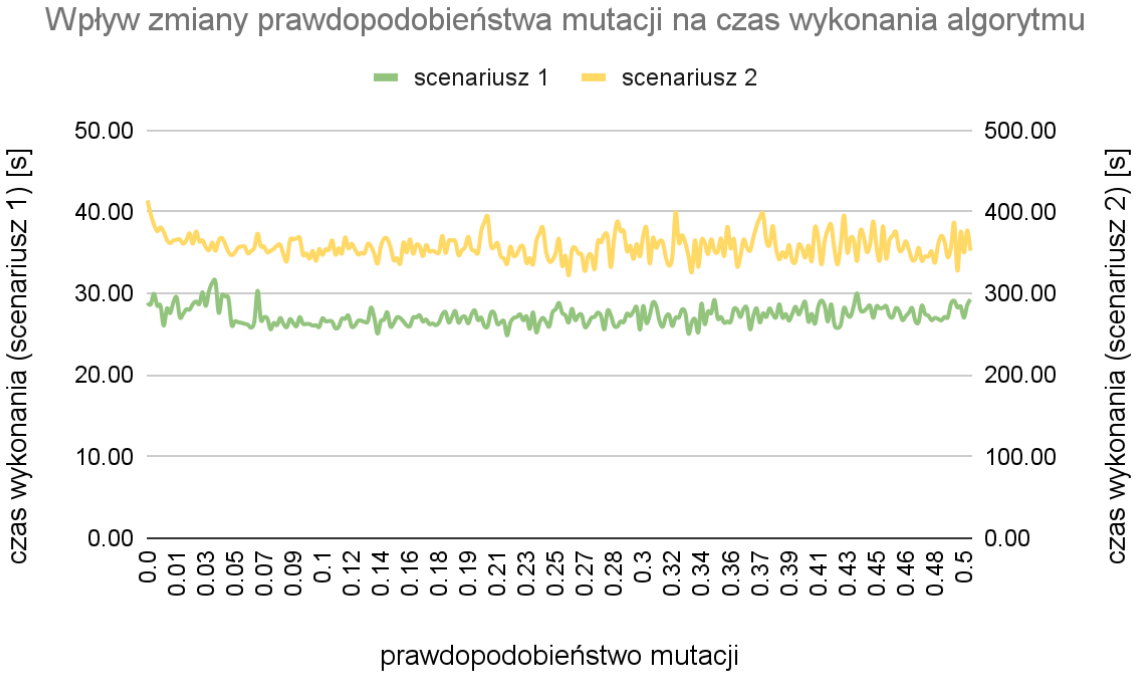
Wnioski:

- Za mała wartość, jak również za duża wartość prawdopodobieństwa krzyżowania przyczynia się do uzyskania gorszych wyników. Dla zbyt małego prawdopodobieństwa krzyżowania (< 10%) algorytm zbyt rzadko krzyżuje osobniki populacji, co może uniemożliwić dojście do optimum globalnego. Z kolei dla zbyt dużego p-stwa krzyżowania (> 90%) algorytm zbyt często krzyżuje osobniki, co może uniemożliwić przejście osobnika o wysokim wyniku do następnej generacji. zamiast tego otrzymujemy średnią z populacji.
- Zmiana prawdopodobieństwa krzyżowania nie wpływa na czas wykonania algorytmu

Test 6 - Badanie wpływu prawdopodobieństwa mutacji na otrzymywane wyniki i czas wykonania

maksymalna waga	1000	2500
wielkość problemu N	100	250
wielkość populacji	2000	5000
liczba generacji	180	450
prawdopodobieństwo krzyżowania	0.25	
prawdopodobieństwo mutacji	0.00 - 0.5 (co 0.01)	





Wnioski:

- Zbyt duża wartość prawdopodobieństwa mutacji skutkuje pogorszeniem otrzymywanych wyników algorytmu ewolucyjnego. Dla wysokiego prawdopodobieństwa mutacji (>~1%) dobre rozwiązania ulegają ciągłym zmianom poprzez proces mutacji, przez co utrudnione jest znalezienie optimum.
- Zerowe prawdopodobieństwo mutacji może uniemożliwić dojście do optimum.
- Optymalna wartość prawdopodobieństwa mutacji dla problemu złodzieja znajduje się w przedziale (0, 0.02).
- Prawdopodobieństwo mutacji nie wpływa na czas wykonania algorytmu.

Podsumowanie

Algorytm ewolucyjny w większości przypadków oferował gorsze rozwiązania niż algorytm plecakowy zarówno pod względem wyliczonego wyniku, jak również ze względu na czas wykonania.

Główną wadą algorytmu plecakowego jest jego złożoność pamięciowa i obliczeniowa, które wynoszą N*X. Algorytm plecakowy bardzo niekorzystnie skaluje się dla bardzo dużych wartości X. Złożoność pamięciowa i obliczeniowa algorytmu ewolucyjnego jest niezależna od zadanej wartości X. Jednak znaczący wpływ na czas jego wykonania ma zmiana wielkości problemu N, która skutkuje potrzebą dostrojenia parametrów algorytmu.
W przeciwnym razie otrzymane wyniki będą znacząco różnić się od optimum. Nie bez wpływu pozostaje liczba generacji i wielkość populacji. Ich niezależna zmiana liniowo wpływa na czas wykonania algorytmu.

Główną zaletą algorytmu plecakowego jest pełne zbadanie przestrzeni przeszukiwań. Dzięki temu możliwe jest wyznaczenie optymalnego rozwiązania. Ideą algorytmu ewolucyjnego jest

przeszukiwanie przestrzeni rozwiązań w sposób niedeterministyczny. Dzięki temu nie jest koniecznie zbadanie pełnej przestrzeni przeszukiwań kosztem uzależnienia jakości rozwiązań od parametrów wpływających na eksplorację i eksploatację badanej przestrzeni. Oznacza to, że może okazać się niemożliwym znalezienie optimum globalnego rozwiązania. Algorytm ewolucyjny może posłużyć do estymacji wyników rozwiązania dla wielkości problemów, dla których pełne zbadanie przestrzeni przeszukiwań jest zbyt kosztowne obliczeniowo i/lub czasowo.

Chcąc efektywnie wykorzystać algorytm ewolucyjny należy go odpowiednio dostroić do rozmiaru i charakterystyki problemu. Na podstawie przeprowadzonych badań zostały wyciągnięte następujące wnioski:

1. Zbyt duże prawdopodobieństwo mutacji prowadzi do utraty zdolności eksploatacji przestrzeni przeszukiwań przez algorytm ewolucyjny. Optymalna wartość prawdopodobieństwa mutacji wyznaczona eksperymentalnie wynosi ok. 1%.
2. Zbyt małe bądź zbyt duże prawdopodobieństwo krzyżowania prowadzi do otrzymania gorszych wyników.
3. Zwiększenie liczby generacji do pewnej wartości prowadzi do polepszenia wyników. Powyżej niej poprawa otrzymywanych wyników jest mało zauważalna, a czas wykonania znacznie dłuższy.
4. Wzrost wielkości populacji oraz liczby generacji powoduje zmniejszenie różnicy wartości. Algorytm ewolucyjny może znaleźć lepsze rozwiązanie kosztem dłuższego czasu jego wykonania.