Akademia Górniczo-Hutnicza w Krakowie
Wydział Fizyki i Informatyki Stosowanej
Informatyka Stosowana
Bartosz Rogowski, V rok, 303793
25 stycznia 2023

Algorytmy genetyczne

Sprawozdanie z projektu

Spawanie trójkątów

Spis treści

---

1 Opis zadania

Problem polega na ułożeniu prętów znajdujących się na taśmie w taki sposób, aby utworzyć z kolejnych trzech trójkąt oraz aby odchylenie standardowe pól otrzymanych w ten sposób trójkątów było jak najmniejsze. Jeśli z prętów nie da się stworzyć trójkąta, są one odrzucane i następne 3 są brane pod uwagę.

Zakłada się, że liczba prętów – n – jest liczbą naturalną dodatnią nie większą niż 1002 oraz, że długość pręta jest liczbą rzeczywistą z przedziału $(0, 50]$.

2 Implementacja problemu

Program został napisany w języku *Python* z pomocą biblioteki *PyGAD* [1, 2]. Napisanie algorytmu genetycznego z pomocą tego modułu jest proste i intuicyjne, bowiem w najprostszym przypadku wystarczy zaimplementować funkcję dostosowania (ang. *fitness function*) oraz szereg parametrów potrzebnych do zdefiniowania problemu [3], m. in.: liczbę epok, sposób reprodukcji (metoda selekcji, rodzaj krzyżowania, liczba osobników wybierana jako rodzice etc.), kodowanie osobników (populacja początkowa, typ genu, zakres wartości, długość chromosomów etc.), sposób mutacji, prawdopodobieństwa operacji (mutacji, krzyżowania), kontrola elityzmu itp.

2.1 Parametry używane w programie

W programie przyjęto następujące parametry:

- `num_generations = 200` – liczba epok (generacji)
- `num_parents_mating = 50` – liczba osobników wybieranych jako rodzice (do reprodukcji)
- `sol_per_pop = 200` – liczba osobników/chromosomów w populacji
- `num_genes` – liczba genów jednego osobnika (równa n)
- `gene_type = np.int16` – typ genu (w tym przypadku: liczba całkowita 16-bitowa)
- `mutation_probability = 5e-2` – prawdopodobieństwo mutacji (w tym przypadku: równe $5 \cdot 10^{-2} = 0.05$,
- `mutation_type = "swap"` – typ mutacji (w tym przypadku: **mutacja przez zamianę** dwóch genów w chromosomie)
- `crossover_type = partially_matched_crossover` – typ krzyżowania (w tym przypadku: **PMX (ang. *partial matched crossover*)** – własna implementacja*)
- `allow_duplicate_genes = False` – flaga zezwalająca na powielanie genów w chromosomie (w tym przypadku: nie zezwala – ze względu na naturę rozwiązywanego problemu)
- `parent_selection_type = "tournament"` – typ selekcji osobników (rodziców) do reprodukcji (w tym przypadku: **metoda turniejowa** – domyślnie $k = 3$ osobników)
- `gene_space` – możliwe wartości genów w obrębie jednego osobnika (w tym przypadku: liczby całkowite od 1 do n)
- `keep_elitism = 10` – liczba osobników, które wejdą w skład kolejnego pokolenia (w tym przypadku: 10 osobników)
- `fitness_func` – metoda obliczająca wartość funkcji przystosowania dla osobnika.

*Moduł `PyGAD` posiada ubogi wybór wbudowanych typów krzyżowania. Można natomiast definiować metody napisane przez użytkownika i w prosty sposób „przyłączać" do głównej instancji algorytmu [4]. Przykład działania takiego operatora: `https://www.researchgate.net/figure/Example-of-partially-mapped-crossover_fig1_312336654`.

2.2 Funkcja przystosowania

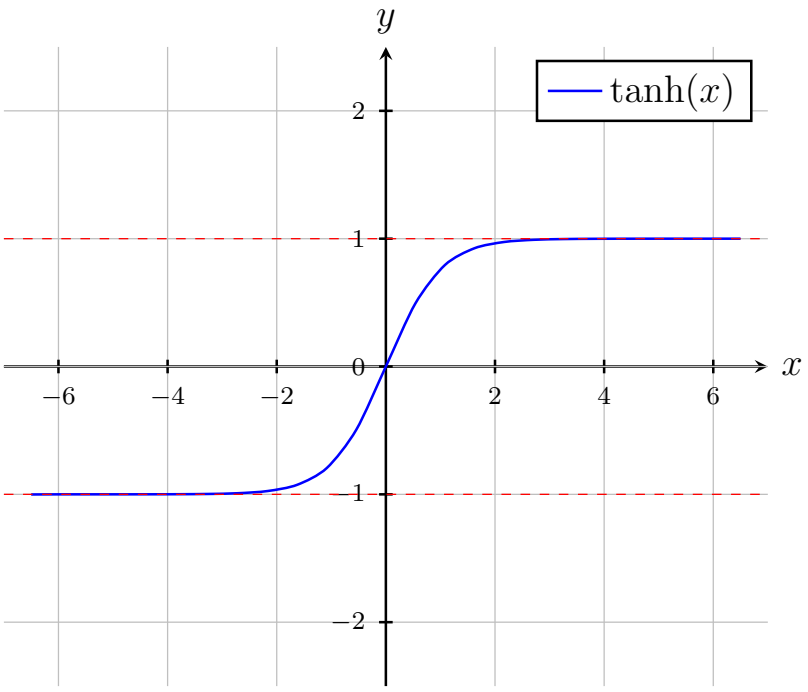
Funkcja przystosowania zawiera wszystkie parametry, które są poddawane optymalizacji; została dobrana empirycznie i ma postać:

$$f = 0.5 \cdot \left(3 \cdot \frac{t}{n} + \tanh\left(\frac{\alpha}{\sigma}\right)\right), \tag{2.1}$$

gdzie:

- t – liczba utworzonych trójkątów
- α – stała empiryczna, czułość na odchylenie standardowe (w programie: $\underline{\alpha = 10}$)
- σ – odchylenie standardowe pól utworzonych trójkątów (jeśli wynosi lub jest bardzo bliskie* 0 bądź, gdy jest to wartość niezdefiniowana (wartość `NaN` w *Pythonie*), to wówczas przyjmuje wartość 100 000 – zabezpiecza to przed potencjalnymi błędami w danych.

a jej zbiór wartości to $[0, 1]$.



Rysunek 2.1: Wykres funkcji $\tanh(x)$

2.3 Funkcjonalność programu

Program został rozdzielony na pliki (katalog `src/`):

- ***`main.py`*** – główny skrypt, zawiera funkcje:
  - `main(filepath: str)` – główna funkcja odczytująca dane z pliku, wykonująca algorytm genetyczny, wyświetla statystyki po znalezieniu najlepszego rozwiązania i zapisuje je do pliku `output.txt`

*co najwyżej rzędu 10^{-6}

  - `run_main_n_times(repeats: int, filepath: str, plot_stats: bool = True)` – wykonuje algorytm genetyczny `repeats` razy i w zależności od flagi `plot_stats` tworzy wykres rozwiązań w przestrzeni parametrów (w celach testowych)

- *`genetic_algorithm.py`* – zawiera klasę będącą opakowaniem na klasę `GA` z modułu *PyGAD*

- *`tools.py`* – zawiera funkcje pomocne w programie:

  - `load_input_file(filepath: str = "prety.txt")` – wczytuje plik wejściowy z podanej ścieżki
  - `generate_input_file(filepath: str, n: int)` – generuje plik wejściowy zgodny z założeniami projektowymi
  - `save_solution_to_file(solution: np.ndarray, filepath=None)` – zapisuje podane rozwiązanie to pliku
  - `can_build_triangle(a: float, b: float, c: float) -> bool` – sprawdza czy z podanych odcinków można utworzyć trójkąt
  - `calculate_triangle_area(a: float, b: float, c: float) -> float` – liczy pole trójkąta według wzoru Herona
  - `get_stats_of_solution(solution, rodes_lengths)` – zwraca statystyki (liczba zbudowanych trójkątów, odchylenie standardowe, wartość funkcji przystosowania) dla podanego rozwiązania
  - `partially_matched_crossover(parents, offspring_size, ga_instance)` – implementacja algorytmu PMX.

W folderze `inputs` umieszczono dodatkowe pliki wejściowe.
Instrukcja instalacji i uruchomienia programu znajduje się w pliku `README.md`.

3 Wyniki

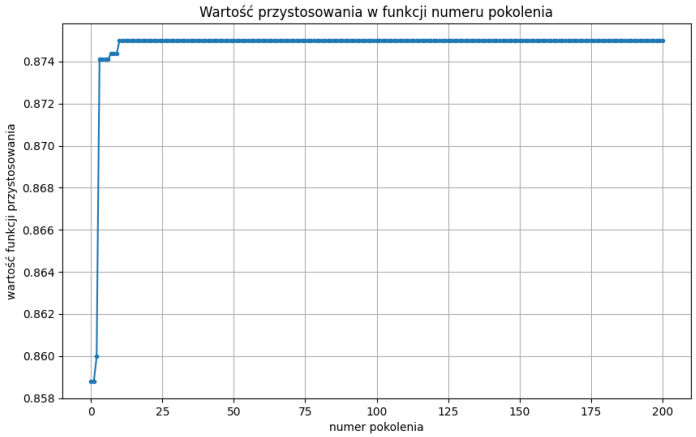
Przykładowy output* zwracany przez program (dla pliku wejściowego: `prety.txt`) został pokazany na rys. 3.1.

```
best_solution_fitness = 0.8749975918637027
triangles = 3, st_dev = 1.546, areas.mean() = 18.104
Program executed in 5.890 seconds
```

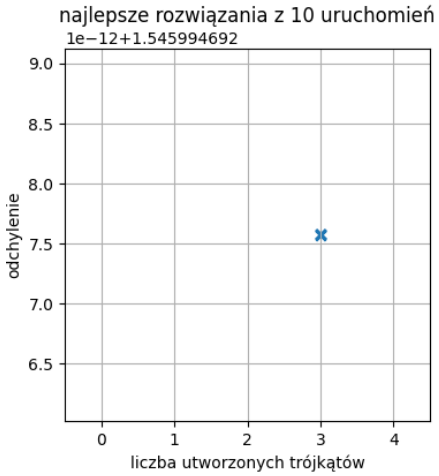
Rysunek 3.1: Zrzut ekranu z konsoli

Dla kilku plików wejściowych uruchomiono program 10 razy, aby zbadać elastyczność programu na różne dane. Dołączono także wykres zależności wartości funkcji przystosowania w funkcji numeru pokolenia (generacji) dla najlepszych osobników.

*Zawiera on podstawowe statystyki, jak również dodatkową zmienną – średnie pole utworzonych trójkątów – wyłącznie w celach informacyjnych.

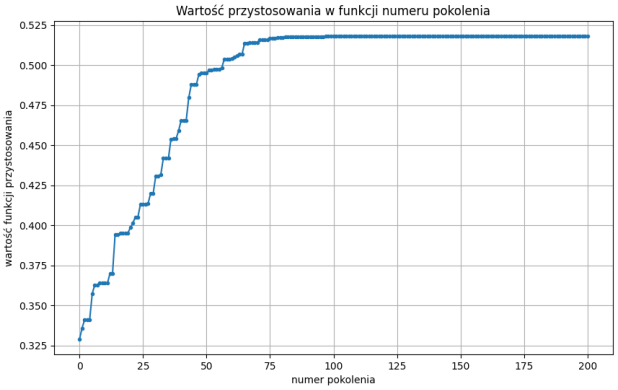


(a) funkcja przystosowania a numer pokolenia

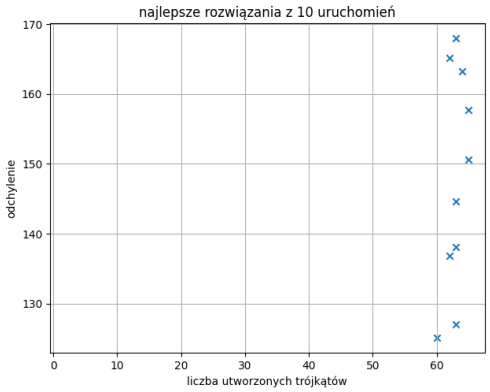


(b) rozwiązania z 10 uruchomień

Rysunek 3.2: Statystyki dla pliku wejściowego `prety.txt`

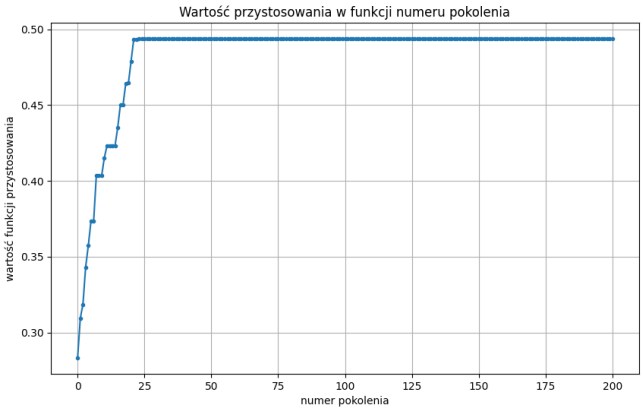


(a) funkcja przystosowania a numer pokolenia

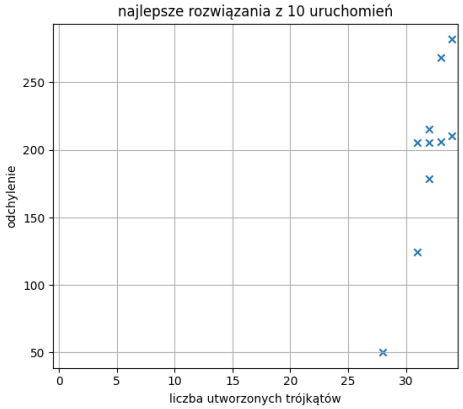


(b) rozwiązania z 10 uruchomień

Rysunek 3.3: Statystyki dla pliku wejściowego `prety1.txt`

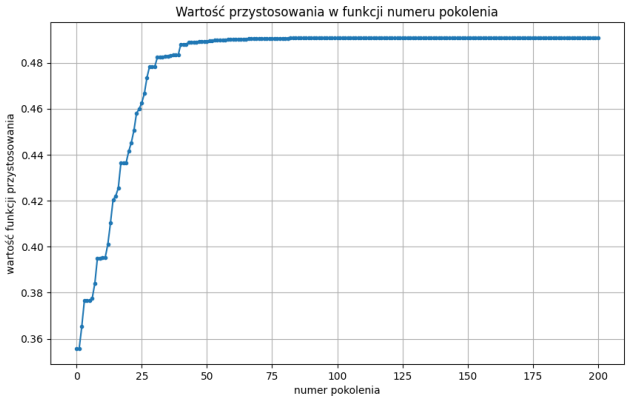


(a) funkcja przystosowania a numer pokolenia

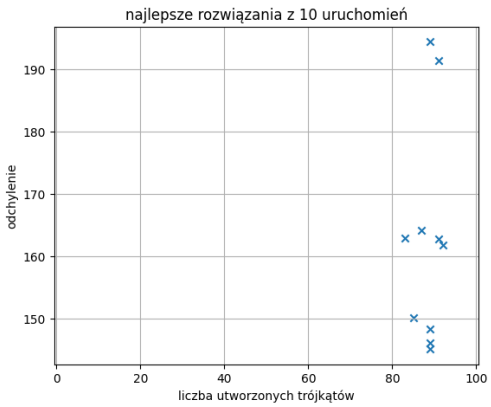


(b) rozwiązania z 10 uruchomień

Rysunek 3.4: Statystyki dla pliku wejściowego `prety4.txt`



(a) funkcja przystosowania a numer pokolenia



(b) rozwiązania z 10 uruchomień

Rysunek 3.5: Statystyki dla pliku wejściowego `myinput_300.txt`

4 Obserwacje, wnioski

- Za pomocą algorytmu genetycznego można znaleźć rozwiązanie nietrywialnego problemu, który często nie posiada analitycznego najlepszego rozwiązania (bądź jest to żmudne).
- Wszystkich możliwych rozwiązań jest $n!$ – to bardzo duża przestrzeń rozwiązań; już dla $n = 13$ jest to ponad 6 miliardów kombinacji.
- Problem, pomimo, że może się to wydawać nieoczywiste na pierwszy rzut oka, jest bardzo zbliżony do problemu komiwojażera, ponieważ ważna jest kolejność ułożenia indeksów (kolejność „odwiedzania" prętów – analogia odwiedzania miast).
- Program daje zadowalające wyniki dla wybranych plików wejściowych. Może o tym świadczyć ustalenie (zbieżność) funkcji przystosowania do pewnej wartości przez wiele epok. Oczywiście lepsze wyniki dalej mogą być znalezione (można zmienić parametry programu, chociażby wydłużyć liczbę epok), co potwierdzają też różnorodne rozwiązania (nie wszystkie są niezdominowane), jednak wydaje się, że w porównaniu do szybkości działania (dla większych plików zwykle jest to do kilku minut na jedno uruchomienie), jest to satysfakcjonujący wynik.
- Moduł *PyGAD* wydaje się dobrym narzędziem do tworzenia algorytmów genetycznych w języku *Python*. Jednak jedną z jego wad jest szybkość działania – operacje są napisane natywnie w *Pythonie*, nie można ich przyspieszyć nawet za pomocą modułu *numba**.

5 Bibliografia

[1] Ahmed Fawzy Gad. *PyGAD: An Intuitive Genetic Algorithm Python Library*. 2021. arXiv: `2106.06158 [cs.NE]`.

[2] *PyGAD – Python Genetic Algorithm!* URL: `https://pygad.readthedocs.io/en/latest/#`. (data dostępu: 21 stycznia 2023).

[3] *`pygad` Module*. URL: `https://pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html`. (data dostępu: 21 stycznia 2023).

*Moduł pozwala na przyspieszenie wykonywanego kodu (często bardzo znaczne) poprzez translacje bezpośrednio na kod maszynowy.

[4] *User-Defined Crossover Operator*. URL: https://pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html#user-defined-crossover-operator. (data dostępu: 21 stycznia 2023).