AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ FIZYKI I INFORMATYKI STOSOWANEJ

KATEDRA INFORMATYKI STOSOWANEJ I FIZYKI KOMPUTEROWEJ

Projekt dyplomowy

Wykorzystanie algorytmu genetycznego do optymalizacji podróży

Application of a genetic algorithm for travel optimization

Autor:	Krzysztof Adam Hrycalik
Kierunek studiów:	Informatyka Stosowana
Opiekun pracy:	dr hab. inż. Małgorzata Krawczyk

Kraków, 2023

Spis treści

1 Wprowadzenie

Mimo bardzo dynamicznego rozwoju sprzętu komputerowego pozwalającego na wykonywanie obliczeń z szybkością niemożliwą do osiągnięcia przez człowieka wciąż istnieją problemy, których rozwiązanie może zająć niewyobrażalnie dużo czasu. Mogą to być problemy charakteryzujące się złożonością obliczeniową n-silnia, czyli na przykład problem komiwojażera [1]. Innym rodzajem złożoności charakteryzującej się szybkim wzrostem ilości możliwych przypadków jest złożoność wykładnicza, którą można zaobserwować w przypadku problemu plecakowego [2].

W celu optymalizacji ilości wykonywanych obliczeń stosuje się różne algorytmy heurystyczne [3], czyli takie, które aby zmniejszyć liczbę obliczeń poświęcają gwarancję otrzymania optymalnego wyniku. Algorytmy genetyczne [4] są metodami z pogranicza informatyki oraz biologii. Korzystając z podstawowych założeń ewolucji biologicznej wypełniają założenia metod heurystycznych. Zostały one sformułowane w latach 60. XX wieku przez Johna Hollanda [5], a ich biologiczne inspiracje znajdują odzwierciedlenie w strukturze algorytmu.

1.1 Cel pracy

Celem pracy jest zrozumienie działania algorytmu genetycznego oraz analiza wpływu zmian parametrów modelu oraz wykorzystywanych w nim metod doboru na wynik. Jako że jest to algorytm heurystyczny, nie ma pewności co do poprawności otrzymanego wyniku, lecz poprzez odpowiednią konfigurację można się do niej zbliżyć.

Algorytmy genetyczne mogą posłużyć jako narzędzie optymalizujące dla problemu komiwojażera (ang. *travelling salesman problem* TSP) oraz problemu plecakowego (ang. *Knapsnack problem*). Wymaga to odpowiedniej konfiguracji modelu, która mocno zależy od rozwiązywanego zagadnienia.

2 Teoria

Program realizujący wcześniej wspomniane cele został zaimplementowany przy użyciu języka programowania Python [6], który korzysta z dynamicznego typowania, co korzystnie wpływa na przejrzystość kodu. Duża liczba dostępnych bibliotek zapewnia elastyczność pozwalającą na rozwiązywanie różnorodnych problemów w dogodny sposób. Jedną z dostępnych bibliotek jest Matplotlib [7], który umożliwia tworzenie wizualizacji zarówno statycznych, animowanych, jak i interaktywnych. Innym pomocnym narzędziem współpracującym z Pythonem jest Pickle [8], który poprzez protokoły binarne pozwala na serializację oraz deserializację obiektów. Umożliwia to wygodny zapis danych do pliku, a następnie ich odczyt do dalszych operacji, na przykład wizualizacji. Aplikacja została utworzona przy wykorzystaniu środowiska PyCharm [9].

2.1 Idea algorytmów genetycznych

Algorytmy genetyczne [4] skupiają się na problemie optymalizacji, czyli na poszukiwaniu najlepszego elementu ze zbioru dostępnych opcji. Funkcja f [5]:

$$f : S \to R \tag{1}$$

umożliwia formalne przedstawienie powyższego problemu. Odwzorowuje ona elementy ze zbioru rozwiązań S, na zbiór liczb rzeczywistych. Dla problemu maksymalizacji szukany jest element $x' \in S$ taki, że

$$f(x') \geqslant f(x)\ \forall\ x \in S \tag{2}$$

oraz alternatywnie dla minimalizacji oczekuje się, że

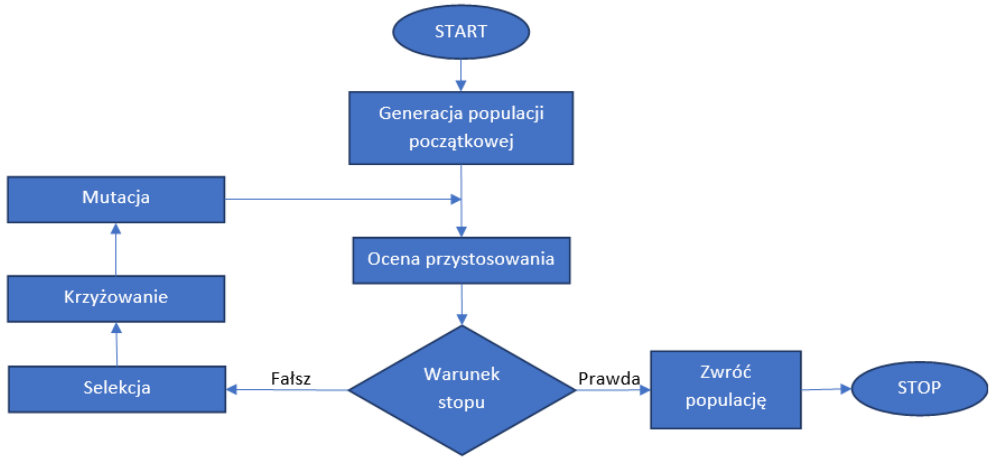
$$f(x') \leqslant f(x)\ \forall\ x \in S \tag{3}$$

Funkcja f nazywana jest funkcją celu (ogólnie), funkcją przystosowania (przy maksymalizacji) lub funkcją kosztu (przy minimalizacji).

Podstawowym założeniem powyższego algorytmu jest istnienie *populacji*, czyli zbio-

ru rozwiązań danego problemu, który jest rozwijany przy użyciu różnorodnych mechanizmów kooperacji oraz rywalizacji. Populacja składa się z elementów zwanych chromosomami, czyli zakodowanej postaci potencjalnego rozwiązania, zapisanego jako ciąg lub łańcuch uporządkowanych genów. W klasycznej wersji algorytmu genetycznego (ang. *Classic Genetic Algorithm CGA* [10]) wartości genów, tzw. *allele*, przyjmują wartości 0 lub 1. Kolejnym powiązaniem algorytmu genetycznego z naturalnym procesem ewolucji jest krzyżowanie. Zaczyna się ono od doboru osobników w pary (osobnik A poszukuje jak najlepszego osobnika B), które to poprzez wydanie potomstwa stają się rodzicami. Proces ten znajduje wierne odwzorowanie w algorytmie genetycznym, gdzie z dostępnej populacji poprzez metodę selekcji następuje wybór przyszłych rodziców, którzy w wyniku operacji krzyżowania wydają potomstwo. Dany nowy osobnik może ulec mutacji, co również jest zgodne procesem ewolucji spotykanym w przyrodzie. Oba te procesy sprowadzają się do tego, aby każdy kolejnym potomek był jak najlepiej dopasowany do środowiska w jakim się znajduje.

Istotne dla algorytmu genetycznego pojęcia zostały przedstawione oraz wytłumaczone w Tab. 1. Na Rys. 1 przedstawiony został schemat przebiegu algorytmu genetycznego.



Rysunek 1: Schemat przebiegu algorytmu genetycznego (za [5])

Tabela 1: Podstawowe pojęcia stosowane w algorytmach genetycznych

pojęcie	**definicja**
Chromosom	uporządkowany ciąg genów, zakodowana postać potencjalnego rozwiązania (punkt z przestrzeni poszukiwań)
Gen	pojedynczy element chromosomu
Allel	wartość genu (w klasycznej wersji algorytmu genetycznego 0 lub 1)
Locus	pozycja genu w chromosomie
Genotyp	zbiór chromosomów danego osobnika (w klasycznej wersji algorytmu genetycznego sprowadza się do pojedynczego chromosomu)
Fenotyp	osobnik w formie niezakodowanej; punkt w przestrzeni poszukiwań, którego zakodowaną postacią jest chromosom (genotyp)
Osobnik	potencjalne rozwiązanie przedstawiane w postaci genotypu lub fenotypu
Populacja	zbiór osobników o określonej liczebności
Przystosowanie	przystosowanie osobników w populacji, definiowane za pomocą funkcji przystosowania (ang. *fitness function*)
Selekcja	wybór osobników do populacji rodzicielskiej, biorącej udział w rekombinacji genów
Krzyżowanie	proces rekombinacji genów, w oparciu o wymianę fragmentów chromosomów rodziców, prowadzący do powstania nowych osobników (potomków)
Rodzic	chromosom łączący się w parę z drugim w celu krzyżowania
Mutacja	proces modyfikacji chromosomu; w klasycznej wersji algorytmu genetycznego sprowadza się do zamiany wybranego genu z 1 na 0 lub odwrotnie
Potomek	chromosom powstały w wyniku operacji krzyżowania oraz mutacji

2.2 Problem plecakowy

Problem plecakowy skupia się na poszukiwaniu podzbioru zbioru elementów zawierającego przedmioty, których sumarycznie wartość jest największa oraz jednocześnie mieszczą się one w plecaku (w oparciu o ich wagę lub rozmiar). Wartość przedmiotu (oznaczona dalej przez v), rozumiana jest jako "jak bardzo zależy nam na danym przedmiocie", aby go ze sobą zabrać.

Jeśli każdy przedmiot jest dostępny maksymalnie w jednej sztuce ($x_i \in \{0, 1\}$)

Tabela 2: Przykładowy zbiór dostępnych przedmiotów

lp.	**przedmiot**	**wartość *v***	**waga *w***
1	laptop	500	2500
2	słuchawki	700	100
3	chusteczki	300	50
4	telefon	900	350
5	zeszyt	100	200
6	butelka wody	950	1500
7	skarpetki	150	150
8	bluza	200	450
9	konsola	10	3750
10	cegła	1	5000

oraz potencjalne rozwiązanie zostanie zapisane w formie binarnej, to występująca w nim wartość 1 oznacza, że dany element zbioru zawiera się w podzbiorze stanowiącym rozwiązanie, a 0 że się w nim nie zawiera. Można zatem zauważyć, że gdy istnieją tylko dwie opcje dla każdego z *n* elementów, to złożoność obliczeniowa takiego problemu jest rzędu $\Theta(2^n)$. Problem ten można zapisać formalnie jako maksymalizację sumy iloczynów:

$$\sum_{i=1}^{n} v_i x_i, \tag{4}$$

gdzie v oznacza wartość, a x liczbę sztuk danego przedmiotu.

Jednocześnie żądamy spełnienia warunku:

$$\sum_{i=1}^{n} w_i x_i \leqslant W, \tag{5}$$

gdzie W jest maksymalną całkowitą dopuszczalną wagą plecaka.

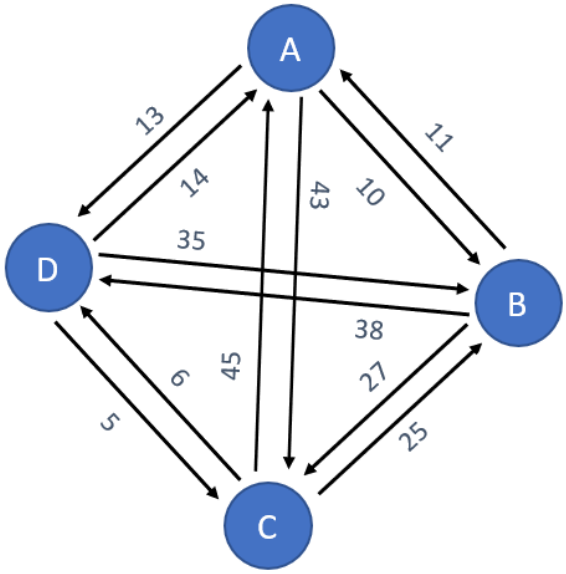
Tabela 3: Przykładowy zapis podzbioru dla problemu plecakowego.

lp.	1	2	3	4	5	6	7	8	9	10
wartość allela	0	1	1	1	1	1	1	1	0	0

Dla przykładowego zestawu przedmiotów, przedstawionego w Tab. 2, optymalne rozwiązanie przedstawione zostało w Tab. 3, gdzie limit wagowy wynosił 3000.

2.3 Problem komiwojażera

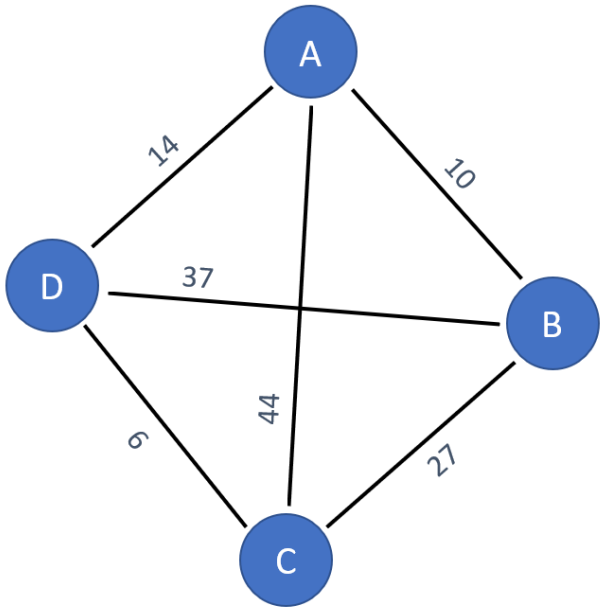
Problem komiwojażera dotyczy przypadku, w którym osoba podróżująca ma do odwiedzenia określoną liczbę miast. Przy założeniu, że znane są odległości pomiędzy każdą dowolnie wybraną parą miast, każde miasto może zostać odwiedzone dokładnie raz oraz należy wrócić do punktu początkowego, szukamy optymalnego rozwiązania, np. minimalizując długość trasy. Maksymalna liczba możliwych sposobów na zrealizowanie trasy wynosi $(n-1)!$, gdzie n jest liczbą miejsc do odwiedzenia. W realnym świecie możemy mieć do czynienia z ulicami jednokierunkowymi, które powodują, że droga w jedną stronę może mieć inną długość niż powrotna. Powoduje to, że graf przedstawiający powyższy problem jest grafem skierowanym (Rys. 2).



Rysunek 2: Przykładowy graf odległości pomiędzy czterema miejscami

.

Gdyby założyć, że długość drogi z punktu A do B i z punktu B do A wynosi tyle samo (Rys. 3), to liczba możliwych tras zredukuje się do $(n-1)!/2$. Złożoność tego problemu wynosi zatem $\Theta(n!)$.



Rysunek 3: Przykładowy graf odległości pomiędzy czterema lokalizacjami przy założeniu jednakowych odległości w obu kierunkach

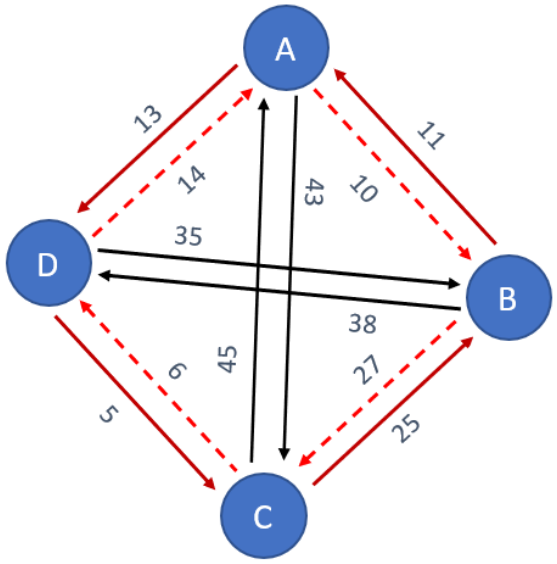
.

W tym przypadku problem sprowadza się do minimalizacji (wzór 3), a zetem poszukiwany wynik powinien być jak najmniejszy. Wówczas nie mamy do czynienia z funkcją przystosowania, a z funkcją kosztu.

2.4 Algorytm najbliższego sąsiada

Częstym wyborem podejmowanym przez człowieka podczas samodzielnego określania kolejności miejsc do odwiedzenia jest wybór najbliższego z możliwych, dlatego też algorytm najbliższego sąsiada może posłużyć do porównania z algorytmem genetycznym. Algorytm najbliższego sąsiada jest algorytmem zachłannym [11], który w celu znalezienia rozwiązania dokonuje wyboru najlepiej rokujących danych w każdym kroku. Wybór lokalnie optymalny pomija ocenę czy dana decyzja ma pozytywny wpływ na kolejne kroki czy nie. Selekcja zachłanna pozbawiona jest aspektu losowości, a zatem zawsze dla tych samych danych daje ten sam wynik. Nie ma również gwarancji, że otrzymane rozwiązanie jest optymalne.

Algorytm najbliższego sąsiada może zostać użyty do rozwiązania problemu komiwojażera. Polega on na wyborze najbliższego, jeszcze nieodwiedzonego miejsca. Algorytm ten charakteryzuje się złożonością czasową rzędu $O(n^2)$ [12]. Porównanie wyniku znalezionego za pomocą algorytmu najbliższego sąsiada z rozwiązaniem optymalnym przedstawione zostało na Rys. 4. Kolorem bordowym zaznaczono rozwiązanie optymalne, a czerwoną linią przerywaną rozwiązane znalezione za pomocą algorytmu najbliższego sąsiada.



Rysunek 4: Porównanie drogi optymalnej z otrzymaną za pomocą algorytmu najbliższego sąsiada

3 Implementacja

3.1 Problem plecakowy

W celu rozwiązania problemu plecakowego zaimplementowany został algorytm genetyczny, zgodnie ze schematem przedstawionym na Rys. 1. W pierwszym kroku należało utworzyć populację początkową (Listing 1), czyli zbiór składający się z określonej liczby osobników. Jako że w tym przypadku istnieją jedynie dwie opcje, "zawiera się w plecaku" reprezentowane jako 1 oraz "nie zawiera się w plecaku" reprezentowane jako 0, daje to

możliwość skorzystania z reprezentacji binarnej osobników. Każdy z przedmiotów branych pod uwagę będzie miał w chromosomie swój allel przyjmujący tylko jedną z dwóch możliwych wartości.

```
def generate_genom(length: int) -> Genome:
    return random.choices([0, 1], k=length)

def generate_populate(size: int, genome_length: int) -> Population:
    return [generate_genom(genome_length) for _ in range(size)]
```

Listing 1: Sposób generowania osobnika oraz całej populacji

Korzystając z wcześniejszego założenia, że allel przyjmuje jedynie wartości 1 lub 0 generowana jest w sposób losowy lista o zadanym rozmiarze *length*, będąca reprezentacją pojedynczego osobnika. Operacja ta powtarzana jest zadaną liczbę razy *size* oraz zapisywana do listy co w końcowym efekcie tworzy zbiór osobników, czyli populację początkową.

Następnym etapem jest dokonanie oceny przystosowania realizowane w sposób przedstawiony na listingu 2.

```
def fitness(genome: Genome, things: [Thing], weight_limit: int) -> int
    :
    if len(genome) != len(things):
        raise ValueError("Genome and things must be of the same length
    ")

    weight = 0
    value = 0

    for i, thing in enumerate(things):
        if genome[i] == 1:
            weight += thing.weight
            value += thing.value

            if weight > weight_limit:
                return 0
```

```
15      return value
```

Listing 2: Funckja przystosowania dla problemu plecakowego

Funkcja przystosowania sumuje w tym przypadku wartości oraz wagi przedmiotów, dla których allel w chromosomie przyjmuje wartość 1, rozumianą jako "znajduje się w plecaku". Korzystając z informacji o maksymalnej dopuszczalnej wadze odrzucamy te osobniki (zestawy przedmiotów), dla których ciężar jest zbyt duży. Powoduje to, że ich szansa wyboru do następnej generacji wynosi 0, a co za tym idzie, nie będą one w ogóle brane pod uwagę przy operacji selekcji.

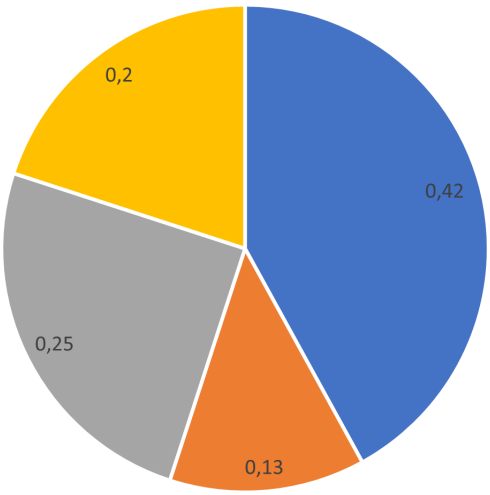
Kontynuując algorytm zgodnie z Rys. 1 kolejnym krokiem jest *warunek stopu* rozumiany jako moment, w którym przerywany jest algorytm oraz zwracany jest najlepszy dotychczas znaleziony chromosom. Warunek ten został zaimplementowany jako ograniczona liczba generacji.

```
1 for i in range(generation_limit):
```

Listing 3: Warunek stopu

Po zakończeniu pętli (Listing 3) następuje ostatnie sortowanie populacji, w oparciu o funkcję przystosowania (Listing 2), oraz jej zwracanie. Dzięki takiemu zabiegowi osobnik najlepiej spełniający założenia problemu plecakowego, czyli jak największa wartość oraz waga nieprzekraczająca dopuszczanego limitu, znajduje się jako pierwszy element listy.

Jeżeli warunek stopu nie jest osiągnięty, następuje faza selekcji. Faza ta realizowana jest za pomocą metody ruletki [5], gdzie każdemu z n osobników zostaje przydzielony odpowiedni sektor (Rys. 5) o wielkości proporcjonalnej do wartości funkcji przystosowania f (Listing 2).



Rysunek 5: Przykładowy podział koła ruletki na sektory oraz odpowiadające im prawdopodobieństwa

Jako że dany osobnik x może być wybrany wiele razy, miejsce osobników *słabszych* zajmują *mocniejsi*. Sektory na kole ruletki odpowiadające za prawdopodobieństwo p wyboru osobnika x określane są wzorem:

$$p_i = \frac{f(x_i)}{\sum_{q=1}^{n} f(x_q)} \tag{6}$$

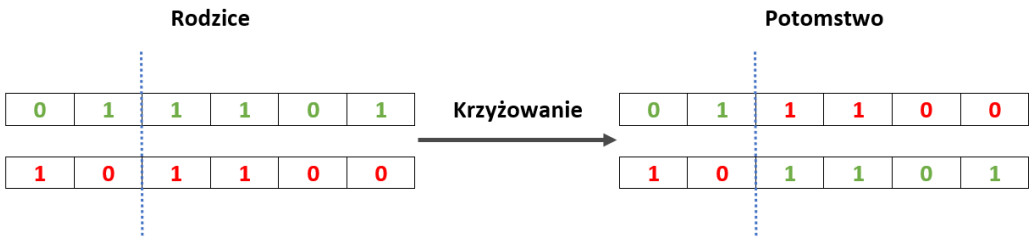
```
def selection_pair(population: Population, weights: List[float]) ->
    Population:
    return choices(
        population=population,
        weights=weights,
        k=2
    )
```

Listing 4: Wybór rodziców metodą ruletki

Zastosowanie takiej metody selekcji skutkuje częstszym wyborem do populacji rodzicielskiej chromosomów o lepszym przystosowaniu.

Zaimplementowany algorytm korzysta również z selekcji metodą elitarną [5], która pomaga przyśpieszyć proces poszukiwania optymalnego rozwiązania [5]. Polega ona na zachowaniu określonej liczby osobników do kolejnej generacji bez żadnych zmian w ich chromosomie. W pracy przyjęto, że w każdym kroku dwa najlepiej przystosowane osobniki przechodzą do kolejnej generacji. Pozostałe, dla utrzymania wielkości populacji, tworzone są w oparciu o osobniki z obecnie istniejącej generacji. Spośród tych osobników operator selekcji wybiera dwóch, którzy mogą zostać poddani krzyżowaniu oraz mutacji, a następnie wcieleni do kolejnej generacji. Proces ten jest powtarzany aż do osiągnięcia wielkości populacji równiej tej z obecnej generacji.

Posiadając osobniki wybrane w operacji selekcji, można dokonać ich krzyżowania. W zaimplementowanym algorytmie wykorzystano krzyżowanie jednopunktowe (ang. *Single point crossover*) [5], którego przykładowy przebieg został przedstawiony na Rys. 6.



Rysunek 6: Przykład krzyżowania jednopunktowego

.

Miejsce oznaczone niebieską przerywaną linią oznacza punkt przecięcia, który wybierany jest w sposób losowy (Listing 5).

```
1 def single_point_crossover(a: Genome, b: Genome) -> Tuple[Genome,
    Genome]:
2     if len(a) != len(b):
3         raise ValueError("Both genomes must have same length")
4
5     length = len(a)
```

```
    if length < 2:
        return a, b

    p = randint(1, length - 1)
    return a[0:p] + b[p:], b[0:p] + a[p:]
```

Listing 5: Realizacja krzyżowania jednopunktowego

Każdy z potomków otrzymanych w wyniku operacji krzyżowania może zostać poddany mutacji. Proces ten można formalnie przeprowadzić dla dowolnej liczby alleli. Dla rozwiązania problemu plecakowego operacja ta została zaimplementowana w sposób wpływający jedynie na jedną wartość. Korzystając z binarnej formy zapisu chromosomu wprowadzenie mutacji oznacza zmianę wartości z 1 na 0 lub odwrotnie.

```
def mutation(genome: Genome, num: int = 1, probability: float = 0.5)
    -> Genome:
    for _ in range(num):
        index = randrange(len(genome))
        genome[index] = genome[index] if random() > probability else
    abs(genome[index] - 1)

    return genome
```

Listing 6: Proces mutacji

.

Operacje począwszy od selekcji (Listing 4), przez krzyżowanie (Listing 5), aż do mutacji (Listing 6) powtarzane są do momentu, gdy zostanie wygenerowana odpowiednia liczba potomków, równa wielkości populacji początkowej. Następnie, zostanie sprawdzony warunek stopu i w zależności od rezultatu wcześniejsze operacje zostaną powtórzone lub zostanie zwrócony najlepiej przystosowany osobnik.

3.2 Problem komiwojażera

Podobnie jak w przypadku problemu plecakowego, pierwszym etapem algorytmu dla problemu komiwojażera jest utworzenie populacji początkowej (Listing 7).

```python
def generate_genom(length: int) -> Genome:
    Genome = [0]
    for i in range(length):
        while True:
            if len(Genome) == length:
                break
            city = randint(1, length - 1)
            if city not in Genome:
                Genome.append(city)
    Genome.append(0)
    return Genome

def generate_populate(size: int, genome_length: int) -> Population:
    return [generate_genom(genome_length) for _ in range(size)]
```

Listing 7: Generacja populacji początkowej dla problemu komiwojażera

Zauważyć można, że proces ten przebiega inaczej niż dla wcześniej omawianego problemu. Wynika to z odmiennej formy prezentacji rozwiązania, która uniemożliwia korzystanie z wcześniejszego założenia dla pojedynczego elementu jako "spełnia warunek" lub "nie spełnia warunku". W tym problemie kluczową rolę odgrywa zależność między poprzednim, a następnym allelem reprezentowanym przez liczbę całkowitą mającą bezpośrednie odniesienie do konkretnego odwiedzanego miejsca.

Tabela 4: Przykładowy chromosom problemu komiwojażera dla mapy o rozmiarze 5

0	4	2	1	3	0

W Tab. 4 wartość 0 powtórzona jest dwa razy co wynika z założenia powrotu do lokacji początkowej oznaczonej właśnie tą cyfrą. Wpływa to również na rozmiar chromosomu, który jest równy $n + 1$, gdzie *n* oznacza liczbę miast. Pozostałe liczby całkowite wskazują kolejność odwiedzania lokalizacji przez nie wskazywanych. Mogą się one pojawić w genotypie tylko raz. Program wymaga utworzenia macierzy odległości (ang. *distance matrix*) o wymiarach nxn, gdzie numer wiersza odpowiada obecnemu miejscu, a numer kolumny miejscu docelowemu.

A \ B	0	1	2	3	4
0	0	5	9	22	4
1	5	0	15	31	27
2	9	15	0	11	19
3	22	31	11	0	10
4	4	27	19	10	0

Tabela 5: Przykładowa macierz odległości z punktu A do punktu B dla 5 miast

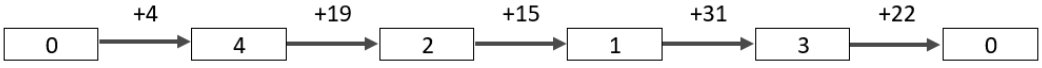
Na podstawie Tab. 5 można by wywnioskować, że nie ma znaczenia czy idziemy np. z punktu A do punktu B czy na odwrót, gdyż odległości są takie same. Jest tak za sprawą wyidealizowanego przypadku. W rzeczywistości trasy w obie strony nie zawsze są równoważne. Dzieje się tak choćby z powodu ulic jednokierunkowych. Ważne jest więc, aby ten fakt uwzględnić podczas tworzenia macierzy odległości.

Inny sposób zapisu chromosomu wymaga również innego sposobu oceny przystosowania. Jako że problem przyjmuje postać minimalizacji zamiast z funkcją przystosowania, mamy do czynienia z funkcją kosztu.

```
1 def fitness(genome: Genome, mapa: List[List[int]]) -> int:
2     if len(genome) != len(mapa) + 1:
3         raise ValueError("Genome and map must be of the same length",
    len(genome), "  ", len(mapa))
4
5     return sum(mapa[genome[i]][genome[i + 1]] for i in range(len(
    genome) - 1))
```

Listing 8: Funkcja kosztu dla problemu komiwojażera

Koszt obliczany jest jako suma odległości pomiędzy miejscami, których kolejność odwiedzenia definiuje dany genotyp (Rys. 7). Analizując listing 8 można zauważyć, że



Rysunek 7: Sposób zliczania drogi dla przykładu przedstawionego w Tab. 4

wyliczenie kosztu dla pojedynczego osobnika wymaga wykonania tylu operacji sumowa-

nia ile jest planowanych miejsc do odwiedzenia. Kolejnym etapem jest analiza warunku stopu, który podobnie jak w przypadku implementacji rozwiązania problemu plecakowego, jest liczbą generacji.

```
for i in range(generation_limit):
```

Listing 9: Warunek stopu dla problemu komiwojażera

W każdej iteracji pętli (Listing 9) dokonywana jest selekcja z wykorzystaniem metody koła ruletki.

```
def selection_pair(population: Population, weights: List[float]) ->
    Population:
    return random.choices(
        population=population,
        weights=weights,
        k=2
    )
```

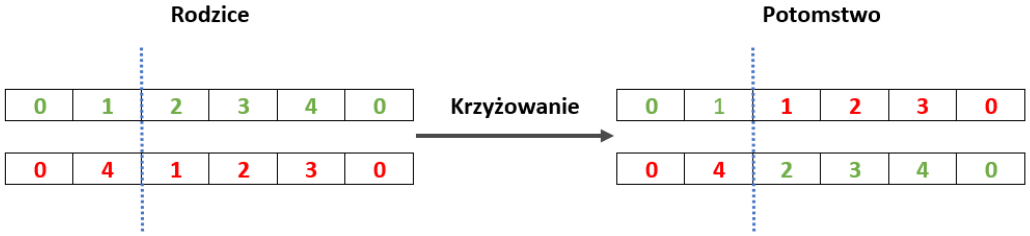
Listing 10: Metoda selekcji dla problemu komiwojażera

Metoda ta wymaga określenia wielkości sektora na kole ruletki proporcjonalnego do wartości kosztu. Sprowadza się to do prawdopodobieństwa z jakim dany osobnik powinien zostać wybrany do dalszych operacji. Funkcja kosztu zwraca długość drogi do przebycia (Listing 8), a co za tym idzie im dłuższa droga tym zwracana wartość jest większa. Przełożenie tej wartości bezpośrednio na prawdopodobieństwo wyboru, spowodowałoby premiowanie dłuższych tras. Problem ten przyjąłby zatem formę maksymalizacji, a nie oczekiwanej wcześniej minimalizacji. Dlatego też wartości te trzeba odpowiednio przekształcić.

```
        ff = {tuple(genome): fitenss_func(genome, mapa) for genome in
    population}
        weight_max = fitenss_func(population[-1], mapa)
        weights = [weight_max - ff.get(tuple(key)) for key in
    population]
```

Listing 11: Przekształcenie wartości przystosowania na wagi do dalszej operacji selekcji

W tym celu został utworzony słownik zawierający ścieżkę oraz jej długość. Następnie znaleziona zostaje najdłuższa trasa, która jest wykorzystywana do przydzielenia osobnikom z populacji odpowiednich wag (Listing 11 linia 3). Operacja ta pozwala przydzielić większe wartości krótszym trasom, a co za tym idzie będą one wybieranie częściej podczas operacji selekcji (Listing 10). Gdy już zostaną wybrani rodzice następuje faza krzyżowania. Ponownie nie można bezpośrednio skorzystać z metody użytej dla problemu plecakowego (Listing 5), gdyż mogłoby to doprowadzić do powstania niewłaściwych osobników. Analizując Rys. 8 można dostrzec, że dostajemy osobniki, w których pewne

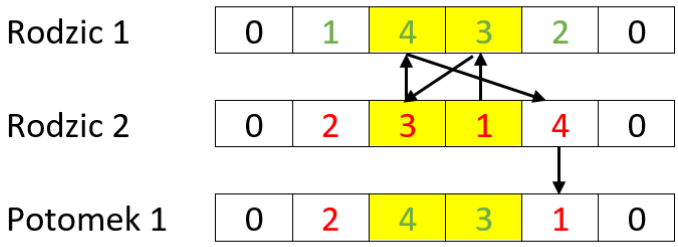


Rysunek 8: Przykład niepoprawnego wyniku operacji krzyżowania jednopunktowego dla problemu komiwojażera

lokalizacje powtarzają się, a niektórych brakuje (w pierwszym potomku lokalizacja oznaczona numerem 1 znajduje się dwukrotnie oraz brakuje lokalizacji oznaczonej numerem 4). Pojawianie się tego typu błędów można rozwiązać na kilka sposobów: można odrzucać wadliwych potomków, naprawiać ich albo zapewnić, żeby w wyniku krzyżowania powstawały zawsze prawidłowe osobniki. Jednym ze sposobów na generowanie zawsze prawidłowych potomków jest metoda *Partially Mapped Crossover*[1] PMX [13].

Przebieg PMX, przedstawiony na Rys. 9, zaczyna się od wyboru przedziału alleli, które będą dziedziczone w sposób niezmieniony (oznaczono je kolorem żółtym). Dane te (z pierwszego rodzica) są przepisywany do pierwszego potomka. Następnie poszukiwane są wartości, które nie powtarzają się w obu żółtych przedziałach. Każdemu takiemu miejscu do odwiedzenia należy znaleźć nową pozycję w osobniku. Sposób poszukiwania nowej pozycji jest jednym z elementów zdefiniowanych przez metodę PMX (przedstawio-

[1]Zgodnie z wiedzą autora nie istnieje polski odpowiednik sformułowania Partially Mapped Crossover, dlatego w pracy użyto angielskiego terminu



Rysunek 9: Przykład krzyżowania metodą PMX

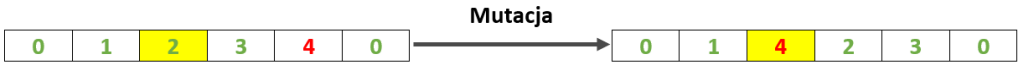
ny jest na Rys. 9 czarnymi strzałkami). Jednym z alleli wymagających przeprowadzenia takiej operacji jest allel o wartości 1, gdyż znajduje się on tylko w drugim rodzicu. W pierwszym etapie sprawdzana jest wartość, która znajduje się na tej pozycji w rodzicu pierwszym. W tym przypadku jest to allel o wartości 3. Jako że znajduje się on w żółtym przedziale należy szukać dalej. Tym razem sprawdzane jest co znajduje się w pierwszym rodzicu na pozycji odpowiadającej pozycji 3 w rodzicu drugim. Jest to 4, która to w rodzicu drugim nie znajduje się w żółtym przedziale, a zatem w to miejsce potomka pierwszego należy wpisać wartość, od której operacja ta się zaczęła, czyli 1. Następnie pozostałe wartości z rodzica drugiego można przepisać do potomka (wartości z żółtego przedziału potomka należą do innego rodzica niż te przepisywane w ostatnim etapie).

```
1  def PMX_crossover(a: Genome, b: Genome) -> Tuple[Genome, Genome]:
2      left, right = random.sample(population=range(1, len(a) - 1), k=2)
3      if left > right:
4          left, right = right, left
5      c1, c2 = b[:left] + a[left:right] + b[right:], a[:left] + b[left:
       right] + a[right:]
6      for i in range(left, right):
7          if not (left <= a.index(b[i]) < right):
8              tmp = i
9              while left <= b.index(a[tmp]) < right:
10                 tmp = b.index(a[tmp])
11             c1[b.index(a[tmp])] = b[i]
12
```

```
13          if not (left <= b.index(a[i]) < right):
14              tmp = i
15              while left <= a.index(b[tmp]) < right:
16                  tmp = a.index(b[tmp])
17              c2[a.index(b[tmp])] = a[i]
18
19      return c1, c2
```

Listing 12: Krzyżowanie metodą PMX

Po etapie krzyżowań następuje etap mutacji, który ponownie za sprawą formy zapisu wymaga skorzystania z innej metody niż było to w przypadku problemu plecakowego. Dzieje się tak, ponieważ nie można zmienić wartości allela na przeciwną. Nie można również zmienić wartości tylko jednego allela, bowiem taki zabieg zaburzyłby poprawność osobnika. Z pomocą przychodzi metoda mutacji insercyjnej, (ang. *insertion mutatuion*) inaczej zwana też mutacją przesuwną [14].



Rysunek 10: Przykład mutacji przesuwnej.

Przykład mutacji przesuwnej przedstawiony został na Rys. 10. Metoda ta zaczyna się od losowego wyboru allela (zaznaczonego kolorem czerwonym), którego wartość należy przenieść na losowo wybrane miejsce (oznaczone kolorem żółtym). Pozostałe wartości na prawo od żółtego pola należy przesunąć o jedną pozycję w prawo. Za sprawą tej techniki mamy pewność, że otrzymany w ten sposób osobnik będzie poprawny.

```
1 def mutation(genome: Genome) -> Genome:
2     index, index2 = random.sample(population=range(1, len(genome) - 1)
      , k=2)
3     if index < index2:
4         return genome[:index] + genome[index2:index2 + 1] + genome[
      index:index2] + genome[index2 + 1:]
5
```

```
6       return genome[:index2] + genome[index:index + 1] + genome[index2:
        index] + genome[index + 1:]
```

Listing 13: Mutacja metodą przesuwną

Podobnie jak w przypadku implementacji algorytmu dla problemu plecakowego, operacje począwszy od selekcji (Listing 10), przez krzyżowanie (Listing 12), aż do mutacji (Listing 13) powtarzane są do momentu, gdy zostanie wygenerowana odpowiednia liczba potomków, równa wielkości populacji początkowej. Następnie, ponownie zostanie wywołana funkcja kosztu, żeby kolejno sprawdzić warunek stopu i w zależności od rezultatu powtórzyć wcześniejsze operacje lub zwrócić najkrótszą znalezioną drogę.

4 Dobór parametrów modelu. Wyniki.

4.1 Problem plecakowy

W celu doboru odpowiednich parametrów zostały wykonane badania wpływu ich wartości na poprawność wyniku. Jako przypadek testowy zostały wykorzystane dane przedstawione w Tab. 6. Dla każdej z wartości badanego parametru wykonano 500 powtórzeń.

Tabela 6: Zbiór dostępnych przedmiotów

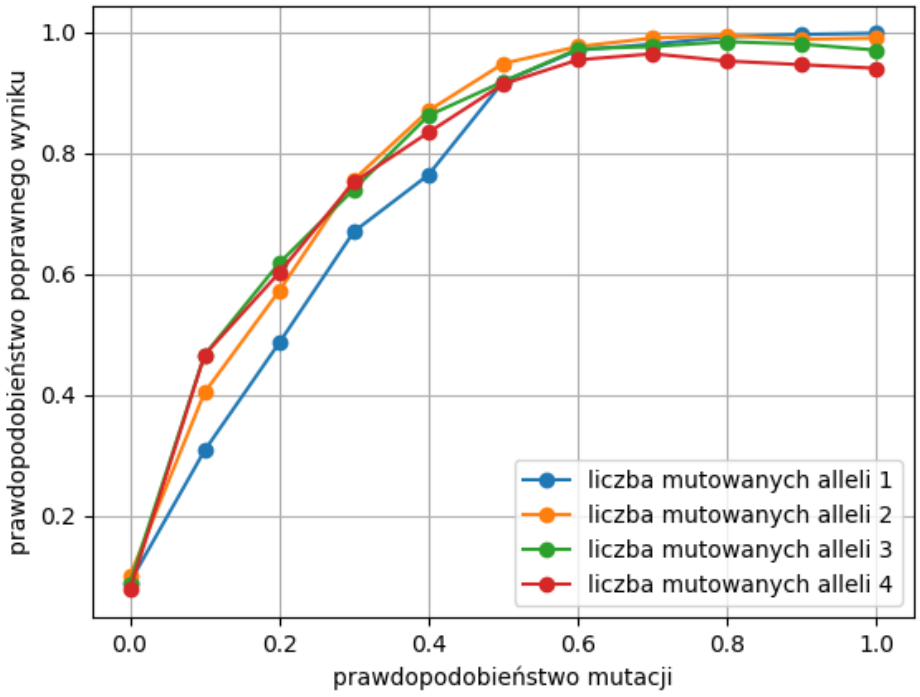
przedmiot	**wartość v**	**waga w**
laptop	500	2500
słuchawki	700	100
chusteczki	300	50
telefon	900	350
zeszyt	100	200
butelka wody	950	1500
skarpetki	150	150
bluza	200	450
konsola	10	3750
cegła	1	5000
jedzenie	700	1000
śpiwór	150	500
herbata	400	700
kask	40	700
ciężarek	5	1000

Dokładna konfiguracja początkowa została przedstawiona w Tab. 7.

Tabela 7: Konfiguracja początkowa

parametr	**wartość**
Liczba generacji	150
Rozmiar populacji	100
Prawdopodobieństwo mutacji	wartość badana
Punkty mutacji	wartość badana
Prawdopodobieństwo krzyżowania	1
Warunek stopu	liczba generacji
Selekcja	koło ruletki, elitarna
Rodzaj mutacji	zmiana pojedynczej wartości
Rodzaj krzyżowania	jednopunktowe
Statystyka	500 powtórzeń
Limit wagowy	3000

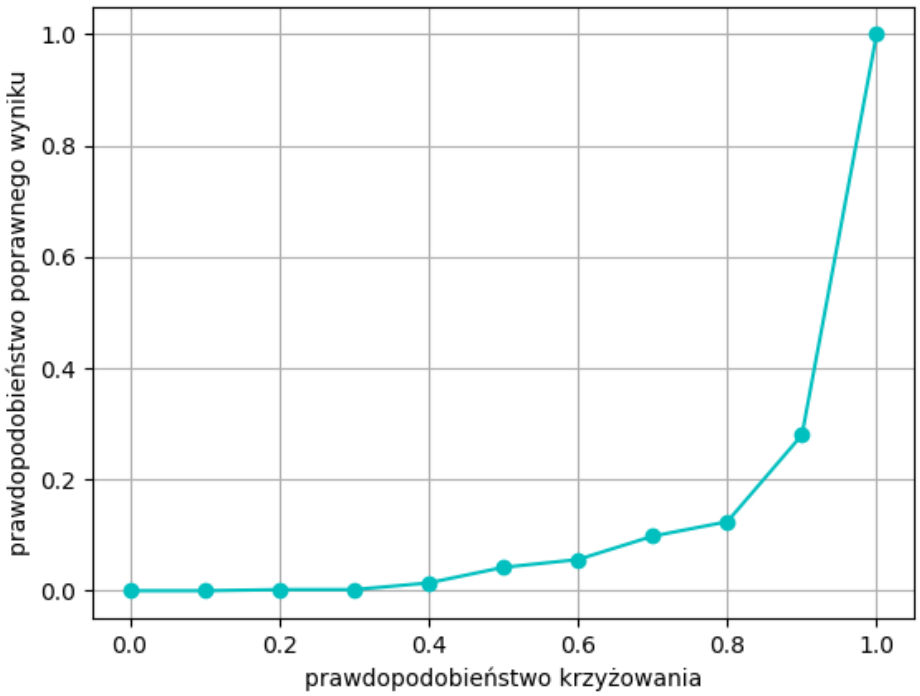
Obliczenia powtórzono dla wartości prawdopodobieństwa mutacji z przedziału $[0, 1]$ z krokiem 0.1 oraz $1 - 5$ punktów mutacji z krokiem 1. Punkty mutacji odpowiadają liczbie alleli, których wartość zostanie zmieniona na przeciwną.



Rysunek 11: Wpływ prawdopodobieństwa oraz liczby punktów mutacji na poprawność wyniku

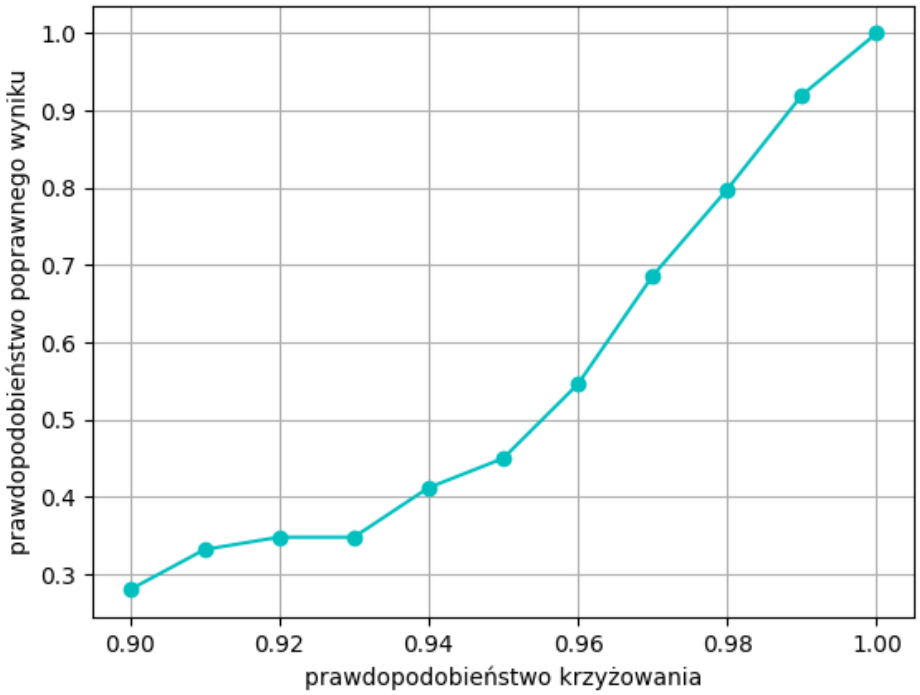
Na podstawie uzyskanych wyników (Rys. 11) stwierdzono, że operacja mutacji wykonywana będzie zawsze, dla jednego allela.

Z tak dobranym parametrem mutacji sprawdzono jaki wpływ ma prawdopodobieństwo krzyżowania na częstotliwość otrzymywania poprawnego wyniku. W pierwszej kolejności sprawdzono prawdopodobieństwo z przedziału $[0, 1]$ z krokiem 0.1.



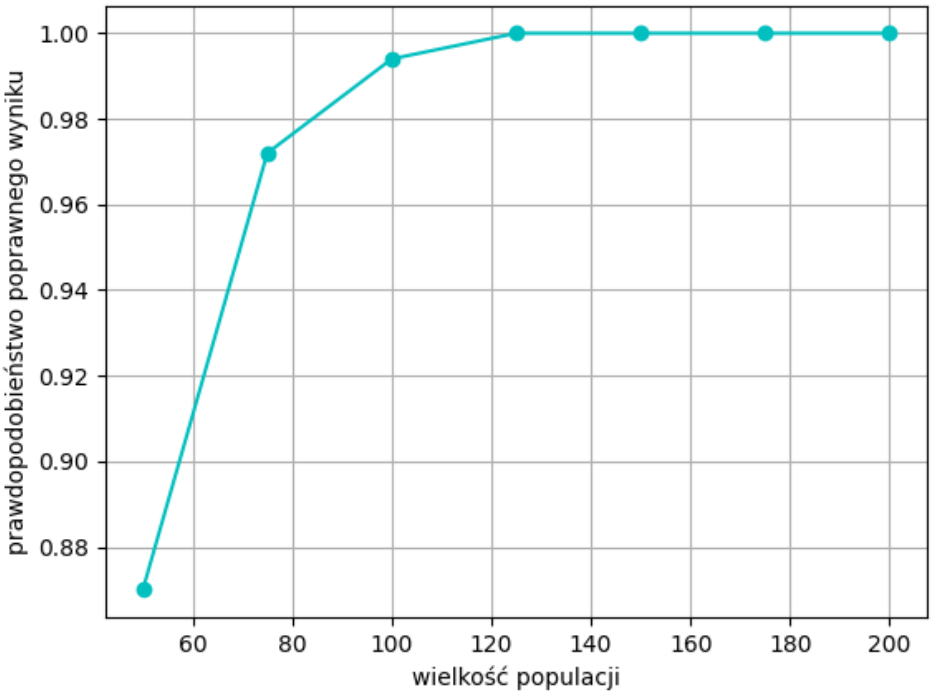
Rysunek 12: Zależność poprawności wyniku od prawdopodobieństwa krzyżowania

Na podstawie otrzymanego wyniku (Rys. 12) stwierdzono, że należy przebadać jeszcze przedział $[0.9, 1.0]$ ze skokiem równym 0.01.



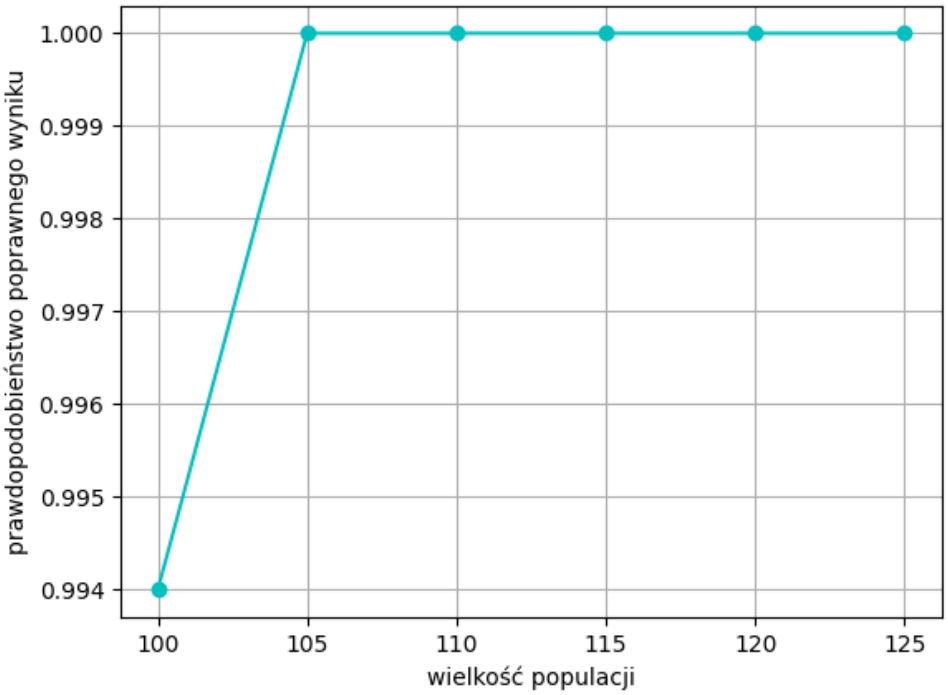
Rysunek 13: Zależność poprawności wyniku od prawdopodobieństwa krzyżowania

Jak wynika z Rys. 13, podobnie jak w przypadku mutacji, każdorazowe wykonywanie krzyżowania pozytywnie wpływa na poprawność wyniku. Dla tak dobranych wartości częstości mutacji oraz krzyżowania, zbadano zależność prawdopodobieństwa otrzymania poprawnego wyniku od wielkości populacji. Początkowo sprawdzono przedział $[50, 200]$ z krokiem równym 25.



Rysunek 14: Zależność poprawności wyniku od wielkości populacji

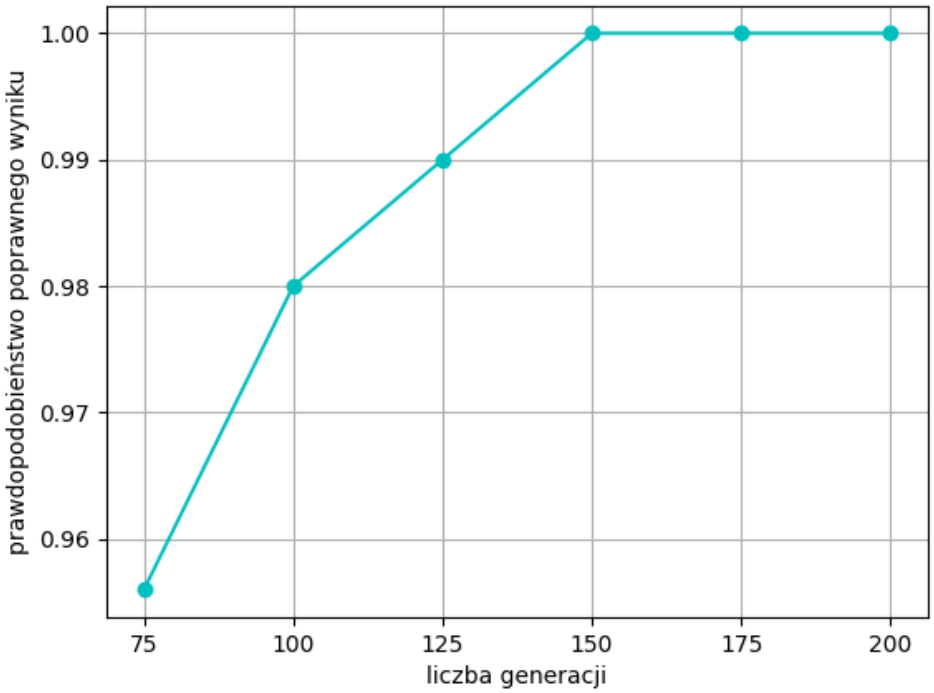
Na podstawie otrzymanego wyniku (Rys. 14) zdecydowano, aby sprawdzić dokładniej przedział $[100, 125]$, dokonano tego z krokiem 5.



Rysunek 15: Zależność poprawności wyniku od wielkości populacji

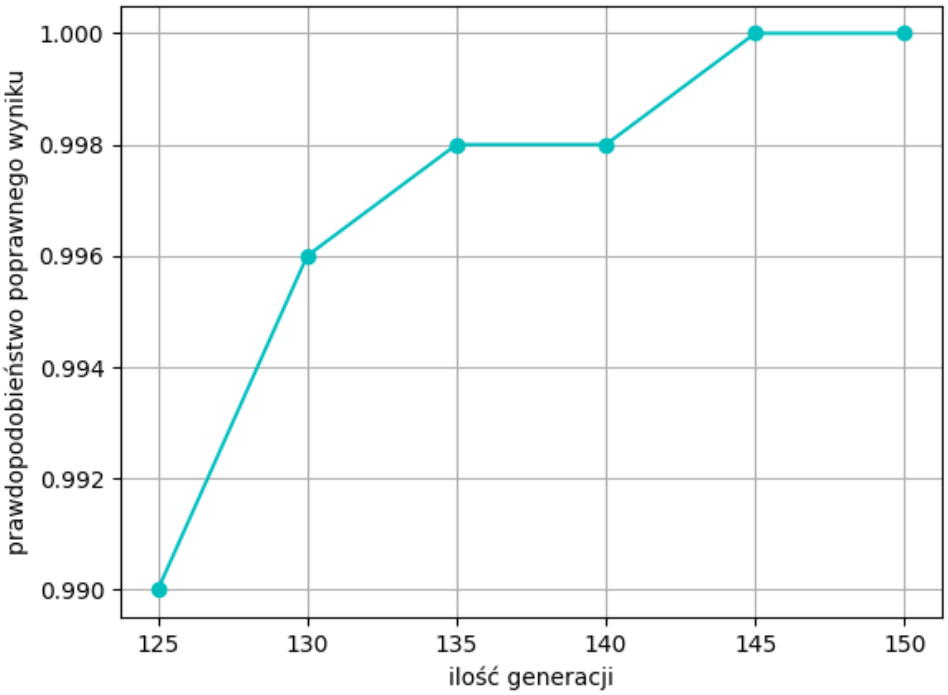
Na podstawie Rys. 15, do dalszych symulacji wybrano najmniejszą wielkość populacji, równą 105 osobników, pozwalającą na otrzymanie poprawnego wyniku.

Kolejnym parametrem, który ma wpływ na poprawność wyniku jest liczba generacji.



Rysunek 16: Zależność poprawności wyniku od liczby generacji

Początkowo przeanalizowano przedział $[75, 200]$ z krokiem równym 25 (Rys. 16), a następnie na jego podstawie dokonano dalszych badań dla przedziału $[125, 150]$ z krokiem 5 (Rys. 17). Pozwoliło to przyjąć liczbę generacji równą 145.



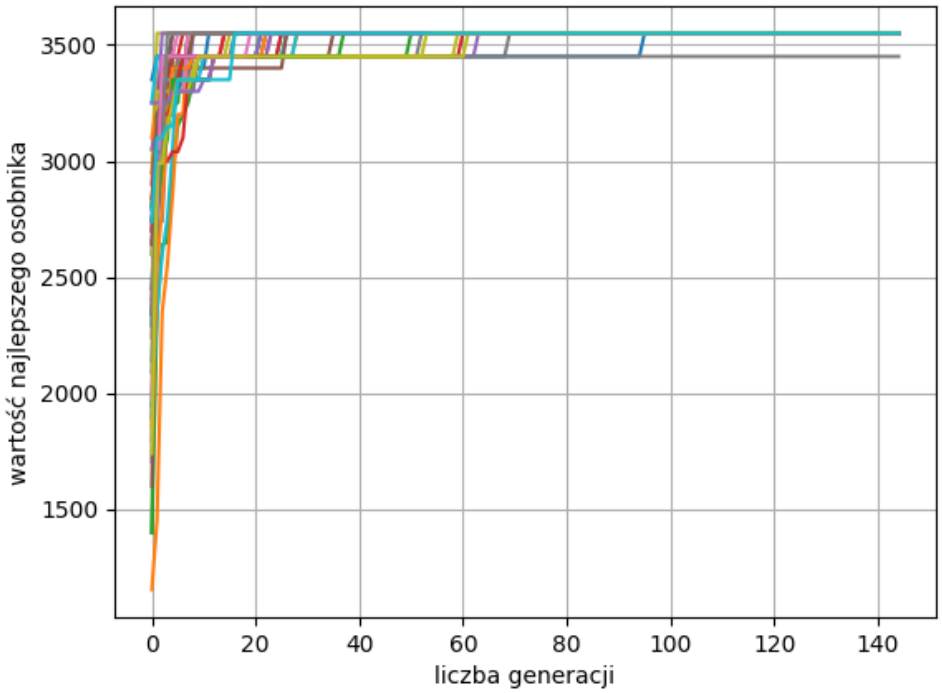
Rysunek 17: Zależność poprawności wyniku od liczby generacji

W ten sposób ustalono ostateczną konfigurację modelu przedstawioną w Tab. 8.

Tabela 8: Konfiguracja ostateczna

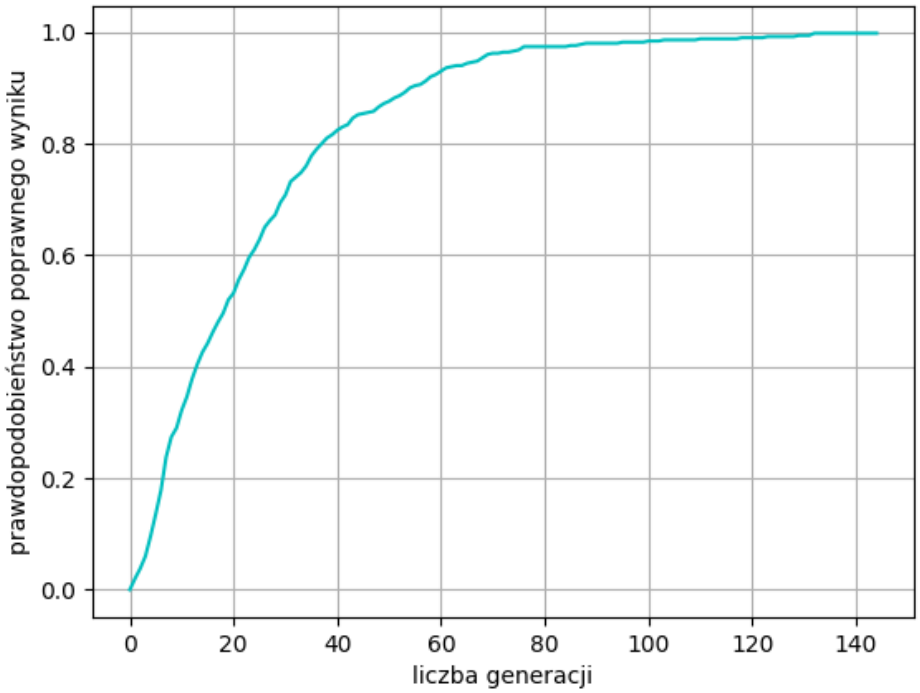
parametr	**wartość**
Liczba generacji	145
Rozmiar populacji	105
Prawdopodobieństwo mutacji	1
Prawdopodobieństwo krzyżowania	1
Warunek stopu	liczba generacji
Selekcja	koło ruletki, elitarna
Rodzaj mutacji	zmiana pojedynczej wartości
Rodzaj krzyżowania	jednopunktowe

W kolejnym kroku postanowiono przyglądnąć się przebiegowi wartości funkcji przystosowania najlepszego osobnika w danej generacji na przestrzeni 145 generacji.



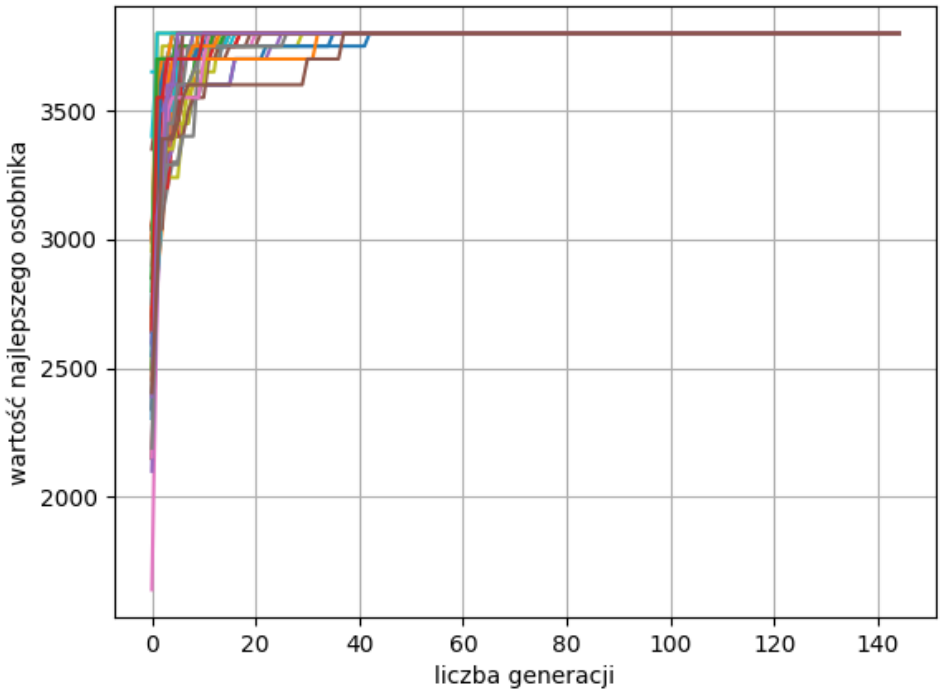
Rysunek 18: Przebieg wartości najlepszego osobnika dla pojemności wagowej plecaka równej 3000

Na Rys. 18, gdzie przedstawiono 50 przypadków widać, że wiele wywołań algorytmu osiąga wartość optymalną bardzo szybko, a jedynie pojedyncze przypadki potrzebują na to większej liczby generacji. Jest to zauważalne na Rys. 19, gdzie optymalna wartość osiągana jest przed 20 generacją dla 50% wywołań algorytmu, 20 generacji później już 80% wywołań algorytmu znajduje wartość optymalną. Po połowie, czyli po 72 ze 145 przewidzianych generacji, 96% uruchomień algorytmu znajduje optymalne rozwiązanie problemu.

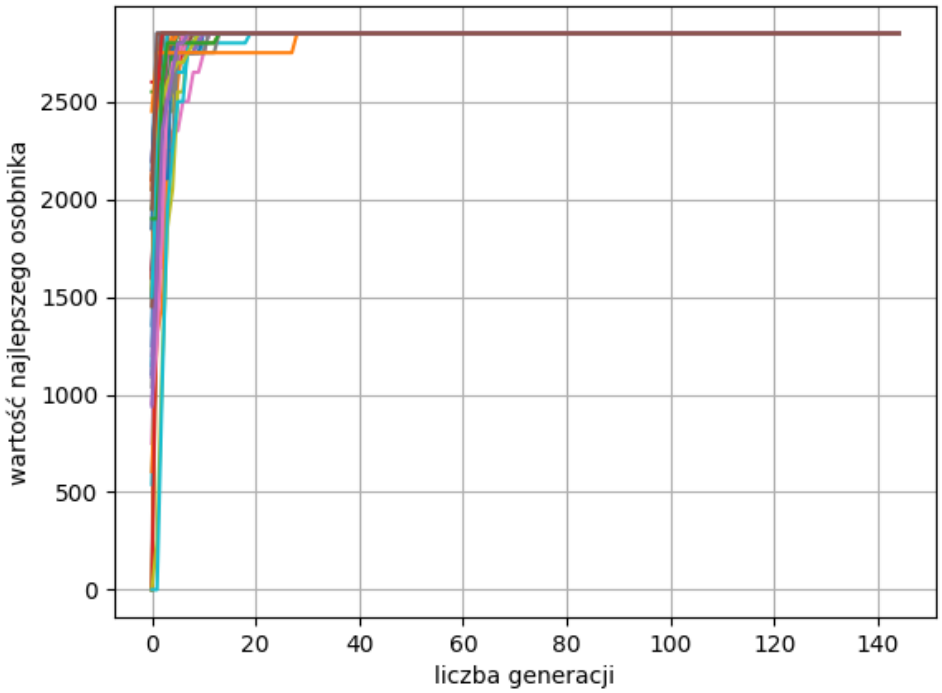


Rysunek 19: Prawdopodobieństwo poprawnego wyniku w zależności od generacji dla pojemności wagowej 3000

W ramach kolejnego badania przeanalizowano ten sam przebieg, tym razem zmieniając pojemność wagową plecaka z 3000 na 3500, a następnie na 2000.



Rysunek 20: Przebieg najlepszego osobnika dla pojemności wagowej plecaka równej 3500



Rysunek 21: Przebieg najlepszego osobnika dla pojemności wagowej plecaka równej 2000

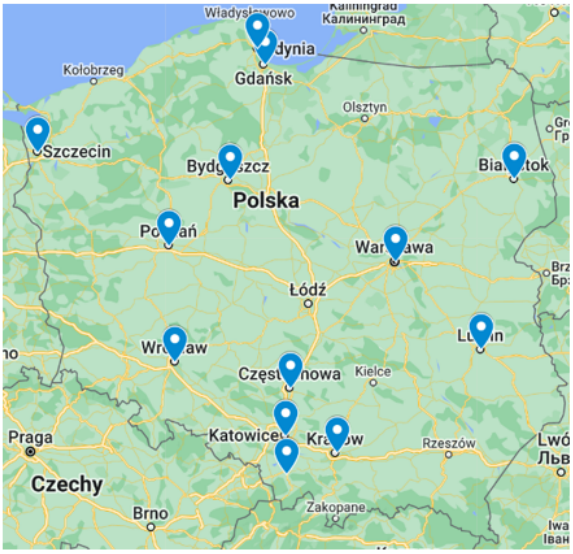
Porównując wykresy na Rys. 18-21 zauważyć można, że pojemność wagowa plecaka ma wpływ na prawdopodobieństwo otrzymania poprawnego wyniku. Jednocześnie, dla analizowanego przypadku, nie zachodzi prosta zależność pomiędzy wzrostem pojemności plecaka a liczbą generacji potrzebnych do uzyskania poprawnego wyniku.

Na koniec warto przeanalizować sens tak skonfigurowanego modelu. Jako że w rozważanym przypadku branych jest pod uwagę 15 przedmiotów, to wszystkich możliwych rozwiązań tego problemu jest 32768 (2^{15}). W czasie całej symulacji, tj. wykonania 145 generacji, powstaje 15225 różnych osobników, czyli 2 razy mniej niż liczba możliwych przypadków. Poszukując optymalnej wartości poprzez sprawdzenie wszystkich przypadków musimy dla każdego z nich wykonać dwie operacje: utworzenie go oraz sprawdzenie jego wartości. Dodatkowo należy sprawdzić czy otrzymana wartość nie przekracza limitu wagowego (sprawdzenie, czy limit nie został przekroczony nie jest liczone jako

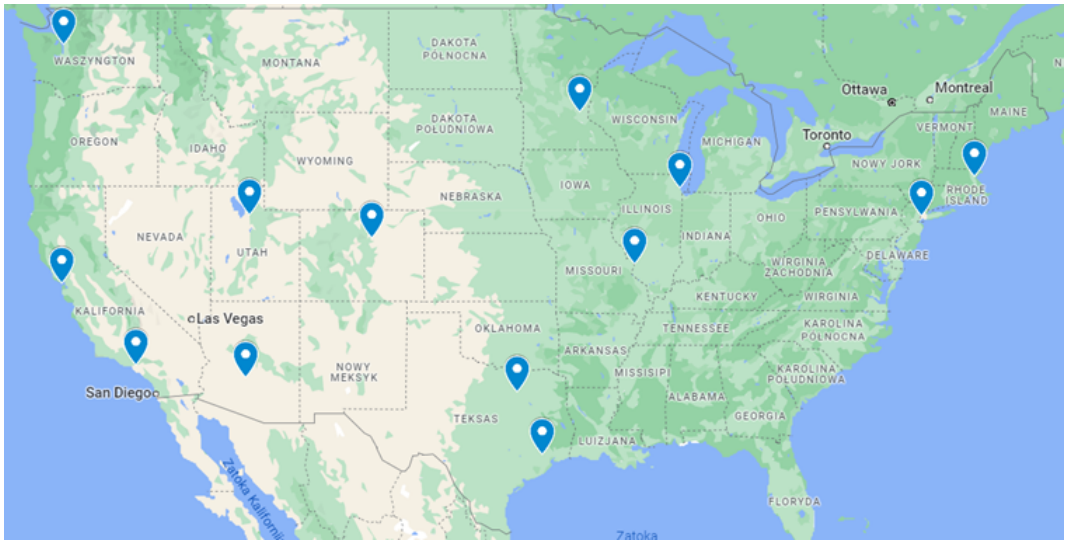
operacja, ponieważ jest to jedynie sprawdzenie czy dana wartość nie jest większa niż dopuszczalna). Podczas wykonywania algorytmu genetycznego, każdy osobnik wymaga wykonania większej liczby operacji: selekcja, krzyżowanie, mutacja oraz przystosowanie. Zatem może się okazać, że analiza wszystkich możliwych przypadków, będzie szybsza niż znalezienie rozwiązania za pomocą tak skonfigurowanego algorytmu genetycznego.

4.2 Problem komiwojażera

Konfiguracja wartości parametrów modelu wykonana została dla dwóch różnych przypadków rozmieszczenia 13 punktów do odwiedzenia, jednego dla Polski (Rys. 22) oraz jednego dla Ameryki (Rys. 23). Pozwoliło to stwierdzić, czy optymalność danego parametru zależna jest od danych wejściowych czy też nie (przynajmniej dla dwóch analizowanych przypadków). Doboru odpowiednich wartości parametrów modelu dokonano na podstawie wyników 500 powtórzeń wykonania algorytmu.



Rysunek 22: Mapa Polski ze zaznaczonymi miejscami do odwiedzenia



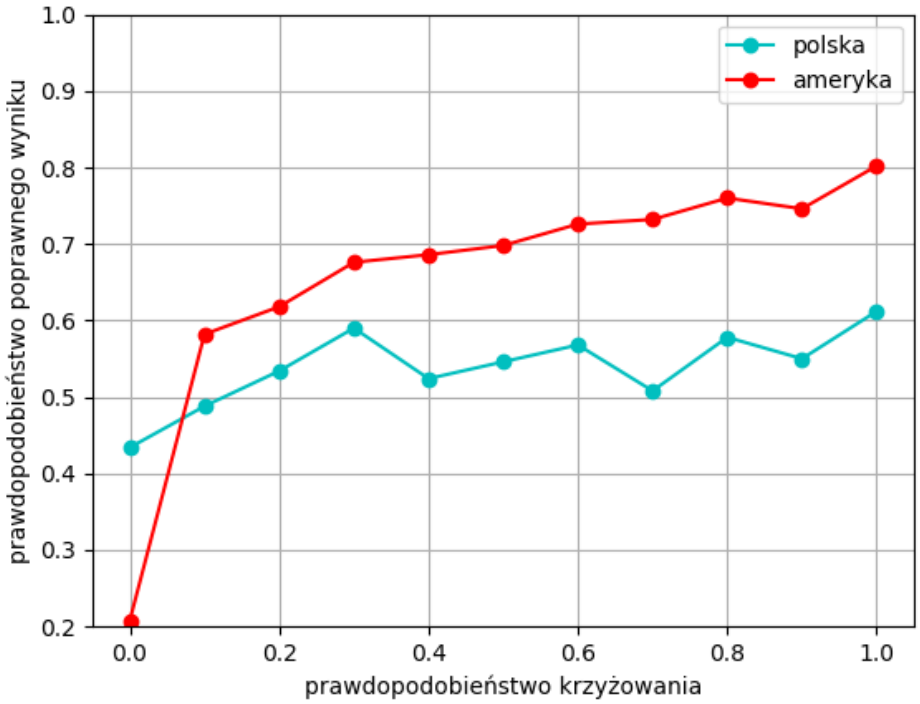
Rysunek 23: Mapa Ameryki ze zaznaczonymi miejscami do odwiedzenia

W pierwszej kolejności przeanalizowano wpływ zmiany prawdopodobieństwa krzyżowania na poprawność wyniku, z początkową konfiguracją przedstawioną w Tab. 9.

Tabela 9: Konfiguracja początkowa

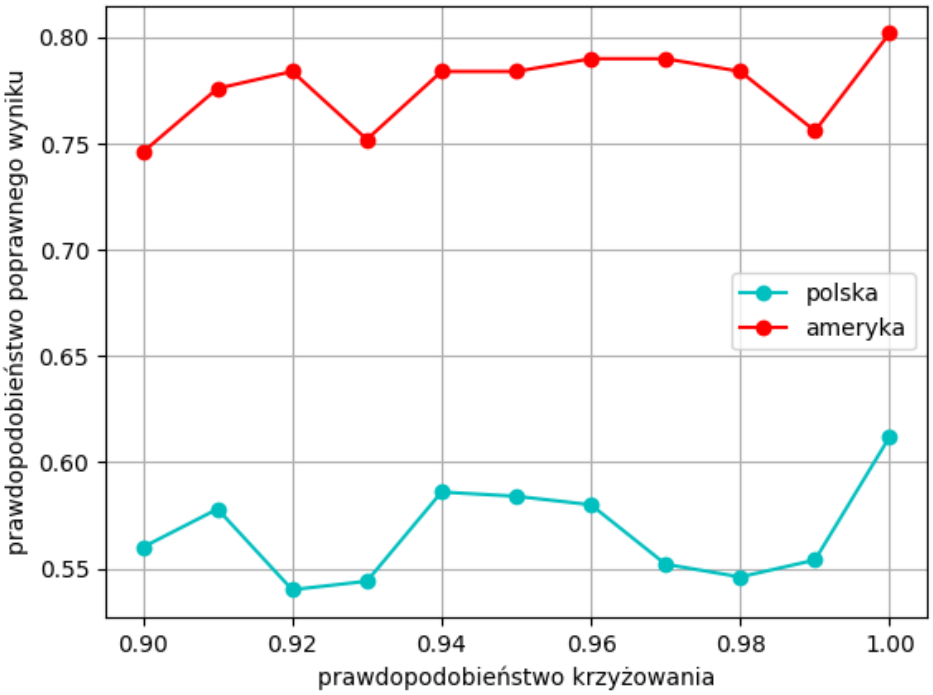
parametr	**wartość**
Liczba generacji	800
Rozmiar populacji	1200
Mutacja	0.03
Krzyżowanie	wartość badana
Warunek stopu	liczba generacji
Selekcja	koła ruletki, strategia elitarna
Rodzaj mutacji	mutacja insercyjna
Rodzaj krzyżowania	Partially Mapped Crossover (PMX)
Statystyka	500 powtórzeń

Rozpoczęto od sprawdzenia prawdopodobieństwa z przedziału $[0, 1]$ ze skokiem 0.1. Wyniki przedstawione zostały na Rys. 24.



Rysunek 24: Wpływ prawdopodobieństwa krzyżowania na poprawność wyniku

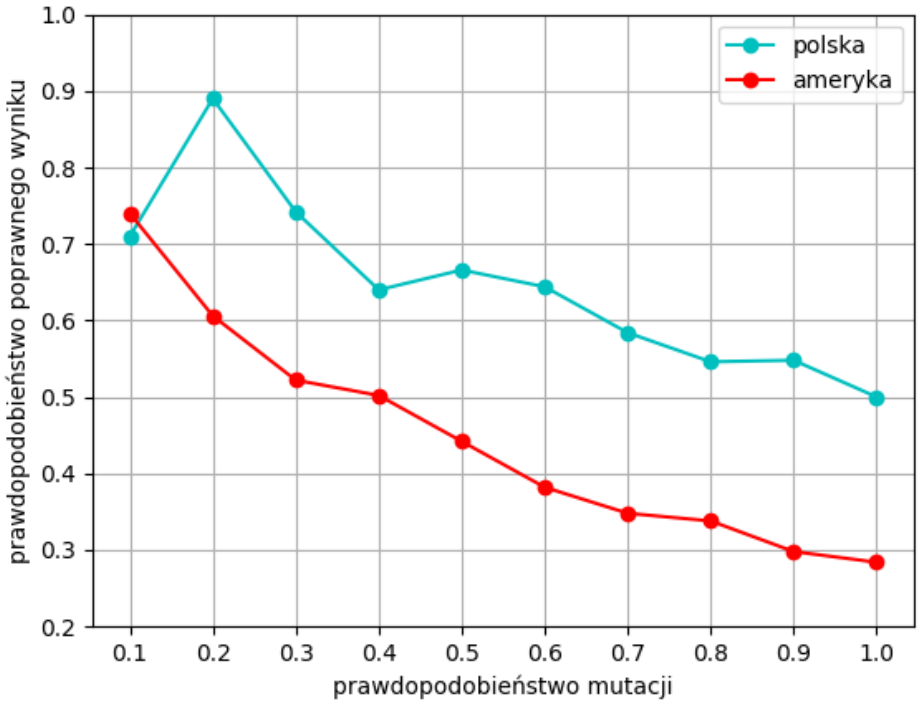
Z wykresu wynika, że warto jeszcze przebadać przedział $[0.90, 1.00]$ ze skokiem 0.01.



Rysunek 25: Wpływ prawdopodobieństwa krzyżowania na poprawność wyniku

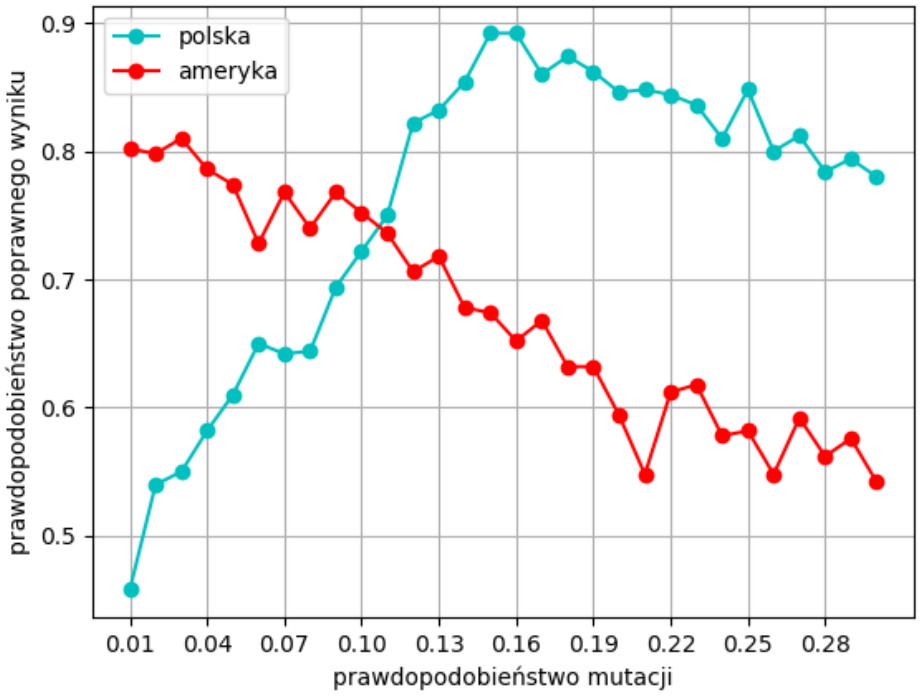
Na podstawie Rys. 25 można wywnioskować, że operacja krzyżowania w obu przypadkach wpływa pozytywnie na prawdopodobieństwo otrzymania poprawnego wyniku, oraz że warto wykonywać ją dla wszystkich nowo powstających osobników.

Podobnie przebiegał proces doboru parametru określającego prawdopodobieństwo mutacji, gdzie również pod uwagę wzięto przedział $[0.1, 1]$ ze skokiem 0.1.



Rysunek 26: Wpływ prawdopodobieństwa mutacji na poprawność wyniku

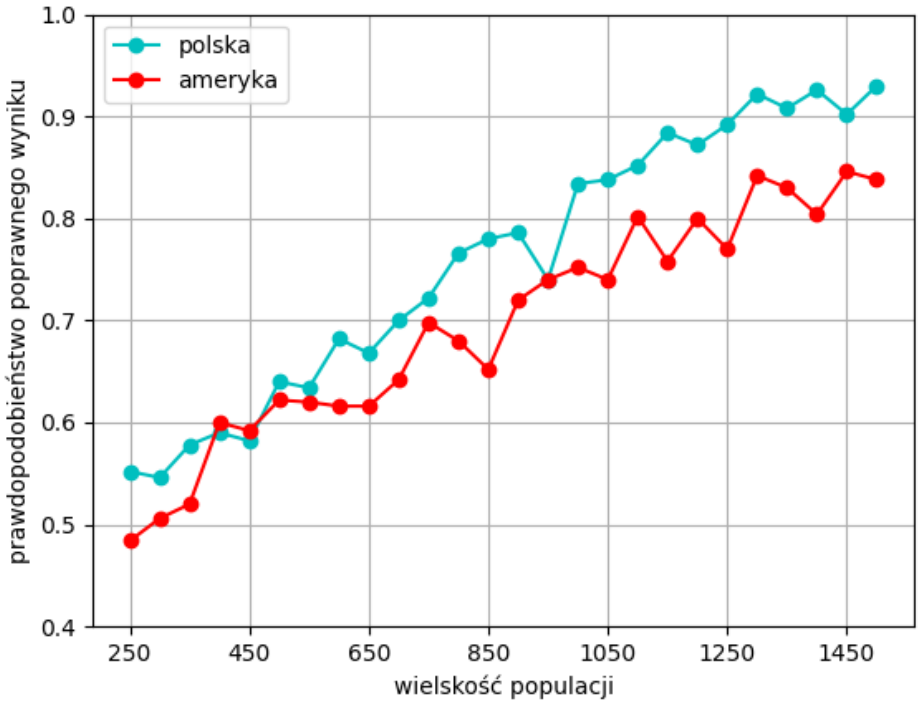
Analizując wykres 26 dostrzec można, że najlepsze efekty powinny zostać otrzymane dla prawdopodobieństwa mutacji z przedziału $[0.01, 0.30]$. Wykonane zostały dalsze badania, dla wartości z tego zakresu ze skokiem prawdopodobieństwa równym 0.01. Warto również zaznaczyć, że całkowity brak mutacji powoduje zmniejszenie różnorodności populacji. W końcowym efekcie otrzymywanych jest 1500 takich samych osobników, z których nie da się utworzyć innego. Jest to zatem lokalne minimum bez możliwości wydostania się z niego.



Rysunek 27: Wpływ prawdopodobieństwa mutacji na poprawność wyniku

Wykres przedstawiony na Rys. 27 pokazuje, że dla mapy Polski najlepsze rezultaty zostały osiągnięte dla wartości prawdopodobieństwa mutacji równego 0.15 oraz 0.16. Jako że prawdopodobieństwo to przekłada się na liczbę wykonywanych operacji, lepszym rozwiązaniem jest wybór wartości mniejszej. Natomiast dla mapy Ameryki najkorzystniej wypadła wartość 0.03. Postanowiono więc, że kolejne badania dla mapy Polski będą wykonane dla prawdopodobieństwo mutacji równego 0.15 oraz dla mapy Ameryki 0.03. Otrzymany wynik wskazuje, że prawdopodobieństwo mutacji jest parametrem, którego wartość należy dobrać do rozwiązywanego przypadku.

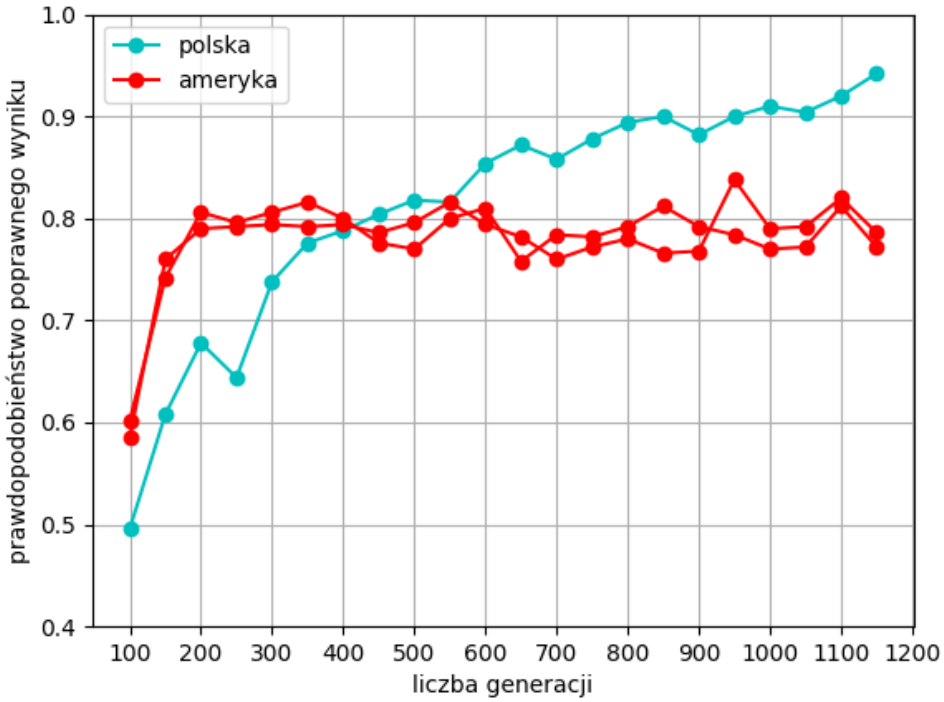
Następnie, dla obu map przebadano jak wielkość populacji wpływa na prawdopodobieństwo poprawnego wyniku. Zbadany został przedział $[250, 1500]$ z krokiem 50.



Rysunek 28: Wpływ rozmiaru populacji na poprawność

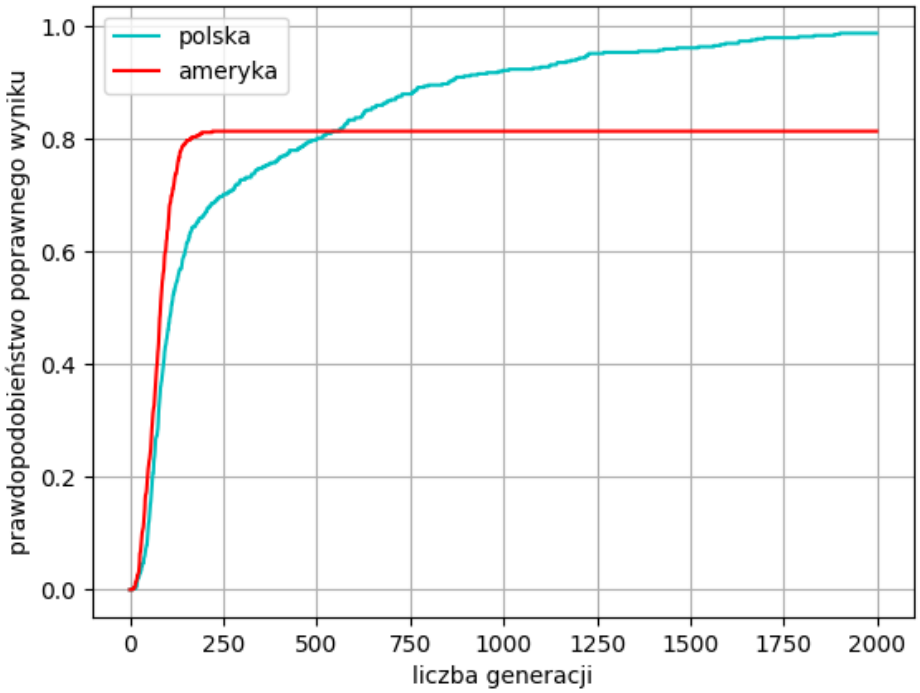
Na podstawie Rys. 28 wybrano wielkość populacji równą 1300, gdyż dla większej liczby osobników nie zaobserwowano poprawy, a wraz ze wzrostem rozmiaru populacji, wzrasta liczba wykonywanych operacji.

Kolejnym parametrem podlegającym badaniom był warunek stopu, czyli maksymalna liczba generacji. Na początku sprawdzono przedział $[100, 1150]$ z krokiem 50. Jak można wywnioskować z Rys. 29, w przypadku Polski wraz ze wzrostem liczby generacji rośnie liczba poprawnych wyników. Natomiast w przypadku Ameryki od pewnego momentu zaobserwowano brak poprawy wyniku. Spowodowało to powtórzenie badania dla tego przypadku, które przyniosło bardzo zbliżone efekty. Postanowiono więc dokonać dalszych badań, z większą liczbą generacji.



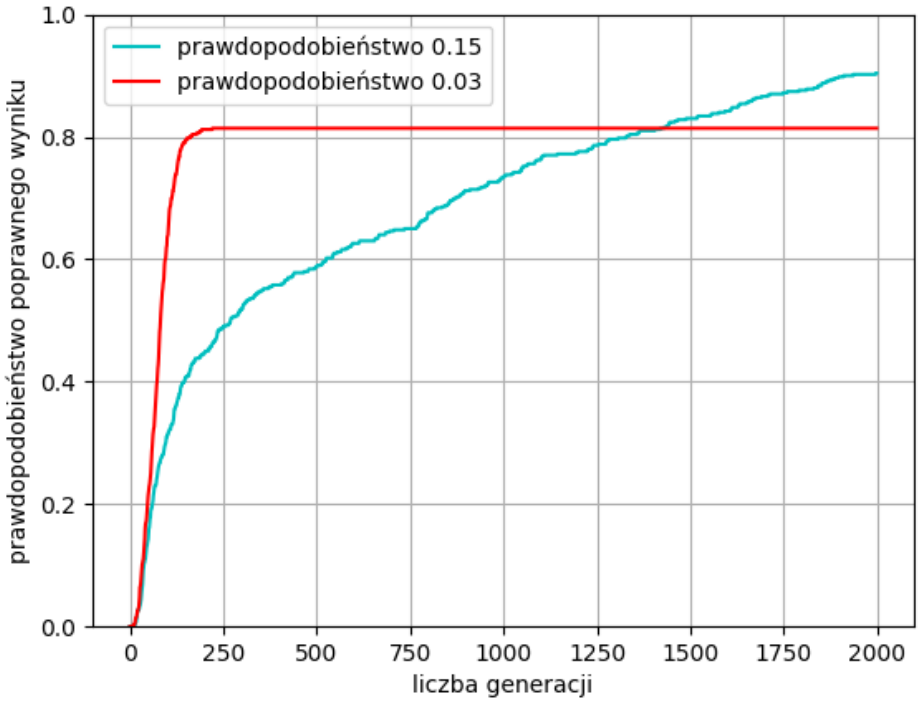
Rysunek 29: Wpływ liczby generacji na poprawność wyniku

Tym razem sprawdzono jak zachowa się algorytm działający przez 2000 generacji (Rys. 30).



Rysunek 30: Wpływ liczby generacji na przebieg najlepszego osobnika

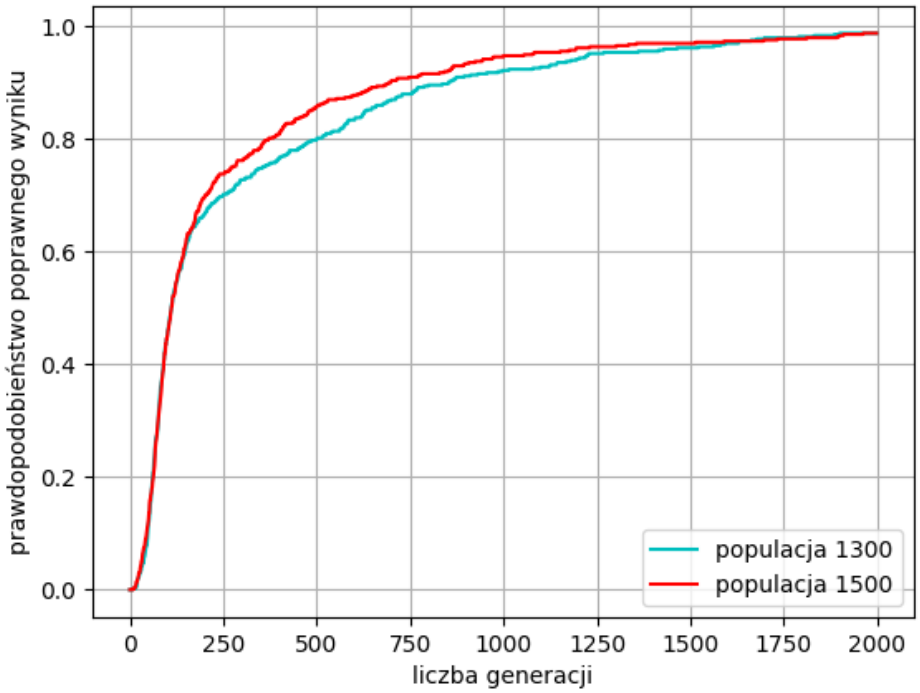
Jak widać, dla mapy Polski prawdopodobieństwo poprawnego wyniku zbliża się do 1. Natomiast dla mapy Ameryki zaobserwować można całkowitą stagnację od około 250 generacji. Postanowiono więc sprawdzić czy zmiana prawdopodobieństwa mutacji z 0.03, na takie jakie zostało ustawione w przypadku Polski, czyli 0.15, pozytywnie wpłynie na otrzymywane wyniki.



Rysunek 31: Wpływ zmiany prawdopodobieństwa mutacji na poprawność wyniku dla Ameryki

Analizując Rys. 31 zauważyć można, że faktycznie zmiana prawdopodobieństwa mutacji zwiększyła prawdopodobieństwo otrzymania poprawnego wyniku, lecz dzieje się tak dopiero od około 1500 generacji. Dalszych badań wpływu liczby generacji na poprawność wyniku nie wykonywano, ponieważ uznano, że wartość 2000 jest odpowiednim kompromisem pomiędzy czasem wykonywania, a poprawnością wyniku.

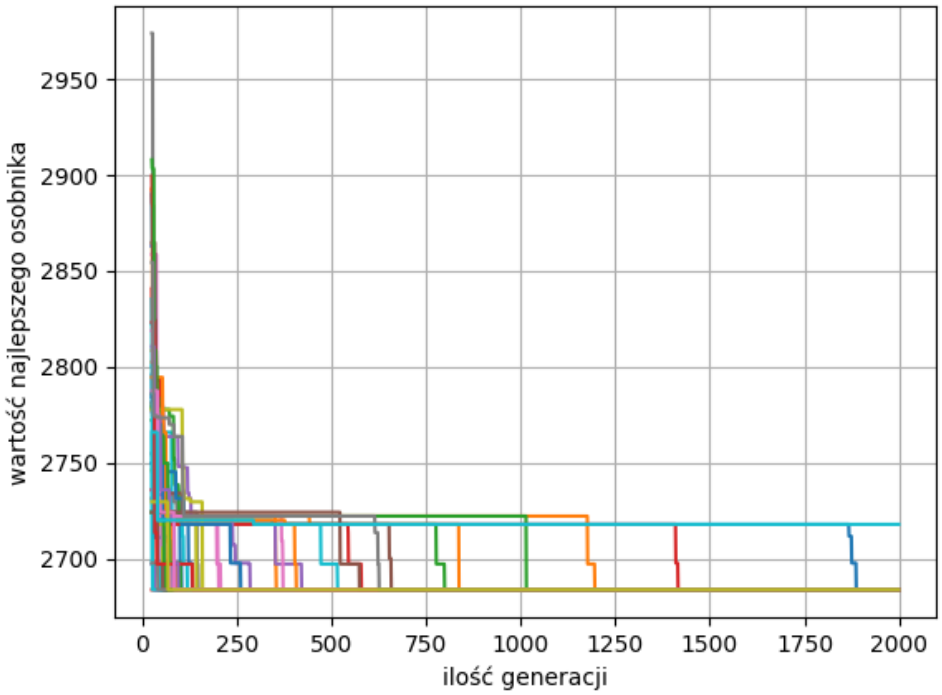
Pamiętając wypłaszczenie wykresu przedstawionego na Rys. 28 dla wartości z przedziału $[1300, 1500]$ postanowiono sprawdzić raz jeszcze, tym razem na nowo dobranych wartościach, czy zwiększenie populacji do 1500 wpłynie pozytywnie na poprawność wyniku. Badanie to przeprowadzono dla mapy Polski.



Rysunek 32: Wpływ zwiększenia wielkości populacji z 1300 do 1500 na poprawność wyniku

Jak wynika z wykresu 32 rozszerzenie populacji z 1300 do 1500 nie wpływa znacząco na zwiększenie prawdopodobieństwa otrzymania poprawnego wyniku. Zatem w dalszych badaniach pozostano przy wcześniej znalezionej wartości, tj. 1300 osobników.

Podjęto również próbę wprowadzenia innego warunku stopu, a dokładnie takiego, który byłby oparty na liczbie generacji, w których nie doszło do zmiany najlepszego osobnika. W tym celu przeanalizowano zmianę wartości funkcji przystosowania dla najlepszego osobnika w każdej generacji, dla mapy Polski. Przebieg dla 50 osobników przedstawiono na Rys. 33, gdzie w celu poprawienia czytelności danych pominięto pierwsze 25 generacji.



Rysunek 33: Zmiana wartości przystosowania najlepszego osobnika na przestrzeni 2000 generacji dla Polski

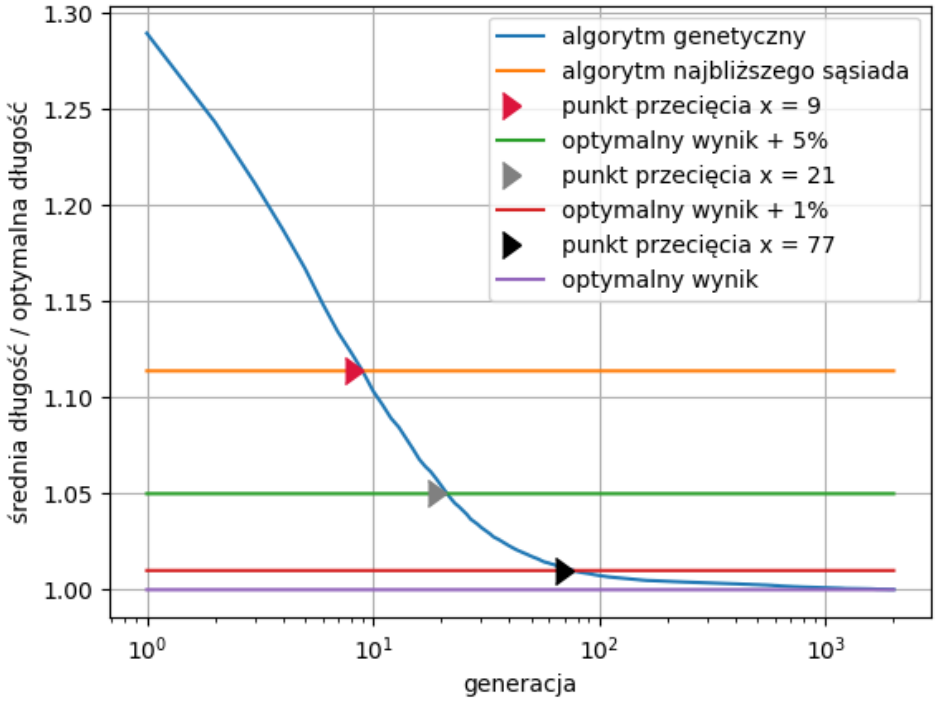
Analizując otrzymane wyniki można stwierdzić, że przyjęcie warunku zakończenia wykonywania algorytmu opartego na liczbie generacji bez zmian nie jest dobrym pomysłem. Pomimo wielu generacji, w których nie dochodzi do zmiany najlepszego chromosomu wciąż nie ma pewności, czy dany wynik nie jest jedynie minimum lokalnym.

Wartości parametrów modelu dobrane na podstawie przedstawionych powyżej analiz zebrano w Tab. 10. Po przyjęciu tych wartości sprawdzono zależność stosunku długości drogi wyznaczonej przez algorytm genetyczny, obliczonej jako średnia po 500 powtórzeniach, do długości drogi optymalnej od numeru generacji. Wartości te porównano z wartościami wyznaczonymi za pomocą algorytmu najbliższego sąsiada, drogą optymalną wydłużoną o 1% oraz o 5%. Na wykresie przedstawionym na Rys. 34 można zauważyć, że algorytm genetyczny zastosowany dla mapy Polski daje lepszy rezultat niż algorytm najbliższego sąsiada już po 9 generacjach. Po 21 generacjach średnia wartość z 500

powtórzeń jest mniejsza od wartości optymalnej zwiększonej o 5%, a po 77 iteracjach wynik jest mniejszy od 101% wartości optymalnej. Po wykonaniu wszystkich 2000 generacji, wartość otrzymana za pomocą algorytmu genetycznego jest większa średnio o $0.156‰$ od wartości optymalnej. Przekłada się to na $0.156km$ na każde $1000km$. Gdyby wartość tą odnieść do obwodu Ziemi, wynoszącego $40075km$, zwiększyłby się on do $40081.236km$.

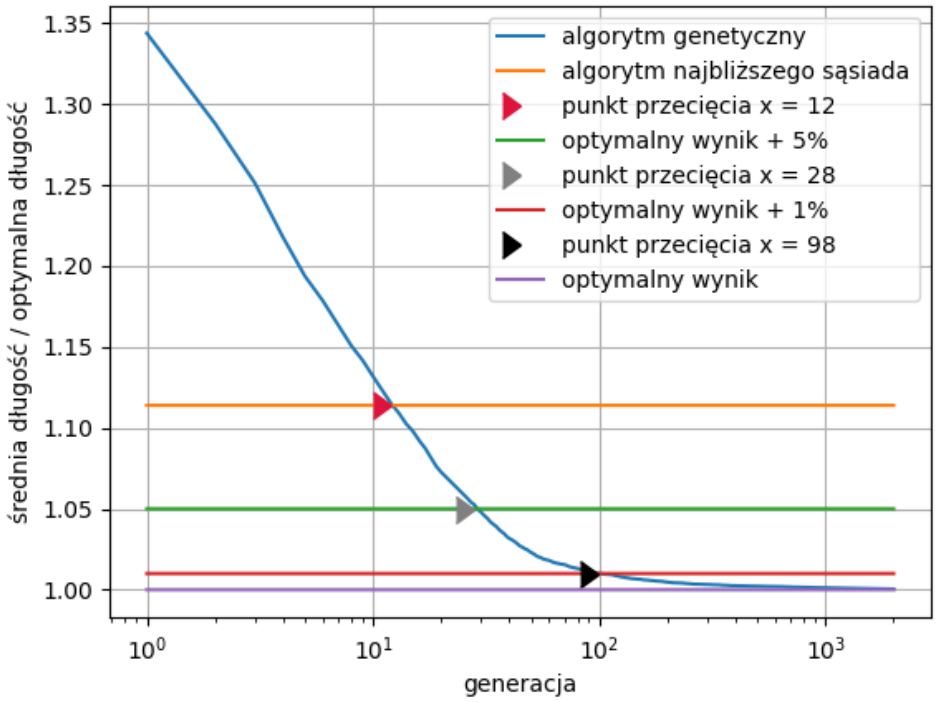
Tabela 10: Konfiguracja ostateczna

parametr	**wartość**
Liczba generacji	2000
Rozmiar populacji	1300
Prawdopodobieństwo mutacji	0.15
Prawdopodobieństwo krzyżowania	1
Warunek stopu	liczba generacji
Selekcja	koła ruletki, strategia elitarna
Rodzaj mutacji	mutacja insercyjna
Rodzaj krzyżowania	Partially Mapped Crossover (PMX)



Rysunek 34: Przebieg stosunku średniej drogi w danej generacji do wartości optymalnej dla mapy Polski

Analogiczne badania zostały wykonane dla mapy Ameryki, a ich wyniki zostały przedstawione na Rys. 35.

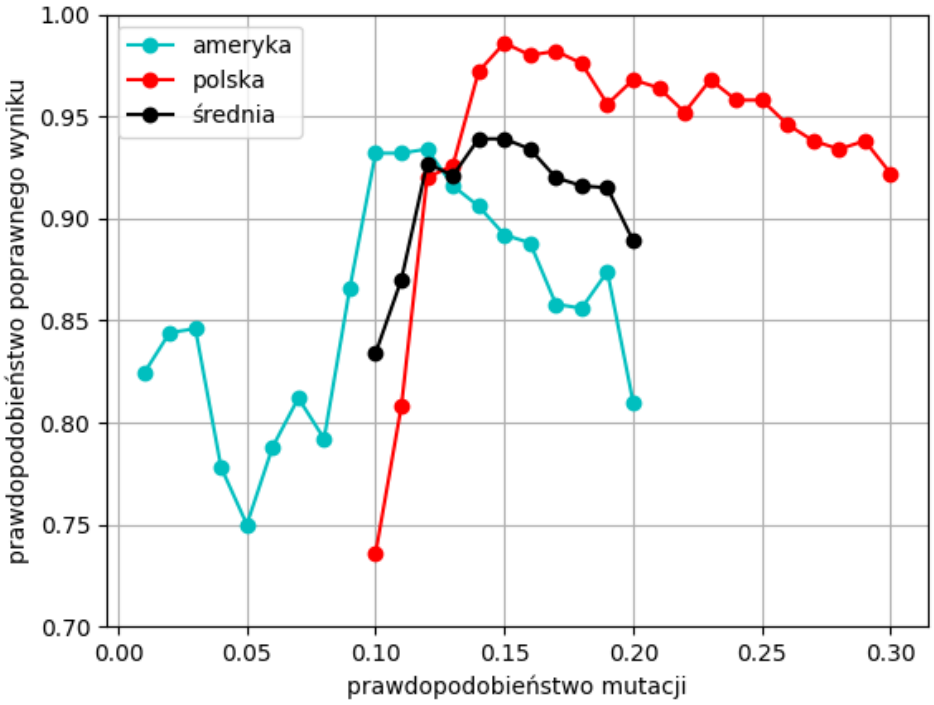


Rysunek 35: Przebieg stosunku średniej drogi w danej generacji do wartości optymalnej dla mapy Ameryki

Zauważyć można, że liczba generacji, potrzebna do uzyskania średniej wartości korzystniejszej niż ta wskazywana przez algorytm najbliższego sąsiada, wynosi 12, czyli o 3 więcej niż w przypadku mapy Polski. Podobnie ma się sprawa z osiągnięciem wartości mniejszej niż 105% optymalnej, gdzie potrzeba średnio 7 generacji więcej. Do przekroczenia 101% wartości optymalnej potrzeba 98 generacji, czyli o 21 więcej. Wynik ostateczny uzyskiwany po wszystkich generacjach jest większy od optymalnego o $0.437‰$. Przekłada się to na $0.437km$ na każde $1000km$ drogi, a w odniesieniu do obwodu Ziemi, zwiększyłby się on do $40092.507km$. Powyższe badania pozwalają stwierdzić, że dane, dla których poszukiwana jest wartość optymalna mają wpływ na skuteczność algorytmu. Warto również zauważyć, że rozważany przypadek zawierał 13 miejsc do odwiedzenia, a macierz odległości była symetryczna, co przekłada się na $239\ 500\ 800$ możliwych rozwiązań. Algorytm o wielkości populacji 1300, w czasie 2000 generacji tworzy jedynie

2600000 osobników, czyli 92 razy mniej niż wynosi liczba możliwych przypadków.

Pamiętając, że prawdopodobieństwo mutacji było tym parametrem, który przyjmował początkowo różne wartości w zależności od przypadku (Polska czy Ameryka) postanowiono ponownie przebadać jaką wartość parametr ten powinien przyjąć. Pozostałe wartości parametrów były zgodne z tymi opisanymi w Tab. 10.



Rysunek 36: Ponowne badanie wpływy prawdopodobieństwa mutacji na poprawność wyniku

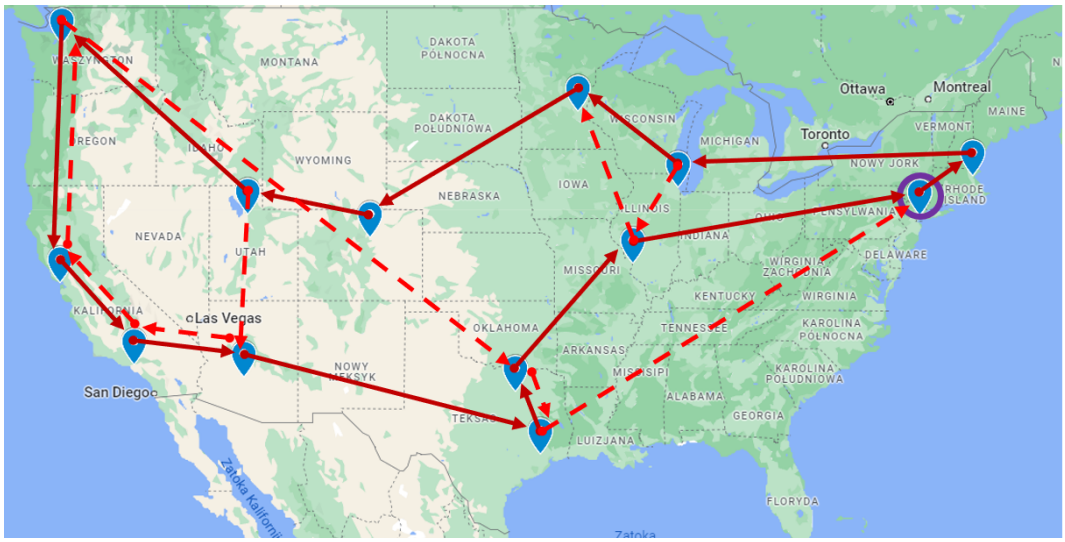
Z wykresu 36 można wyczytać, że ponownie najkorzystniejsza wartość prawdopodobieństwa mutacji jest inna dla przypadku Polski, niż dla Ameryki. Uśredniając wyniki otrzymane dla obu przypadków najkorzystniejsze prawdopodobieństwo wynosi 0.14 oraz 0.15, dla którego wykonywane były wcześniejsze badania. Pokazuje to zatem, że uprzednio dobrana wartość wciąż daje najlepsze rezultaty. Na takiej konfiguracji modelu (Tab. 10) poprzestano dalszych badań.

Na koniec w celu przedstawienia różnic między drogą wskazaną przez algorytm naj-

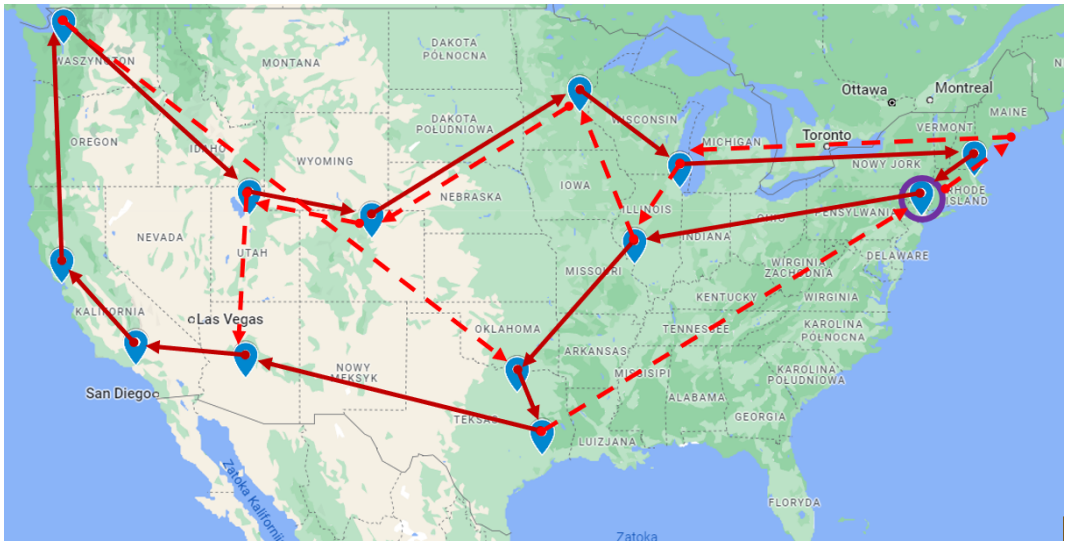
bliższego sąsiada a najlepszą znalezioną przez algorytm genetyczny, utworzono wizualizacje przedstawiające różnicę między tymi drogami. Rezultaty dla Polski przedstawiono na Rys. 37 oraz 38, a dla mapy Ameryki na Rys. 39 oraz 40. Kolorem bordowym przedstawiono drogi znalezione przez algorytm genetyczny, a czerwone, przerywane strzałki odpowiadają drogom wskazanym przez algorytm najbliższego sąsiada. Jeśli oba algorytmy wskazały te samą drogę, to została ona oznaczona jedynie jedną, bordową strzałką.

Rysunek 37: Porównanie dróg znalezionych przez algorytm genetyczny i najbliższego sąsiada dla Polski

Rysunek 38: Porównanie dróg znalezionych przez algorytm genetyczny i najbliższego sąsiada dla Polski



Rysunek 39: Porównanie dróg znalezionych przez algorytm genetyczny i najbliższego sąsiada dla Ameryki



Rysunek 40: Porównanie dróg znalezionych przez algorytm genetyczny i najbliższego sąsiada dla Ameryki

Przedstawione wizualizacje pokazują przewagę algorytmu genetycznego nad zachłannym podejściem algorytmu najbliższego sąsiada.

5 Możliwości rozwoju

W ramach dalszego rozwoju projektu dla algorytmu rozwiązującego problem plecakowy, można by było przebadać jaki wpływ na poprawność wyniku ma liczba przedmiotów, rozrzut wartości oraz wag przedmiotów oraz sama pojemność wagowa plecaka. Kolejnym elementem dalszych badań powinno być sprawdzenie czy obecna konfiguracja modelu, opracowana w oparciu o przypadek przedstawiony w Tab. 6, jest poprawna dla innych zbiorów przedmiotów.

Warte uwagi są również metody selekcji, krzyżowania i mutacji oraz warunek stopu. Są to elementy, które można zrealizować w różny sposób. Na przykład, w ramach selekcji można skorzystać z metody turniejowej [5] lub metody rankingowej [5]. W ramach krzyżowania można skorzystać z wersji wielopunktowej, a nie jednopunktowej jak zostało to zrobione w ramach projektu. Warunek stopu również może przyjmować inną formę, albo można korzystać z wielu warunków jednocześnie. Jednym z możliwych rozszerzeń jest np. skorzystanie z warunku stopu opartego o liczbę generacji, w których nie nastąpiła zmiana najlepszego osobnika.

Algorytm genetyczny rozwiązujący problem komiwojażera również zawiera elementy, które można poddać dalszym badaniom. Jednym z nich jest obserwacja jak algorytm zachowuje się w zależności od wielkości rozwiązywanego problemu. Przy ilu przypadkach algorytm staje się mniej optymalny czasowo, niż sprawdzenie wszystkich możliwości, a kiedy jego poprawność ze względu na duża liczbę miast do odwiedzenia staje się nieakceptowalna.

Kolejnym aspektem wartym analizy jest stwierdzenie z czego wynika to, że w przypadku takiej samej konfiguracji modelu, dla dwóch różnych map otrzymano różne prawdopodobieństwo uzyskania poprawnego wyniku, pomimo tego, że na obu naniesiono taką samą liczbę miejsc do odwiedzenia. Może to np. wynikać z większych dystansów dla Ameryki lub z innego ułożenia miejsc na mapie.

Również w tym przypadku warto by było przebadać metody selekcji, krzyżowania i mutacji oraz warunek stopu, pod kątem tego czy ich zmiana na inne nie spowodowałaby przyśpieszenia działania algorytmu oraz zwiększenia prawdopodobieństwa uzyskania

poprawnego wyniku.

Cały projekt można by rozwinąć w aplikację internetową, w której przy użyciu zewnętrznego API zwracającego macierz odległości dla zadanych miast, można by rozwiązywać problem komiwojażera i wskazywać optymalny sposób odwiedzenia wybranych miejsc.

6 Podsumowanie

Tak jak zauważono we wstępie, istnieją problemy charakteryzujące się bardzo silnym wzrostem liczby możliwych przypadków wraz z niewielkim wzrostem ich rozmiaru. Mimo rozwoju sprzętu komputerowego wciąż przeanalizowanie ich wszystkich pod kątem optymalności stanowi wyzwanie. Z pomocą przychodzą tutaj między innymi algorytmy genetyczne. Na podstawie wykonanych badań zauważono, że mimo braku pewności otrzymania optymalnego wyniku, szansa na jego uzyskanie wciąż jest bardzo wysoka. Jednocześnie liczba przeanalizowanych przypadków jest znacząco mniejsza niż liczba wszystkich możliwości. Świadczy to o tym, że reguły, z których korzysta algorytm genetyczny pozwalają przyśpieszyć rozwiązanie problemu, przy niewielkim ryzyku otrzymania wyniku nieoptymalnego, a jedynie zbliżonego do niego. Ważne jest również, aby model rozwiązujący dany problem korzystał z odpowiednio dobranych metod oraz wartości parametrów takich jak prawdopodobieństwo krzyżowania czy prawdopodobieństwo mutacji, co z jednej strony pozwala znacząco przyśpieszyć proces poszukiwania rozwiązania problemu a z drugiej sprawia, że otrzymane rozwiązanie jest akceptowalne.

Podziękowania

Praca została wykonana z wykorzystaniem Infrastruktury PLGrid.

Bibliografia

[1] Cormen Thomas H. et al. “Wprowadzenie do algorytmów”. In: Wydawnictwo Naukowe PWN, 2012. Chap. 35.2.

[2] Cormen Thomas H. et al. “Wprowadzenie do algorytmów”. In: Wydawnictwo Naukowe PWN, 2012. Chap. 16.2.

[3] *Prezentacja "Heurystyki i metaheurystyki" do przedmiotu Inteligencja obliczeniowa. Katedra Informatyki Stosowanej.* [online]. (dostęp w dniu: 20.12.2022).

[4] Tomasz Dominik Gwiazda. *Algorytmy genetyczne Kompendium Tom 1.* Wydawnictwo Naukowe PWN, 2007. ISBN: 9788301151683.

[5] *Prezentacja "algorytmy genetyczne" do kursu Metody Inteligencji Obliczeniowej dr hab. inż. Szymon Łukasik.* [online]. maj 2022. (dostęp w dniu: 14.12.2022).

[6] *Python.* URL: `https://www.python.org/`. (dostęp w dniu: 14.12.2022).

[7] *Matplotlib.* URL: `https://matplotlib.org/`. (dostęp w dniu: 14.12.2022).

[8] *Pickle.* URL: `https://docs.python.org/3/library/pickle.html`. (dostęp w dniu: 14.12.2022).

[9] *PyCharm.* URL: `https://www.jetbrains.com/pycharm/learn/`. (dostęp w dniu: 19.12.2022).

[10] Clemens Heitzinger. *The Classic Genetic Algorithm.* URL: `https://www.iue.tuwien.ac.at/phd/heitzinger/node30.html`. (dostęp w dniu: 19.12.2022).

[11] Cormen Thomas H. et al. “Wprowadzenie do algorytmów”. In: Wydawnictwo Naukowe PWN, 2012. Chap. 16.

[12] Lawrence Weru. *TSP Algorithms and heuristics.* URL: `https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/`. (dostęp w dniu: 20.12.2022).

[13] Deepak Khemani. *Partially mapped crossover.* URL: `https://www.youtube.com/watch?v=3lc_Fcga5z%5C8%5C&ab_channel=NPTEL-NOCIITM`. (dostęp w dniu: 19.12.2022).

[14] Kusum Deep, Hadush Mebrahtu. *Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman problem.* [online]. 2011. (dostęp w dniu: 14.12.2022).