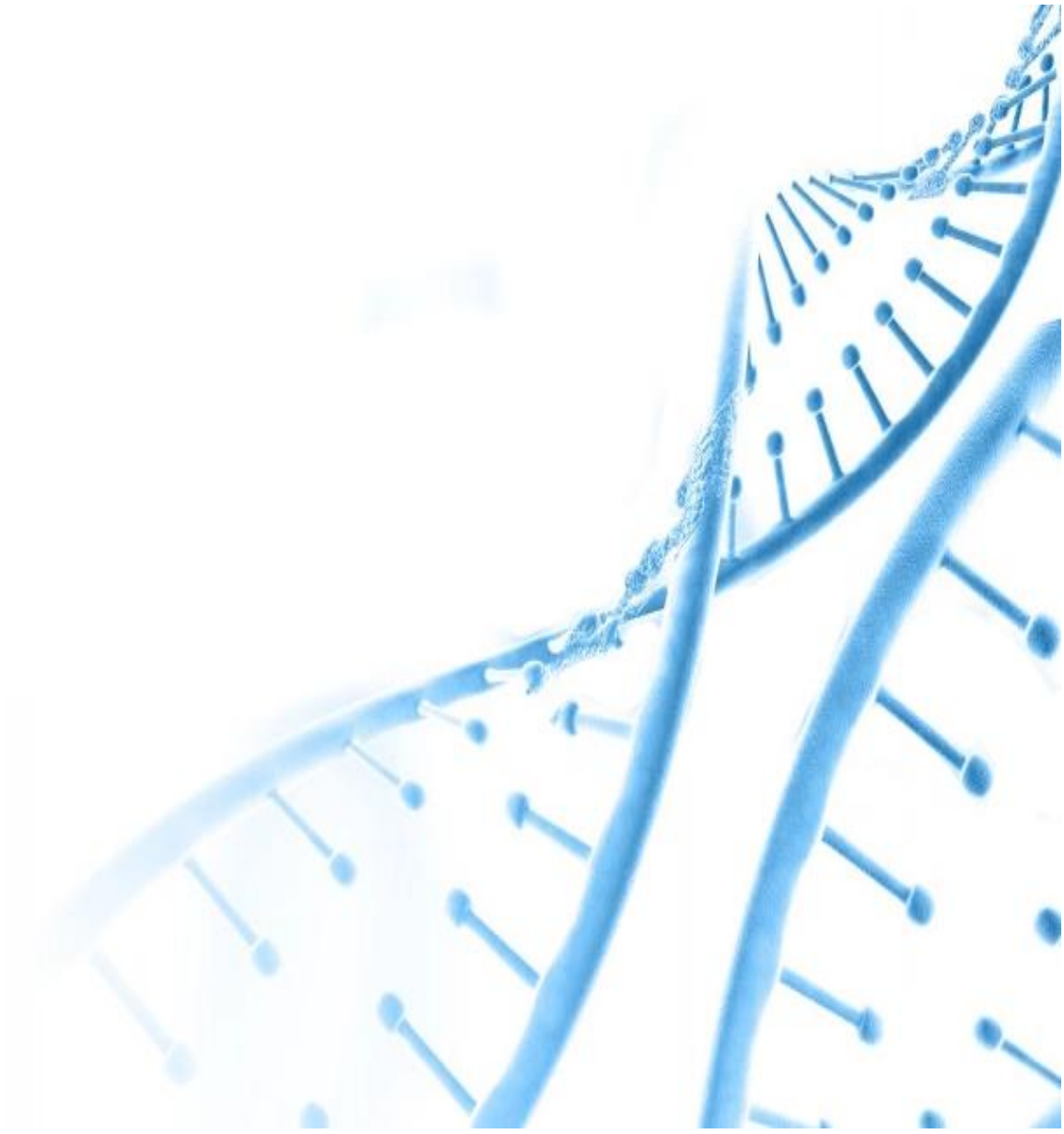
# WYKORZYSTANIE ALGORYTMU GENETYCZNEGO DO ROZWIAZANIA PROBLEMU PLECAKOWEGO I TSP

Mateusz Birkholz | Robert Chyrek



Informatyka stosowana ST | Sztuczna inteligencja
Rok akademicki 2020/2021

# Spis treści

# 1. Wstęp do działania algorytmu

## O algorytmie

Algorytm genetyczny, podobnie jak algorytm PSO jest inspirowany biologicznie. Jego działanie imituje zjawisko selekcji naturalnej i ewolucji. Łączy on w sobie zasadę przeżycia najlepiej przystosowanych jednostek oraz zasadę losowej wymiany informacji.

Jest to algorytm iteracyjny, który pracuje tak długo aż nie zostanie spełniony warunek kończący jego pracę.

Algorytm genetyczny tak jak algorytm PSO jest wykorzystywany do rozwiązywania problemów optymalizacyjnych których rozwiązaniem jest jak najbardziej optymalna wartość zadanej funkcji celu. Wynikiem algorytmu jest chromosom które reprezentuje to rozwiązanie.

Jest to algorytm heurystyczny co oznacza że nie zawsze zwróci właściwy wynik, jest on jednak zobowiązany do zbliżania się do niego z każdą iteracją. Zawiera on elementy losowości więc możliwym jest że czasem znalezienie rozwiązania zajmie więcej czasu, a czasem mniej.

## Zasada działania

W algorytmie genetycznym tworzone są chromosomy (osobniki). Każdy osobnik reprezentuje jedno z wielu możliwych rozwiązań problemu. Osobniki składają się z genów które mogą być reprezentowane przez bity lub np. liczby całkowite. Każdy gen określa jedną z cech rozwiązania.

W pierwszym przejściu generowana zostaje populacja chromosomów. Chromosomy są generowane losowa a ich genom przypisywane są wartości z odpowiedniego zakresu.

Następnie selekcjonowane są najlepsze osobniki z populacji. Selekcja polega na obliczeniu wartości każdego z rozwiązań za pomocą funkcji celu. Określa ona jak optymalne jest dane rozwiązanie. Sposób selekcjonowania może być rankingowy w którym liczą się tylko osobniki z najwyższym wynikiem lub kołowy w którym każdy osobnik ma szanse zostać wybrany, jednak im jest lepszy tym ta szansa jest większa.

Wyselekcjonowane osobniki zostają rodzicami dla nowego pokolenia. Ich geny się ze sobą krzyżują na różne sposoby tworząc kolejne pokolenie. Wielokrotnie krzyżowanie z elementem losowości powtarzane jest tak długo aż nie zostanie wytworzona odpowiednia ilość osobników do kolejnej generacji.

Następnie pokolenie to przechodzi przez proces mutacji. W procesie chromosomy mają szanse na zmianę pewnych genów. Mutacje tak jak i krzyżowanie może wyglądać w różny sposób. Istotny jest jednak fakt że jest to zmiana losowa która nawet nie musi zajść w danym chromosomie. Sekwencja selekcji, krzyżowania i mutacji wykonywana jest z każdą iteracją.

Nazewnictwo:

- Gen – bit lub liczba określająca pewną cechę rozwiązania
- Chromosom – ciąg genów
- Populacja – zbiór chromosomów
- Krzyżowanie – wymiana pewnymi elementami chromosomu między osobnikami
- Mutacja – losowa zmiana wartości genów.

## Przykład cyklu krzyżowania i mutacji

Chromosomy z genami o wartościach [0-1]:

Chromosomy wypełniane są wartościami z zakresu. Ilość genów jest zależna od ilości cech jakie musi zawierać rozwiązanie.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

Krzyżowanie się chromosomów poprzez podzielenie ich na pół i wymianę.

W tym wypadku dla przykładu chromosomy są przecięte w połowie jednak miejsce przecięcia również może być losowe.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

Mutacja chromosomów.

Wartości losowo wybranych genów zostają zmienione na inne. W jednym chromosomie może występować więcej niż jedna mutacja ale również może ona nie występować wcale.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Szansa na wystąpienie mutacji może być zmieniana wedle uznania, tak samo jak ich ilość.

## Schemat działania algorytmu

- Wygeneruj populacje początkową
- Dla każdej iteracji:
    - Posortuj chromosomy według wartości funkcji celu dla ich genów.
    - Dopóki populacja nie zostanie zapełniona:
        - Wybierz osobniki
        - Krzyżuj wybrane osobniki
        - Krzyżówki poddawaj mutacji
        - Dodaj krzyżówki do populacji kolejnej iteracji
- Zwróć rozwiązanie z najwyższym wynikiem w całej populacji

# 2. Problem plecakowy

## Opis problemu

Założenie problemu jest takie że mamy plecak o danej pojemności(tudzież udźwigu). Mamy również różne przedmioty. Każdy z nich posiada swoją własną wielkość(tudzież wagę) oraz wartość. Rozwiązaniem problemu jest znalezienie takiej kombinacji przedmiotów, aby ich wartość była jak najwyższa ale żeby jednocześnie nie przekraczała ona maksymalnego udźwigu plecaka.

Żeby łatwiej to sobie wyobrazić można postawić się na miejscu handlarza który idzie na targ z plecakiem. Sprzedawca wie, że wszystko co zaniesie sprzeda za określoną cenę. Musi on tak zapakować rzeczy do plecaka żeby mógł go unieść, a przy tym jak najwięcej zarobić.

## Sposób wykorzystania algorytmu genetycznego w problemie plecakowym

Aby wykorzystać algorytm genetyczny przygotowaliśmy klasę Thing której obiekty posiadają swoja nazwę, wagę i wartość. Użytkownik ma możliwość wprowadzenia wszystkich przedmiotów i ich właściwości do programu.

### Projekt chromosomu

Zaprojektowaliśmy chromosom w taki sposób aby był on długości listy obiektów wprowadzonych przez użytkownika. Oznacza to ze każdy gen w chromosomie odpowiada jednemu przedmiotowi z listy. Geny przyjmują wartości 1 lub 0. Jeśli gen jest równy 1 oznacza to, że przypadający mu przedmiot jest pakowany do plecaka w danym rozwiązaniu. Analogicznie jeśli jest równy 0 to przedmiot ten nie jest uwzględniany w tym rozwiązaniu.

### Funkcja celu

Funkcja celu jest sumą wartości przedmiotów które wchodzą w skład rozwiązania. Jednak suma wag tych przedmiotów nie może przekraczać maksymalnego udźwigu plecaka. W związku z tym napisaliśmy funkcję która dla zadanego chromosomu będzie sumować wartości i wagi przedmiotów dla których jego geny mają wartość 1. Pod koniec sumowania, jeśli waga przedmiotów jest większa niż maksymalny udźwig plecaka, wartość tego rozwiązania jest ustawiana na 0, ponieważ rozwiązanie przekraczające udźwig jest nic niewarte.

*Przykładowa lista rozwiązań oraz przykładowy chromosom w naszym programie:*

| Przedmiot | Waga | Wartość |
|---|---|---|
| Laptop | 2200 | 1200 |
| Kubek | 600 | 150 |
| Notatnik | 200 | 340 |
| Długopis | 30 | 100 |
| Telefon | 140 | 700 |
| Chlebak | 400 | 320 |

Zgodnie z tymi danymi w tym chromosomie, rozwiązanie zawierać będzie w sobie laptopa, notatnik, długopis i telefon.

Wartość tego rozwiązania będzie zatem równa:

$$V = 1200 + 340 + 100 + 700 = 2340$$

Natomiast waga tego rozwiązania będzie równa:

$$W = 2200 + 200 + 30 + 140 = 2570$$

| | | | | | |
|---|---|---|---|---|---|
| **1** | **0** | **1** | **1** | **1** | **0** |

## Selekcja

Selekcja odbywa się za pomocą sortowania chromosomów w populacji. Funkcja sortujące bierze pod uwagę wynik funkcji celu podczas sortowania. Oznacza to że najwyżej w hierarchii będą te rozwiązania dla których funkcja celu zwróci najwyższą wartość.

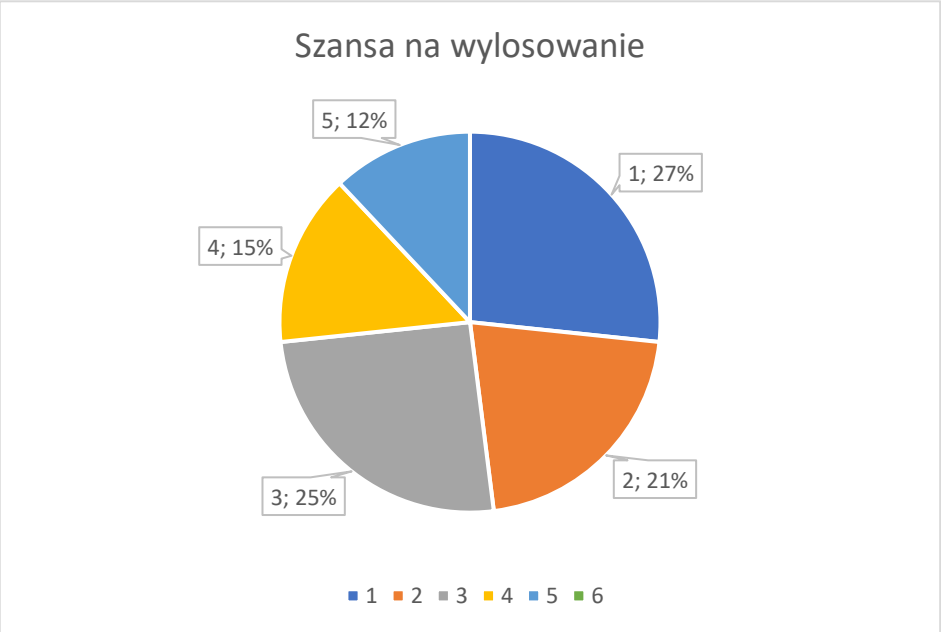
Nasz program uwzględnia dodanie dwóch najlepszych osobników danej generacji do nowej generacji.

Następnie selekcja odbywa się w sposób kołowy. Oznacza to że każde rozwiązanie ma szansę zostać wybrane, jednak szansa ta jest zależna od wyniku jaki osiąga dane rozwiązanie. Jest to pewien element losowości i mechanizm mający na celu sprawić ze algorytm będzie imitował „szczęście".

*Działania selekcji kołowej w praktyce:*

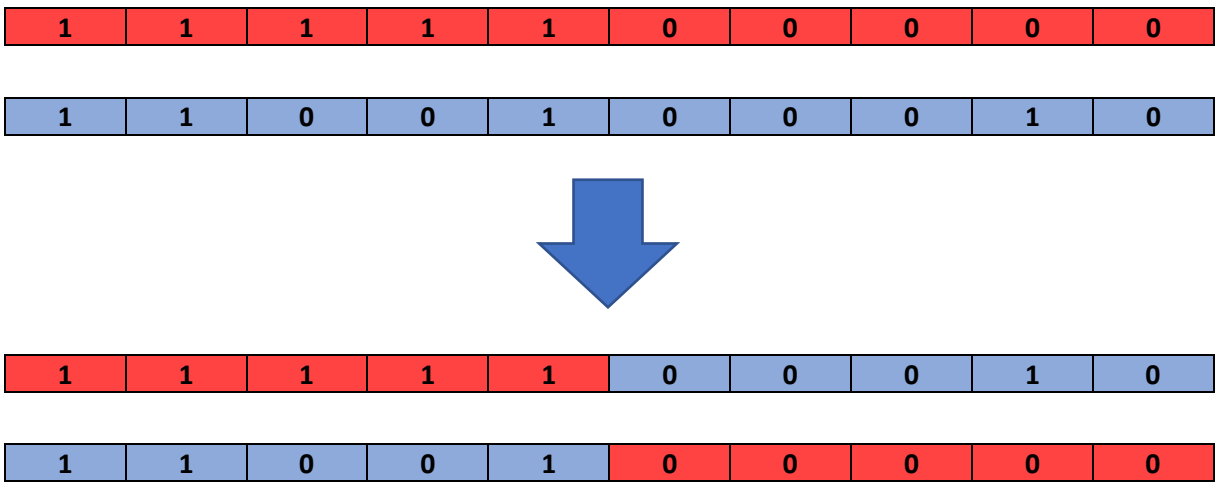
| Numer rozwiązania | Wartość rozwiązania |
|---|---|
| 1 | 4000 |
| 2 | 3200 |
| 3 | 3800 |
| 4 | 2200 |
| 5 | 1800 |
| 6 | 0 |

Wykres szansy na wylosowanie jako rodzic




## Krzyżowanie

Krzyżowanie odbywa się na zasadzie przecięcia chromosomów w losowym miejscu a następnie na wymianie powstałymi połówkami. W ten sposób z dwojga rodziców powstaje dwoje potomków.



## Mutacja

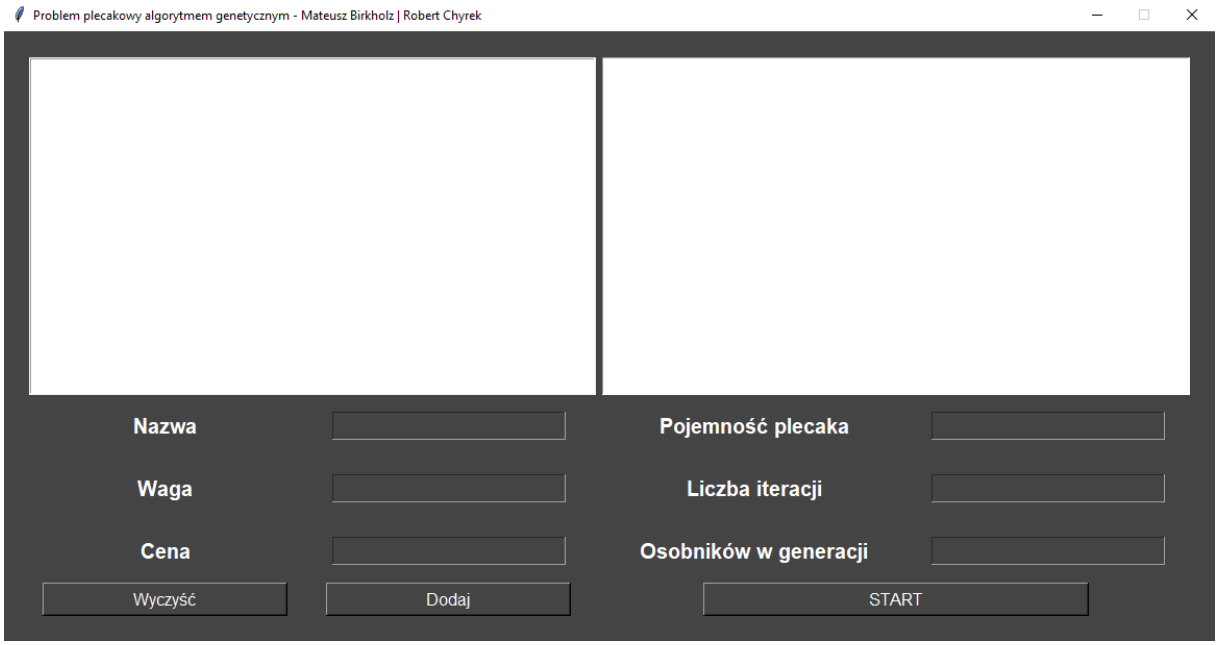
Mutacja odbywa się na zasadzie negacji losowych bitów chromosomu. Ilość mutacji ustawiona jest na 1 a prawdopodobieństwo jej wystąpienia na 50%. Funkcja mutująca uruchamia pętlę tyle razy ile mutacji ma zajść. Wewnątrz tej pętli losowany jest gen. Następuje losowanie czy dany gen zostanie poddany mutacji czy nie i w zależności od wyniku zostaje poddany mutacji lub nie.

## Wynik

Wynikiem działania algorytmu jest najnowsza generacja chromosomów posortowana według wartości funkcji celu. W związku z tym najlepszym rozwiązaniem będzie pierwszy chromosom w tej generacji.

## Opis działania programu

Okno programu wygląda następująco:




Po lewej stronie mamy możliwość dodawania własnych przedmiotów wraz z ich właściwościami. Po prawej zaś możemy ustawić pojemność plecaka, liczbę iteracji oraz ilość osobników w generacji.

UWAGA! W programie na sztywno przypisaliśmy przykładowe przedmioty. Aby się ich pozbyć z puli wystarczy na początku działania programu kliknąć przycisk „Wyczyść”.

## Testowanie działania algorytmu

W Internecie udało nam się znaleźć przykładowe dane dla problemu plecakowego (link do strony w źródłach).

Dane wyglądają następująco:

| Numer przedmiotu | Wartość | Waga |
|---|---|---|
| 1 | 92 | 23 |
| 2 | 57 | 31 |
| 3 | 49 | 29 |
| 4 | 68 | 44 |
| 5 | 60 | 53 |
| 6 | 43 | 38 |
| 7 | 67 | 63 |
| 8 | 84 | 85 |
| 9 | 87 | 89 |
| 10 | 72 | 82 |

Rozmiar plecaka w tym przypadku wynosi 165 natomiast oczekiwany rezultat to kombinacja przedmiotów o numerach 1,2,3,4 i 6.

Program zaczynamy testować najpierw na 10 iteracjach i 10 osobnikach w generacji. Otrzymywane przez nas wyniki rzadko się powtarzały a ich wartość oscylowała w granicach między 228 a 284.

Zwiększyliśmy zatem ilość generacji do 100.

Dla tej ilości iteracji poza, rzadkimi przypadkami, wynikiem były na przemian występujące kombinacje:

| Kombinacja przedmiotów | Wartość rozwiązania |
|---|---|
| 1,2,4,7 | 284 |
| 1,2,3,4,6 | 309 |
| 1,2,3,10 | 270 |

Można już zauważyć występowanie oczekiwanego rozwiązania jednak nie zawsze jest ono zwracane. Zwiększyliśmy zatem liczbę iteracji o kolejne 100.

Dla 200 iteracji program w 80% przypadków zwracał najlepszy możliwy wynik. Poza tym wynikiem pojawiała się jedynie kombinacja 1,2,4,7 o wartości 284 która jest, najbliższym do optymalnego, dotychczas zanotowanym wynikiem.

W tym momencie zauważyliśmy że dalsze zwiększanie ilości iteracji w dość mały stopniu wpływa na poprawę wyników. Postanowiliśmy zatem zwiększyć liczbę osobników w generacji do 20. Wynik był bardzo satysfakcjonujący ponieważ dla takich danych program w prawie 100% przypadków zwracał oczekiwaną kombinację 1,2,3,4,6. Wywnioskowaliśmy z tego ze program działa prawidłowo. Wykazuje coraz lepsze wyniki im więcej prób i osobników się mu da.

# 3. Problem TSP

## Opis problemu

Problem TSP (Traveling Salesman Problem), inaczej nazywany problemem komiwojażera, polega na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Co to oznacza? Oznacza to że mając graf(którego krawędzie posiadają wagi) mamy znaleźć taką kombinacje tych krawędzie która pozwoli nam przejść przez wszystkie wierzchołki grafu a następnie wrócić do początkowego wierzchołka. Najbardziej optymalnym rozwiązaniem jest rozwiązanie które uwzględnia najkrótsza możliwą trasę, to znaczy przechodzi po krawędziach których suma wag będzie jak najniższa.

Wyobrazić to sobie można wczuwając się w rolę wędrującego handlarze od którego pochodzi nazwa tego problemu. Handlarz wędruje między miastami. Na swoich podróżach chciałby jak najwięcej zarobić. W związku z tym szuka takiej trasy która pozwoli mu objechać wszystkie miasta i wrócić do domu, w taki sposób aby jak najtaniej/najkrócej wyszła go jego podróż.

## Sposób wykorzystania algorytmu genetycznego w problemie plecakowym

Aby wykorzystać algorytm genetyczny przygotowaliśmy klasę Miasto której obiekty posiadają swoja nazwę, współrzędną X i współrzędną Y. Użytkownik ma możliwość wprowadzenia wszystkich miast i ich pozycji do programu.

### Projekt chromosomu

Zaprojektowaliśmy chromosom w taki sposób aby był on długości listy miast wprowadzonych przez użytkownika. Geny przyjmują wartości od 0 do ilości wprowadzonych miast zmniejszonej o 1. Przykładowo jeśli jest 10 miast to geny przyjmują wartości od 0 do 9. Pozycja genu oznacza którym w kolejności odwiedzanym miastem jest jego wartość. Na przykład jeśli 2 gen jest wartości 2 to znaczy że drugie miasto jest trzecim (ze względu na wspomniane przesunięcie -1) odwiedzanym w tym rozwiązaniu miastem. Wartości w genach nie mogą się powtarzać ponieważ oznaczałoby to odwiedzenie któregoś z miast więcej niż 1 raz.

### Funkcja celu

Funkcja celu jest sumą odległości między miastami liczonej zgodnie z kolejnością przechodzenia między nimi zawartą w chromosomie. Funkcja ta nie ma żadnego ograniczenia w odróżnieniu od problemu plecakowego w którym problemem była waga wszystkich przedmiotów. Jak można się domyślić w tym przypadku najlepszym rozwiązaniem będzie to które uzyska **najniższy** wynik.

*Przykładowa lista rozwiązań oraz przykładowy chromosom w naszym programie:*

| Nr Miasta | X | Y |
|---|---|---|
| 0 | 835 | 98 |
| 1 | 930 | 356 |
| 2 | 825 | 805 |
| 3 | 716 | 143 |
| 4 | 396 | 333 |
| 5 | 110 | 348 |

Zgodnie z tymi danymi w tym chromosomie, rozwiązanie przewiduję trasę 3-2-5-0-4-1-3

Wartość tego rozwiązania będzie zatem równa:

$$V = \sum_{i=0}^{n-1} \sqrt{(Miasto_i.x - Miasto_{i+1}.x)^2 + (Miasto_i.y - Miasto_{i+1}.y)^2} + \sqrt{(Miasto_0.x - Miasto_{n-1}.x)^2 + (Miasto_0.y - Miasto_{n-1}.y)^2}$$

Gdzie: n - liczba miast

| | | | | | |
|---|---|---|---|---|---|
| **3** | **2** | **5** | **0** | **4** | **1** |

## Selekcja

Selekcja odbywa się za pomocą sortowania chromosomów w populacji. Funkcja sortujące bierze pod uwagę wynik funkcji celu podczas sortowania. Oznacza to że najwyżej w hierarchii będą te rozwiązania dla których funkcja celu zwróci najniższą wartość.

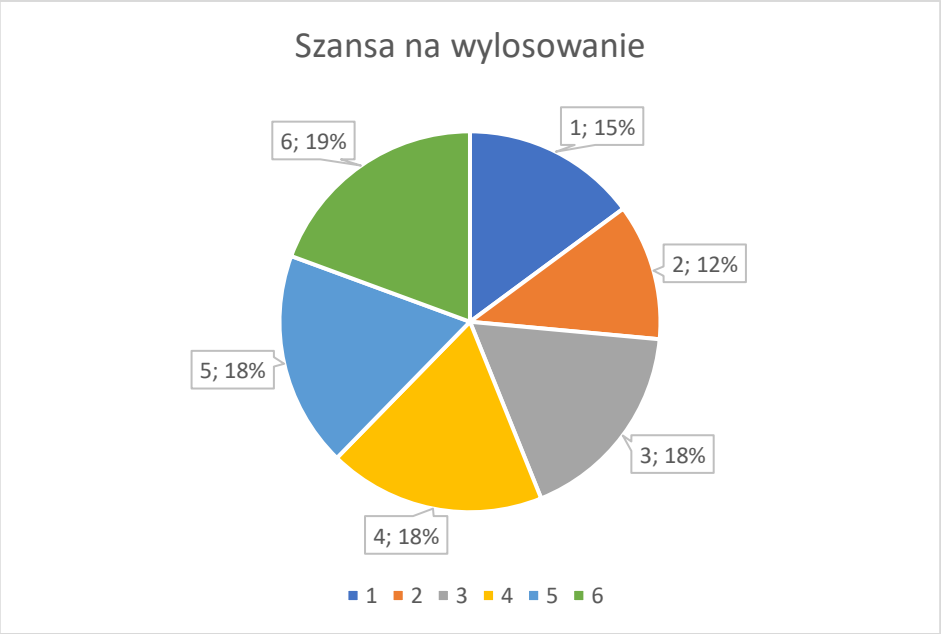
Tak jak w przypadku problemu plecakowego nasz program uwzględnia dodanie dwóch najlepszych osobników danej generacji do nowej generacji.

Następnie selekcja odbywa się w sposób kołowy. Oznacza to że każde rozwiązanie ma szansę zostać wybrane, jednak szansa ta jest zależna od wyniku jaki osiąga dane rozwiązanie. Jest to pewien element losowości i mechanizm mający na celu sprawić ze algorytm będzie imitował „szczęście".

*Działania selekcji kołowej w praktyce:*

| Numer rozwiązania | Wartość rozwiązania |
|---|---|
| 1 | 4200 |
| 2 | 6900 |
| 3 | 2100 |
| 4 | 1300 |
| 5 | 1400 |
| 6 | 500 |

Wykres szansy na wylosowanie jako rodzic




## Krzyżowanie

Krzyżowanie wymyśliliśmy w taki sposób że chromosomy zostają przecięty w połowie. Pierwsze dziecko dostaje pierwszą połowę pierwszego z rodziców a drugie pierwszą połowę z drugiego z rodziców. Następnie do pierwszego dziecka dodawane są brakujące geny w takiej kolejności w jakiego występuje to u rodzica drugiego. Analogicznie w przypadku drugiego dziecka i pierwszego rodzica.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|

| 4 | 2 | 0 | 6 | 9 | 1 | 3 | 5 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|

Rodzice:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|

| 4 | 2 | 0 | 6 | 9 | 1 | 5 | 8 | 7 | 3 |
|---|---|---|---|---|---|---|---|---|---|

Dzieci:

| 0 | 1 | 2 | 3 | 4 | 6 | 9 | 5 | 8 | 7 |
|---|---|---|---|---|---|---|---|---|---|

| 4 | 2 | 0 | 6 | 9 | 1 | 3 | 5 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|

## Mutacja

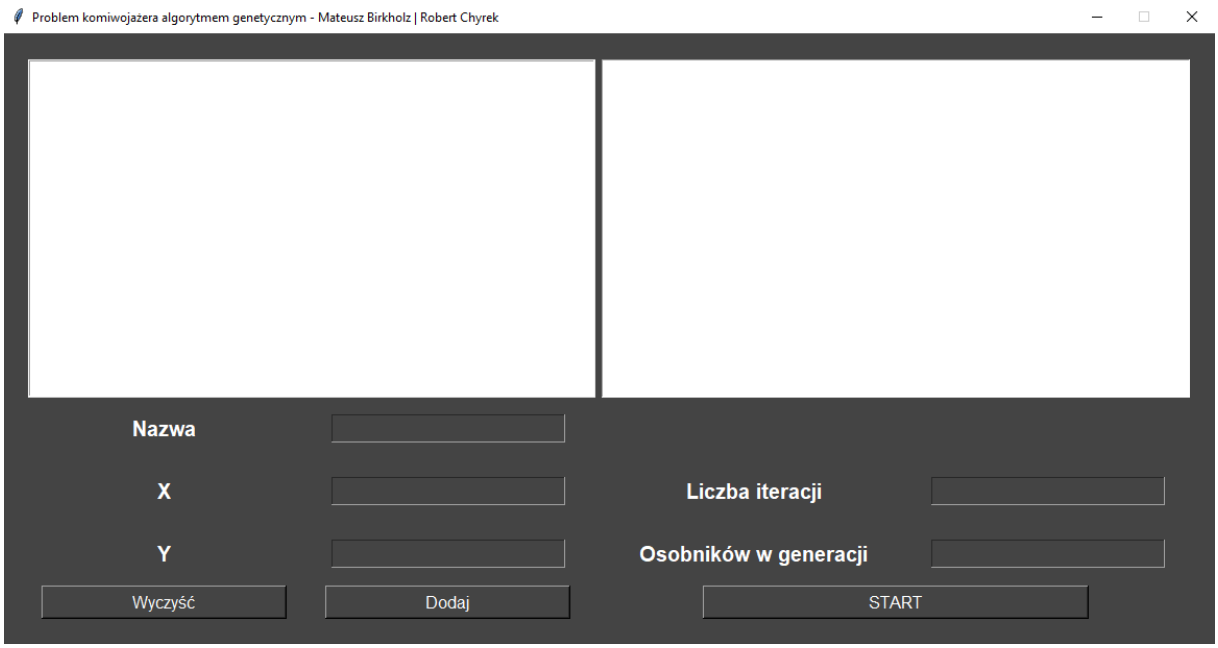
Mutacja odbywa się na zasadzie zamiany miejscami losowo wybranych 2 genów. Ilość mutacji ustawiona jest na 1 a prawdopodobieństwo jej wystąpienia na 30%. Funkcja mutująca uruchamia pętlę tyle razy ile mutacji ma zajść. Następuje losowanie czy dany gen zostanie poddany mutacji czy nie i w zależności od wyniku zostaje poddany mutacji lub nie.

## Wynik

Wynikiem działania algorytmu jest najnowsza generacja chromosomów posortowana według wartości funkcji celu. W związku z tym najlepszym rozwiązaniem będzie pierwszy chromosom w tej generacji.

## Opis działania programu

Okno programu wygląda następująco:




Po lewej stronie mamy możliwość dodawania własnych miast wraz z ich pozycjami. Po prawej zaś możemy ustawić liczbę iteracji oraz ilość osobników w generacji.

UWAGA! W programie na sztywno przypisaliśmy przykładowe miasta. Aby się ich pozbyć z puli wystarczy na początku działania programu kliknąć przycisk „Wyczyść”.

## Testowanie działania algorytmu

W Internecie udało nam się znaleźć stronę która generowała losowe punkty i obliczała najlepszą trasę pomiędzy nimi (link do strony w źródłach). Na tej stronie wygenerowaliśmy przykładowe punkty i przeprowadziliśmy testy działania aplikacji w podobny sposób co w przypadku problemu plecakowego.

Dane wyglądają następująco:

| Numer Miasta | X | Y |
|---|---|---|
| 1 | 835 | 98 |
| 2 | 930 | 356 |
| 3 | 825 | 805 |
| 4 | 716 | 143 |
| 5 | 396 | 333 |
| 6 | 110 | 348 |
| 7 | 124 | 307 |
| 8 | 405 | 873 |
| 9 | 154 | 529 |

Dla podanych punktów najlepszym rozwiązaniem jaki podał kalkulator na stronie była sekwencja 8-9-6-7-5-4-1-2-3-8.

Program zaczynamy testować najpierw na 10 iteracjach i 10 osobnikach w generacji. Otrzymywane wyniki było kompletnie różne od siebie w każdej próbie. Ich wartości oscylowały w granicach między 2807 a 4014.

Zwiększyliśmy zatem ilość generacji do 100.

Dla tych wartości program dalej wyrzucał dość mocno różniące się między sobą wyniki jednak tym razem ich wartości oscylowały w o wiele bardziej optymalnym zakresie 2590-3465.

Ponownie zwiększyliśmy ilość iteracji o 100.

W tym momencie – dla 200 iteracji i 10 osobników w generacji – program w zdecydowanej większości wyrzucał 3 powtarzające się wyniki:

| Sekwencja miast | Wartość rozwiązania |
|---|---|
| 9-8-3-2-4-1-5-7-6-9 | 2742.356730534938 |
| 9-7-6-5-4-1-2-3-8-9 | 2640.469776325456 |
| 7-6-9-8-3-2-1-4-5-7 | 2589.569962293973 |

Ostatni z wyników był sekwencją podaną przez kalkulator na stronie, jednak zostawał on zwracany tylko w 60% przypadków. W związku z tym zwiększyliśmy liczbę osobników w generacji do 20.

Jednak nie spowodowało to znacznego wzrostu występowania poprawnego wyniku. Postanowiliśmy zatem przywrócić liczbę osobników w generacji do 10 ale za to zwiększyć liczbę iteracji do 300. Efektem tego zabiegu był ten sam wynik który uzyskaliśmy w przypadku zwiększenia liczby osobników.

Sprawdziliśmy zatem wyniki dla 400 iteracji i 25 osobników. W ten sposób trafność wyników podskoczyła do 85% jednak to wciąż niezadowalający wynik.

Zwiększyliśmy zatem ilość iteracji do 500 a liczbę osobników do 30. Znacząco wydłużyło to czas pracy programu jednak uzyskaliśmy w ten sposób ponad 95% trafność. Pozwoliło to nam uznać że program działa prawidłowo.

# 4. Podsumowanie

Algorytm genetyczny świetnie sprawdza się w rozwiązywaniu problemów w których należy odnaleźć prawidłową kombinację lub sekwencję. Jest on do tego dobrze dostosowany zarówno pod względem działania jak i łatwości implementacji.

Pozwala on bardzo przyspieszyć rozwiązywanie problemów NP-trudnych których standardowe rozwiązywanie zajęłoby bardzo dużo czasu, szczególnie dla dużej liczby obiektów.

Program można przetestować za pomocą zapisanych w kodzie programu przykładowych danych na których przeprowadzaliśmy powyższe testy. Jeśli ktoś nie chce z nich korzystać wystarczy na początku działania programu wcisnąć przycisk „Wyczyść”,

## Wnioski:

- Mimo mniejszej puli danych oraz większej liczby iteracji i osobników niż w przypadku problemu plecakowego, algorytm rozwiązywania problemu komiwojażera wciąż miał problemy ze znalezieniem prawidłowego rozwiązania. Myślimy że może to być przez fakt iż mutacja w chromosomie tego algorytmu może wprowadzić o wiele większe zmiany w wartości ogólnej rozwiązania.

- W komiwojażerze należy przyjrzeć się wynikom ponieważ ten sam wynik może zostać różnie wypisany. Chodzi o to że algorytm nie uwzględnia w którym punkcie należy zacząć, a także w którą stronę się poruszać. W związku z tym należy patrzeć na kolejność następujących po sobie miast a nie tylko na to które z nich jest na którym miejscu.

- Algorytm komiwojażera zajmował zdecydowanie więcej czasu niż algorytm problemu plecakowego. Podejrzewamy że może to być przez znacznie bardziej skomplikowane obliczenia odległości między punktami a także trudniejszą do przekształcania budowę chromosomu.

- Czas jaki zajmuje znalezienie optymalnego wyniku w przypadku obu problemów rośnie wykładniczo, tym szybciej im więcej elementów jest w puli.

- Losowość znajduje zastosowanie również w algorytmach genetycznych. Dzięki niej możliwe jest mutowanie rozwiązań a także zdecydowanie mniej powtarzalne krzyżowanie się ze sobą chromosomów.

- Mimo niezbyt obiecujących wyników w pierwszych generacjach, rozwiązania dość szybko zbliżały się do osiągnięcia najbardziej optymalnego. Jednak im bliżej niego były tym większy spadek tej prędkości można było zaobserwować.

- Algorytmy genetyczne miewają problemy będąc bardzo blisko poprawnego rozwiązania. Czas jaki potrzebny jest na zbliżenie się do rozwiązania jest mniejszy niż czas potrzebny do jego dokładniejszego poznania.

# 5. Bibliografia


1) Kalkulator problemu TSP
https://www.lancaster.ac.uk/fas/psych/software/TSP/TSP.html

2) Przykładowe dane problemu plecakowego
https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html

3) Teoria na temat algorytmu genetycznego
https://www.youtube.com/watch?v=JgqBM7JG9ew

4) Opis problemu TSP
http://algorytmy.ency.pl/artykul/problem_komiwojazera

5) Opis problemu plecakowego
https://www.youtube.com/watch?v=BoVIsmjCkmg