# **Dokumentacja projektu:**

# **Algorytm genetyczny dla problemu plecakowego**

Autorzy: 51528 Maciej Morawiak, 49597 Alicja Żydek

1. Wprowadzenie

1. **Problem plecakowy**

Problem plecakowy (ang. *Knapsack Problem*) to jedno z klasycznych zagadnień optymalizacyjnych, w którym celem jest maksymalizacja sumy wartości przedmiotów umieszczonych w plecaku o ograniczonej pojemności. Każdy przedmiot posiada określoną wartość i wagę, a wybór przedmiotów musi spełniać ograniczenia wagowe.

Wariant 0-1 problemu plecakowego, używany w tym projekcie, wymaga podjęcia decyzji, czy dany przedmiot zostanie wybrany (1), czy nie (0).

1. **Algorytmy genetyczne**

Algorytmy genetyczne (AG) są inspirowane procesami ewolucji biologicznej, takimi jak selekcja naturalna, krzyżowanie i mutacja. AG są szczególnie przydatne w rozwiązywaniu problemów NP-trudnych, gdzie przestrzeń możliwych rozwiązań jest zbyt duża, aby przeszukać ją metodami wyczerpującymi. Kluczowe cechy AG to:

* **Populacja**: zbiór potencjalnych rozwiązań.
* **Funkcja fitness**: ocena jakości rozwiązań.
* **Selekcja**: wybór najlepszych osobników.
* **Krzyżowanie**: generowanie nowych rozwiązań na podstawie wybranych rodziców.
* **Mutacja**: losowe zmiany, wprowadzające różnorodność.

2. Struktura programu

**Ogólny opis**

Program został podzielony na trzy moduły:

1. **import\_data.py:**

wczytuje dane wejściowe z pliku tekstowego i przekształca je w strukturę danych potrzebną do rozwiązania problemu plecakowego. Klasa Knapsack zarządza tym procesem:

* + - **Inicjalizacja:** Podczas tworzenia obiektu wczytywane są dane z pliku
    - **Wczytywanie danych:** Pierwsza linia pliku zawiera pojemność plecaka i liczbę przedmiotów, a kolejne – wartości i wagi przedmiotów, które są przechowywane jako krotki (***wartość, waga***) w **self.items.**
    - **Wyświetlanie danych**: Metoda **display\_data** umożliwia łatwe podglądnięcie wczytanych danych.

Moduł przygotowuje dane w formie pozwalającej na dalsze operacje algorytmiczne.

1. **genetic\_algorithm.py:**   
   implementuje algorytm genetyczny do rozwiązywania problemu plecakowego. Klasa **GeneticAlgorithm** zarządza całym procesem optymalizacji, uwzględniając mechanizmy selekcji, krzyżowania i mutacji.
   * **Inicjalizacja**:
     + Ustawia parametry, takie jak rozmiar populacji, liczba generacji, metoda selekcji (tournament lub rank) oraz sposób krzyżowania (one-point lub two-point).
     + Tworzy początkową populację losowych rozwiązań (chromosomów).
   * **Funkcja przystosowania**:
     + Oblicza wartość rozwiązania, uwzględniając sumę wartości przedmiotów w plecaku.
     + Dodaje karę za przekroczenie pojemności plecaka, aby preferować poprawne rozwiązania.
   * **Selekcja:**
     + **Turniejowa**: Losowo wybiera grupę osobników i wybiera najlepszego.
     + **Rankingowa**: Sortuje populację według przystosowania i przypisuje wyższe prawdopodobieństwo wyboru osobnikom o wyższym rankingu.
   * **Krzyżowanie:**
     + Tworzy nowe rozwiązania, mieszając geny dwóch rodziców. Obsługuje krzyżowanie jedno- i dwupunktowe.
   * **Mutacja:**
     + Wprowadza losowe zmiany w chromosomach, aby zachować różnorodność populacji.
   * **Główna pętla algorytmu:**
     + Dla każdej generacji tworzy nową populację przez selekcję, krzyżowanie i mutację.
     + Co 10 generacji raportuje najlepsze rozwiązanie.
   * **Wyniki:**
     + Zwraca najlepsze znalezione rozwiązanie, jego wartość oraz dane do wizualizacji postępu optymalizacji.

Moduł ten skutecznie poszukuje optymalnego rozwiązania problemu plecakowego poprzez ewolucję populacji rozwiązań w kolejnych generacjach.

1. **main.py:**

integruje funkcjonalności z modułów **import\_data.py** i **genetic\_algorithm.py**, zapewniając interfejs użytkownika do uruchamiania algorytmu genetycznego na danych problemu plecakowego.

* + **Interakcja z użytkownikiem**:
    - Pobiera dane wejściowe od użytkownika, takie jak ścieżka do pliku z danymi, rozmiar populacji, liczba generacji, współczynnik mutacji, metoda selekcji i sposób krzyżowania.
    - Domyślne wartości parametrów są ustawione dla wygody użytkownika.
  + **Inicjalizacja i wyświetlanie danych**:
    - Wczytuje dane przy użyciu klasy Knapsack, a następnie wyświetla je w czytelnej formie za pomocą metody **display\_data().**
  + **Uruchomienie algorytmu genetycznego:**
    - Tworzy instancję klasy **GeneticAlgorithm** z odpowiednimi parametrami.
    - Rozpoczyna proces optymalizacji i mierzy czas jego wykonania.
  + **Prezentacja wyników:**
    - Wyświetla najlepsze rozwiązanie, jego wartość oraz czas wykonania algorytmu.
    - Zapisuje wyniki do pliku tekstowego (**wyniki.txt**) w celu ich późniejszego przeglądu.
  + **Wizualizacja wyników:**
    - Generuje wykres przedstawiający postęp algorytmu (najlepsza wartość rozwiązania w kolejnych generacjach).
    - Zapisuje wykres jako plik PNG (**best\_value\_plot.png**) i wyświetla go użytkownikowi.
  + **Obsługa wyjątków:**
    - Wykrywa potencjalne błędy, takie jak brak pliku lub niepoprawny format danych, i informuje użytkownika o problemie.

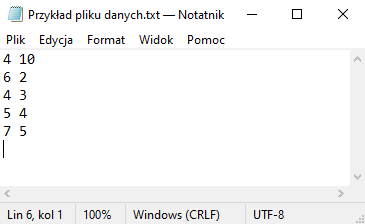
Moduł main.py działa jako punkt wejścia aplikacji, umożliwiając wygodne przeprowadzenie eksperymentu z wykorzystaniem algorytmu genetycznego i zapewniając szczegółowy wgląd w wyniki oraz parametry działania.

3. Przewodnik użytkownika

1. **Przygotowanie pliku danych**:

Dane wejściowe muszą być zapisane w pliku tekstowym zgodnie z poniższym formatem:

* **Pierwsza linia**: dwie liczby całkowite oddzielone spacją:
  + ***liczba\_przedmiotó****w* – liczba dostępnych przedmiotów.
  + ***pojemność\_plecaka*** – maksymalna pojemność plecaka.
* **Kolejne linie**: pary liczb całkowitych oddzielonych spacją, reprezentujące przedmioty:
  + ***wartość*** – wartość przedmiotu.
  + ***waga*** – waga przedmiotu.

**Przykład pliku danych**:  


* **4 przedmioty.**
* **Pojemność plecaka: 10.**
* **Przedmioty: (wartość = 6, waga = 2), (wartość = 4, waga = 3), (wartość = 5, waga = 4), (wartość = 7, waga = 5).**

**Uwagi:**

* Upewnij się, że plik danych jest poprawny i zgodny z formatem, aby uniknąć błędów podczas uruchamiania programu.
* Jeśli plik nie zostanie znaleziony lub format będzie niepoprawny, program poinformuje o błędzie.

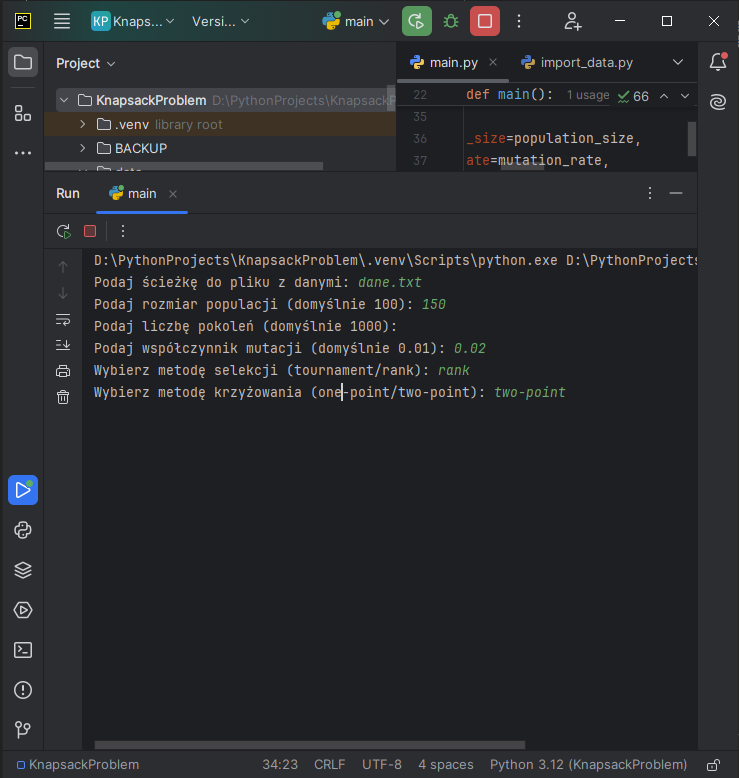
1. **Uruchomienie programu**:
   * Otwórz terminal i uruchom program poleceniem:

*python* ***main.py***

\*(Lub tak jak w naszym przypadku za pomocą zintegrowanego środowiska programistycznego „Pycharm”)

* + Wprowadź wymagane parametry, takie jak:
    - Ścieżkę do pliku z danymi (np. dane.txt).
    - Rozmiar populacji (domyślnie 100).
    - Liczbę pokoleń (domyślnie 1000).
    - Współczynnik mutacji (np. 0.01).
    - Metodę selekcji: **tournament** (domyślnie) lub **rank**.
    - Metodę krzyżowania: **one-point** lub **two-point** (domyślnie).

Przykład użycia:



**Wyjaśnienie wartości:**

* **Rozmiar populacji (10–200):** Balans między różnorodnością genetyczną a czasem obliczeń. Większe populacje dla złożonych problemów.
* **Liczba pokoleń (200–1000):** Umożliwia osiągnięcie konwergencji. Mniej dla prostszych zbiorów.
* **Współczynnik mutacji (0.01):** Zapewnia eksplorację przestrzeni rozwiązań bez nadmiernej przypadkowości.
* **Selekcja turniejowa:** Stabilna i efektywna, zwłaszcza dla złożonych zbiorów.
* **Krzyżowanie jednopunktowe:** Proste i szybkie, wystarczające dla większości przypadków.

1. **Generowanie wyników**:
   * **Najlepsze rozwiązanie**:   
     Po zakończeniu programu, w terminalu wyświetlone zostaną:
     + Najlepsze znalezione rozwiązanie (chromosom).
     + Wartość tego rozwiązania (całkowita wartość przedmiotów w plecaku).
     + Czas wykonania algorytmu.
   * **Zapis wyników do pliku**:   
     Wyniki są zapisywane w pliku tekstowym wyniki.txt, w formacie:

*Pojemnosc plecaka: 10, Wielkosc problemu: 4, Rozmiar populacji: 150, Liczba pokolen: 1000, Wspolczynnik mutacji: 0.02, Selekcja: rank, Krzyzowanie: two-point, Wartosc najlepszego rozwiazania: 16, ---*

Każde uruchomienie programu dopisuje wyniki na końcu pliku.

* + **Wizualizacje wyników:**   
    Program generuje wykres przedstawiający najlepszą wartość znalezioną w kolejnych pokoleniach. Wykres zostaje zapisany jako best\_value\_plot.png i wyświetlony na ekranie.
    - Opis na wykresie zawiera parametry uruchomienia programu i najlepszą wartość rozwiązania.

**Uwagi**:

* Wykres pozwala wizualnie ocenić konwergencję algorytmu i stabilność rozwiązania.
* Plik wyniki.txt oraz wykres best\_value\_plot.png są tworzone w bieżącym katalogu programu.

4. Eksperymenty

1. **Dane**

Eksperymenty przeprowadzono na zbiorach low-dimensional i large-scale. Przykładowe zbiory:

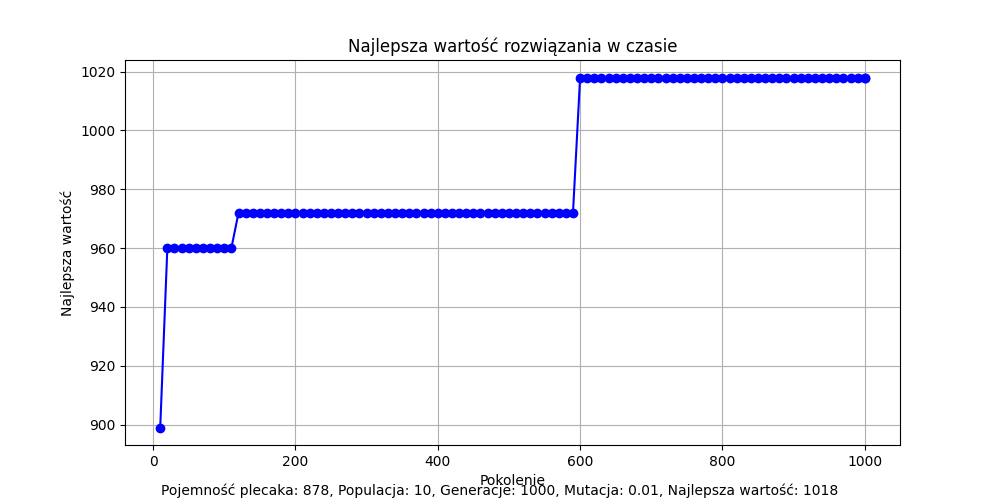
* **f2\_l-d\_kp\_20\_878**: 20 przedmiotów, pojemność plecaka 878, optimum: 1024.
* **knapPI\_1\_100\_1000\_1**: 100 przedmiotów, pojemność 1000, optimum: 9147.

1. **Wyniki**

* Dla zbioru **f2\_l-d\_kp\_20\_878**:
  + Selekcja turniejowa i krzyżowanie dwupunktowe: 1018 (blisko optimum).
  + Selekcja rankingowa i krzyżowanie jednopunktowe: 990.
* Dla zbioru **knapPI\_1\_100\_1000\_1**:
  + Selekcja turniejowa i krzyżowanie dwupunktowe osiągnęła wartość optymalną: 9147 (optimum).
  + Selekcja rankingowa i krzyżowanie jednopunktowe: 5972.

Więcej informacji i przykładów w części eksperymentalnej.

5. Wizualizacje

1. Wykresy konwergencji
   * Na wykresach przedstawiono najlepszą wartość w populacji w kolejnych pokoleniach.
   * Wyraźnie widać, że metoda selekcji ma wpływ na szybkość i jakość konwergencji:
     + **Selekcja turniejowa** prowadzi do szybszej i bardziej stabilnej konwergencji, często osiągając wartość bliską optimum.
     + **Selekcja rankingowa** jest bardziej podatna na wolniejszy wzrost wartości oraz niestabilność wyników, szczególnie w przypadku większych zbiorów danych.
2. Przykładowy wykres z selekcją turniejową: 
   * **Zbiór danych**: f2\_l-d\_kp\_20\_878 (20 przedmiotów, pojemność plecaka: 878).
   * **Parametry algorytmu**:
     + Rozmiar populacji: 10.
     + Liczba pokoleń: 1000.
     + Współczynnik mutacji: 0.01.
     + Selekcja: turniejowa.
     + Krzyżowanie: jednopunktowe.
   * Wyniki pokazują:
     + Wartość bliską optimum (1018) została osiągnięta już po około 300 pokoleniach.
     + Algorytm wykazuje stabilność – wartość ta nie spada w kolejnych iteracjach.
   * Uwagi:
     + Na podobnych wykresach dla selekcji rankingowej można zaobserwować wolniejszy wzrost oraz większe wahania wartości.

6. Wnioski i rekomendacje

* **Wnioski**

**Na podstawie przeprowadzonych eksperymentów stwierdzono:**

* + Selekcja turniejowa jest bardziej efektywna niż selekcja rankingowa, szczególnie w większych zbiorach danych, zapewniając:
    - Stabilność wyników, co oznacza mniejsze wahania najlepszej wartości w kolejnych pokoleniach.
    - Szybszą konwergencję, osiągając wyniki bliskie optimum w mniejszej liczbie iteracji.
  + Krzyżowanie dwupunktowe przynosi lepsze wyniki w bardziej złożonych przypadkach (np. dla większych rozmiarów problemu i populacji), podczas gdy krzyżowanie jednopunktowe jest wystarczające w prostszych zbiorach danych.
  + Algorytm genetyczny skutecznie znajduje rozwiązania bliskie optymalnym w rozsądnym czasie, co czyni go dobrym wyborem dla problemu plecakowego i podobnych zagadnień optymalizacyjnych.

7. Repozytorium GitHub

Projekt jest dostępny w publicznym repozytorium GitHub: [KnapsackProblem](https://github.com/mmorawiak/KnapsackProblem).

Repozytorium zawiera:

1. **Kod źródłowy**:
   * Moduły import\_data.py, genetic\_algorithm.py, main.py
2. **Przykładowe dane wejściowe**:
   * Przykładowe pliki z danymi testowymi w folderze data/, które można wykorzystać do uruchomienia programu.
3. **Wyniki eksperymentów**:
   * Zapisane wyniki działania programu (np. wyniki.txt) oraz wygenerowane wykresy konwergencji.
4. **Dokumentacja**:
   * Obecna dokumentacja w formacie Markdown (README.md).
   * Można znaleźć również plik który właśnie czytasz! 😊