Patryk Morawski  
Index: 288115

Inteligencja Obliczeniowa  
Monogram

Generowanie monogramu bazując na pixelart’cie  
Rozwiązywanie kolorowego monogramu  
Statystyka wpływu rozmiaru oraz ilości kolorów na czas rozwiązania

Spis treści

[1. Wykorzystane biblioteki 3](#_Toc125535845)

[2. Generowanie Monogramu na podstawie pixelart-u 4](#_Toc125535846)

[2.1. Wczytywanie obrazu 4](#_Toc125535847)

[2.2. Przygotowanie palety 5](#_Toc125535848)

[2.3. Przygotowanie obrazu 6](#_Toc125535849)

[2.4. Generowanie rozwiązania 7](#_Toc125535850)

[3. Rozwiązywanie Monogramu 9](#_Toc125535851)

[3.1. Parametry 9](#_Toc125535852)

[3.2. Fitness 11](#_Toc125535853)

[3.3. Uruchomienie instancji 12](#_Toc125535854)

[4. Statystyka 13](#_Toc125535855)

[4.1. Badanie wpływu rozmiaru na czas rozwiązywania monogramu 13](#_Toc125535856)

[4.2. Badanie wpływu ilości kolorów na czas rozwiązywania monogramu 17](#_Toc125535857)

[4.3. Wnioski 21](#_Toc125535858)

# Wykorzystane biblioteki

**PyGAD** – to otwarta biblioteka Pythona do budowania algorytmu genetycznego i optymalizacji algorytmów uczenia maszynowego.

**NumPy** – to podstawowy pakiet do obliczeń naukowych w Pythonie. Jest to biblioteka Pythona, która udostępnia wielowymiarowy obiekt tablicowy, różne obiekty pochodne (takie jak zamaskowane tablice i macierze) oraz zestaw procedur do szybkich operacji na tablicach, w tym matematycznych, logicznych, manipulacji kształtami, sortowania, wybierania, I/O , dyskretne transformaty Fouriera, podstawy algebry liniowej, podstawowe operacje statystyczne, symulacje losowe i wiele więcej.

**ImageIO** – to biblioteka Pythona, która zapewnia łatwy interfejs do odczytu i zapisu szerokiego zakresu danych obrazu, w tym obrazów animowanych, danych wolumetrycznych i formatów naukowych.

**DiffLab** – moduł udostępnia klasy i funkcje do porównywania sekwencji. Może być używany na przykład do porównywania plików i może generować informacje o różnicach plików w różnych formatach, w tym HTML i kontekst oraz ujednolicone różnice.

# Generowanie Monogramu na podstawie pixelart-u

## Wczytywanie obrazu

Przy pomocy metody imread biblioteki ImageIO wczytujemy obraz do tablicy 3 wymiarowej. Kolejne wymiary mają rozmiar kolejno wysokości i szerokości obrazu oraz głębi kolorów (3 dla kanałów kolorów – czerwony, zielony i niebieski - oraz ewentualnie 4 kanał przezroczystości o ile plik został zapisany w takim formacie).

Kod:

raw\_img = np.array(iio.imread("assets/test.png"))

Wynik:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (255,255,255) | (255,255,255) | (34, 177, 76) | (255,255,255) | (255,255,255) |
| (255,255,255) | (34, 177, 76) | (34, 177, 76) | (34, 177, 76) | (255,255,255) |
| (255,255,255) | (34, 177, 76) | (34, 177, 76) | (34, 177, 76) | (255,255,255) |
| (34, 177, 76) | (34, 177, 76) | (34, 177, 76) | (34, 177, 76) | (34, 177, 76) |
| (255,255,255) | (255,255,255) | (185, 122, 87) | (255,255,255) | (255,255,255) |

## Przygotowanie palety

Z obrazu pobierane są unikatowe kolory i umieszczane w tablicy. Kolory są sortowane rosnąco względem kolejnych kanałów (czerwony, zielony, niebieski i ewentualnie przezroczystość).

Kod:

pixels = np.reshape(raw\_img, (raw\_img.shape[0] \* raw\_img.shape[1], raw\_img.shape[2]))

color\_palette = np.unique(pixels, axis=0)

Skrypt generujący rozwiązanie monogramu traktuje kolor o indeksie 0 jako niezamalowane pole. Udostępniona jest funkcjonalność pozwalająca na ustalenie który kolor ma być traktowany jako „pusty”. Funkcja z założenia umieszcza wskazany kolor na początku listy (jeżeli kolor już istnieje w liście nie jest on duplikowany tylko umieszczony na pozycji 0).

Kod:

def set\_nonfilled\_color(color):

    \_color = np.array(color, dtype=np.uint8)

    if \_color.shape[0] > color\_palette.shape[1]:

        for n in range(\_color.shape[0] - color\_palette.shape[1]):

            \_color = np.delete(\_color, \_color.shape[0] - 1)

    else:

        for n in range(color\_palette.shape[1] - \_color.shape[0]):

            \_color = np.append(\_color, [255])

    new\_color\_palette = [\_color]

    for color\_in\_palette in color\_palette:

        if not (color\_in\_palette == \_color).all():

            new\_color\_palette = np.vstack([new\_color\_palette, color\_in\_palette])

    return new\_color\_palette

#Dodawanie koloru oznaczającego puste pola

color\_palette = set\_nonfilled\_color([255,255,255])

Wynik:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 0 | (34, 177, 76) | | 1 | (185, 122, 87) | | 2 | (255,255,255) | | ⇒ | |  |  | | --- | --- | | 0 | (255,255,255) | | 1 | (34, 177, 76) | | 2 | (185, 122, 87) | |

## Przygotowanie obrazu

Na podstawie przygotowanej palety obraz jest zamieniany na tablicę dwuwymiarową o identycznej szerokości i wysokości jaką posiadał obraz. Konkretne pola są uzupełniane indeksami kolorów z palety dla odpowiednich pikseli.

Kod:

def find\_color(color):

    index = 0

    for color\_in\_palette in color\_palette:

        if np.array\_equal(color\_in\_palette, color):

            return index

        index += 1

    return None

def use\_palette\_on\_img(\_img):

    tmp\_img = np.full((\_img.shape[0], \_img.shape[1]), 0)

    for col in range(\_img.shape[0]):

        for row in range(\_img.shape[1]):

            tmp\_img[col][row] = find\_color(\_img[col][row])

    return tmp\_img

img = use\_palette\_on\_img(raw\_img)

Wynik:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 2 | 0 | 0 |

## Generowanie rozwiązania

Na podstawie sformatowanego obrazu przy pomocy palety generowane jest rozwiązanie monogramu. Rozwiązanie jest przechowywanie w postaci tablicy dwuwymiarowej. Pierwszy wymiar rozdziela rozwiązanie wierszy oraz rozwiązanie kolumn. Drugi wymiar to poszczególne wiersze lub odpowiednio kolumny. Zawieraj one listy z strukturami w formacie „{kolor, ilość}”.

Kod:

def nonogram\_solution\_build(\_img):

    answer = [[],[]]

    # Wiersze

    for col in range(\_img.shape[0]):

        recipe = []

        color = 0

        count = 0

        for row in range(\_img.shape[1]):

            if \_img[col][row] != 0:

                if \_img[col][row] == color:

                    count += 1

                else:

                    if color != 0 and count != 0:

                        recipe.append([color, count])

                    color = \_img[col][row]

                    count = 1

            else:

                if color != 0 and count != 0:

                    recipe.append([color, count])

                color = 0

                count = 0

            if row == \_img.shape[1] - 1:

                if color != 0 and count != 0:

                    recipe.append([color, count])

        answer[0].append(recipe)

    # Kolumny

    for col in range(\_img.shape[1]):

        recipe = []

        color = 0

        count = 0

        for row in range(\_img.shape[0]):

            if \_img[row][col] != 0:

                if \_img[row][col] == color:

                    count += 1

                else:

                    if color != 0 and count != 0:

                        recipe.append([color, count])

                    color = \_img[row][col]

                    count = 1

            else:

                if color != 0 and count != 0:

                    recipe.append([color, count])

                color = 0

                count = 0

            if row == \_img.shape[1] - 1:

                if color != 0 and count != 0:

                    recipe.append([color, count])

        answer[1].append(recipe)

    return answer

nonogram\_solution = nonogram\_solution\_build(img)

Wynik:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | {1, 4} |  |  |
| {1, 1} | {1, 3} | {2, 1} | {1, 3} | {1, 1} |
|  | {1, 1} | 0 | 0 | 1 | 0 | 0 |
|  | {1, 3} | 0 | 1 | 1 | 1 | 0 |
|  | {1, 3} | 0 | 1 | 1 | 1 | 0 |
|  | {1, 5} | 1 | 1 | 1 | 1 | 1 |
|  | {2, 1} | 0 | 0 | 2 | 0 | 0 |

# Rozwiązywanie Monogramu

## Parametry

**Przestrzeń genów (gene\_space)** – w przypadku kolorowego monogramu są to indeksy wszystkich kolorów w palecie. Jako że kolory są numerowane od 0 bez przerw to przestrzeń genów jest wyliczeniem wszystkich liczb całkowitych od zera do długości palety.

Kod:

gene\_space = [\*range(len(color\_palette))]

**Ilość genów (num\_genes)** – to ilość pikseli z jakich składa się dany obraz.

Kod:

num\_genes = img.shape[0] \* img.shape[1]

**Rozwiązania na populację (sol\_per\_pop)** – ilość wygenerowanych chromosomów (rozwiązań) na generację.

Kod:

sol\_per\_pop = 10

**Ilość generacji (num\_generations)** – maksymalna ilość symulowanych generacji.

Kod:

num\_generations = 1000

**Ilość rodziców użytych do generowania kolejnej populacji (num\_parents\_mating)**

Kod:

num\_parents\_mating = 5

**Ilość pozostawianych (keep\_parents)** – ilość rodziców którzy trafiają do nowej generacji.

Kod:

keep\_parents = 2

**Typ mieszania rodziców (crossover\_type)** – metoda którą są mieszane chromosomy rodziców. Biblioteka PyGAD udostępnia metody single\_pioint (podział względem jednego punktu), two\_points (podział względem dwóch punktów), scattered (podział względem losowego punktu), uniform (jednolity podział - maska) lub None dla braku mieszania.

Kod:

crossover\_type = "scattered"

**Typ mutacji (mutation\_type) –** sposób modyfikowania wygenerowanych chromosomów. Biblioteka PyGAD udostępnia metody random (losowa), swap (zamiana), inversion (inwersja), scramble (mieszania), adaptive (adaptacyjny) lub None dla braku mutacji.

Kod:

mutation\_type = "random"

**Prawdopodobieństwo mutacji (mutation\_percent\_genes) –** szansa na zajście mutacji

Kod:

mutation\_percent\_genes = 1

**Typ oceniania rodziców (parent\_selection\_type)** – metoda wybierania rodziców do wygenerowania nowej populacji. Biblioteka PyGAD udostępnia metody sss (stan ustalony), rws (ruletka), sus (stochastyczna uniwersalna selekcja), rank (ranking), random (losowo) i turnament (turniejowy).

Kod:

parent\_selection\_type = 'rank'

## Fitness

Metoda oceny (fitness) skonstruowana jest na zasadzie przygotowania rozwiązania wygenerowanego monogramu (solution). Następnym krokiem jest porównanie odpowiednio kolumny lub wiersza wygenerowanego rozwiązania z rozwiązaniem oryginalnego obrazu. Przygotowane są dodatkowe parametry do modyfikowania działania metody fitness:

Kod:

acceptable\_threshold = 0.0

calc\_fitness\_on\_similarity\_cond = True

Odpowiadają one kolejno za akceptowalny próg podobieństwa które jest sumowane do oceny rozwiązania oraz flaga włączająca wyłączjąca ocenę w formacie zmiennoprzecinkowym.

Kod:

def fitness(solution, solution\_idx):

    fitness\_solution\_nonogram = nonogram\_solution\_build(

        solution.reshape((img.shape[0], img.shape[1])))

    fitness = 0

    d = 0

    for direction in fitness\_solution\_nonogram:

        l = 0

        for line in direction:

            line\_flat = np.array(line).flatten()

            sol\_flat = np.array(nonogram\_solution[d][l]).flatten()

            similarity = SequenceMatcher(None, line\_flat, sol\_flat).ratio()

            if similarity >= acceptable\_threshold:

                if calc\_fitness\_on\_similarity\_cond:

                    fitness += similarity

                else:

                    fitness += 1

            l += 1

        d += 1

    return -(img.shape[0] + img.shape[1]) + fitness;

## Uruchomienie instancji

Przygotowanie instancji algorytmu generycznego na podstawie przygotowanej konfiguracji. Dodatkowo ustalenie kryteriów zatrzymania algorytmu. Saturacja którą można pominąć ponieważ określa maksymalną ilość generacji (w tym przypadku identyczna jak maksymalna ilość generacji) oraz uzyskanie określonego wyniku (0 czyli 100% zgodności)

Kod:

ga\_instance = pygad.GA(gene\_space=gene\_space,

                       num\_generations=num\_generations,

                       num\_parents\_mating=num\_parents\_mating,

                       fitness\_func=fitness,

                       sol\_per\_pop=sol\_per\_pop,

                       num\_genes=num\_genes,

                       parent\_selection\_type=parent\_selection\_type,

                       keep\_parents=keep\_parents,

                       crossover\_type=crossover\_type,

                       mutation\_type=mutation\_type,

                       mutation\_percent\_genes=mutation\_percent\_genes,

                       stop\_criteria = ["saturate\_1000", "reach\_0"])

ga\_instance.run()

# Statystyka

## Badanie wpływu rozmiaru na czas rozwiązywania monogramu

Test wpływu rozmiaru na czas rozwiązywania monogramu. Ustalona jest stała ilość kolorów równa 4 oraz przeprowadzone są próby dla „łatwego” oraz „trudnego” obrazu. Wykonywane jest 100 prób. Metoda fitness ocenia jedynie kolumnę lub wiersz które są identyczne z oryginalnym rozwiązaniem.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | 3x3 | 4x4 | 5x5 | 6x6 | 7x7 |
| Łatwy | Procent poprawnych rozwiązań | 100% | 100% | 71% | 46% | 3% |
| Minimalny czas | 1.98s | 4.99s | 16.51s | 14.44s | 4.86s |
| Maksymalny czas | 26.51s | 171.18s | 200.02s | 199.99s | 204.5s |
| Średni  czas | 7.96s | 34.99s | 74.67s | 83.1s | 119.06s |
| Minimalna ilość generacji | 8 | 23 | 99 | 270 | 637 |
| Maksymalna ilość generacji | 115 | 805 | 990 | 977 | 998 |
| Średnia ilość generacji | 33 | 157 | 421 | 658 | 799 |
| Minimalne podobieństwo | 100% | 100% | 36% | 22.22% | 16.33% |
| Maksymalne podobieństwo | 100% | 100% | 100% | 100% | 100% |
| Średnie podobieństwo | 100% | 100% | 92.92% | 75.69% | 50.29% |
| Trudny | Procent poprawnych rozwiązań | 100% | 78% | 53% | 77% | 2% |
| Minimalny czas | 1.5s | 2.52s | 7.01s | 23.9s | 173.49s |
| Maksymalny czas | 20.06s | 229.72s | 190.5s | 209.2s | 208.56s |
| Średni  czas | 7.35s | 50.61s | 71.41s | 78.1s | 191.02s |
| Minimalna ilość generacji | 7 | 15 | 83 | 153 | 872 |
| Maksymalna ilość generacji | 78 | 972 | 996 | 991 | 973 |
| Średnia ilość generacji | 30 | 244 | 430 | 437 | 922 |
| Minimalne podobieństwo | 100% | 50% | 20% | 25% | 30.61% |
| Maksymalne podobieństwo | 100% | 100% | 100% | 100% | 100% |
| Średnie podobieństwo | 100% | 91.68% | 83.6% | 92.58% | 69.43% |

Test dla zmodyfikowanej metody fitness która przyjmuje ocenę podobieństwa rozwiązania z zakresu 0.0 do 1.0.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | 3x3 | 4x4 | 5x5 | 6x6 | 7x7 |
| Łatwy | Procent poprawnych rozwiązań | 100% | 100% | 78% | 44% | 4% |
| Minimalny czas | 1.35s | 3.06s | 11.67s | 14.75s | 40.42s |
| Maksymalny czas | 36.92s | 218.2s | 215.16s | 197.66s | 170.27s |
| Średni  czas | 8.25s | 39.38s | 85.75s | 87.93s | 95.79s |
| Minimalna ilość generacji | 5 | 14 | 59 | 294 | 279 |
| Maksymalna ilość generacji | 161 | 974 | 989 | 992 | 793 |
| Średnia ilość generacji | 34 | 176 | 463 | 609 | 516 |
| Minimalne podobieństwo | 100% | 100% | 48% | 11.11% | 24.49% |
| Maksymalne podobieństwo | 100% | 100% | 100% | 100% | 100% |
| Średnie podobieństwo | 100% | 100% | 94.96% | 77.94% | 49.1% |
| Trudny | Procent poprawnych rozwiązań | 100% | 72% | 59% | 67% | 1% |
| Minimalny czas | 1.44s | 5.27s | 11.74s | 19.77s | 20.5s |
| Maksymalny czas | 20.9s | 198.25s | 193.78s | 216.01s | 20.5s |
| Średni  czas | 7.01s | 46.62s | 65s | 83.05s | 20.5s |
| Minimalna ilość generacji | 6 | 30 | 63 | 175 | 643 |
| Maksymalna ilość generacji | 84 | 971 | 996 | 945 | 643 |
| Średnia ilość generacji | 29 | 236 | 395 | 454 | 643 |
| Minimalne podobieństwo | 100% | 50% | 20% | 38.89% | 34.69% |
| Maksymalne podobieństwo | 100% | 100% | 100% | 100% | 100% |
| Średnie podobieństwo | 100% | 88.44% | 86.56% | 89% | 70.82% |

## Badanie wpływu ilości kolorów na czas rozwiązywania monogramu

Test wpływu ilości kolorów na czas rozwiązywania monogramu. Ustalona jest stały rozmiar monogramu 5 na 5 oraz przeprowadzone są próby dla „łatwego” oraz „trudnego” obrazu. Wykonywane jest 100 prób. Metoda fitness ocenia jedynie kolumnę lub wiersz które są identyczne z oryginalnym rozwiązaniem.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | 2 kolory | 3 kolory | 4 kolory | 5 kolorów | 6 kolorów |
| Łatwy | Procent poprawnych rozwiązań | 90% | 73% | 71% | 76% | 67% |
| Minimalny czas | 8.19s | 2.44s | 15.8s | 9.07s | 17.53% |
| Maksymalny czas | 196.87s | 165.31s | 200.57s | 212.72s | 212.7s |
| Średni  czas | 60.34s | 35.22s | 67.46s | 84.03s | 85.44s |
| Minimalna ilość generacji | 38 | 12 | 78 | 71 | 92 |
| Maksymalna ilość generacji | 956 | 945 | 952 | 962 | 982 |
| Średnia ilość generacji | 287 | 177 | 370 | 458 | 458 |
| Minimalne podobieństwo | 48% | 36% | 40% | 36% | 28% |
| Maksymalne podobieństwo | 100% | 100% | 100% | 100% | 100% |
| Średnie podobieństwo | 97.48% | 89.2% | 93.8% | 92.96% | 90.28% |
| Trudny | Procent poprawnych rozwiązań | 95% | 91% | 91% | 54% | 50% |
| Minimalny czas | 9.65s | 8.26s | 12.17s | 3.06s | 6.98s |
| Maksymalny czas | 201.39s | 169.98s | 196.81s | 181.41s | 203.42s |
| Średni  czas | 44.44s | 58.54s | 70.75s | 39.23s | 60.51s |
| Minimalna ilość generacji | 47 | 45 | 59 | 17 | 74 |
| Maksymalna ilość generacji | 914 | 822 | 978 | 880 | 964 |
| Średnia ilość generacji | 206 | 286 | 349 | 198 | 362 |
| Minimalne podobieństwo | 40% | 44% | 60% | 40% | 32% |
| Maksymalne podobieństwo | 100% | 100% | 100% | 100% | 100% |
| Średnie podobieństwo | 97.84% | 96.76% | 98.4% | 77.64% | 86.6% |

Test dla zmodyfikowanej metody fitness która przyjmuje ocenę podobieństwa rozwiązania z zakresu 0.0 do 1.0.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | 2 kolory | 3 kolory | 4 kolory | 5 kolorów | 6 kolorów |
| Łatwy | Procent poprawnych rozwiązań | 84% | 67% | 77% | 71% | 70% |
| Minimalny czas | 8.54s | 3.42s | 13.68s | 16.2s | 12.36s |
| Maksymalny czas | 184.2s | 185.08s | 237.18s | 202.67s | 206.07s |
| Średni  czas | 65.79s | 36.86s | 74.62s | 77.63s | 81.16s |
| Minimalna ilość generacji | 52 | 15 | 70 | 123 | 86 |
| Maksymalna ilość generacji | 942 | 899 | 966 | 922 | 993 |
| Średnia ilość generacji | 325 | 185 | 398 | 426 | 451 |
| Minimalne podobieństwo | 60% | 32% | 16% | 36% | 36% |
| Maksymalne podobieństwo | 100% | 100% | 100% | 100% | 100% |
| Średnie podobieństwo | 96.8% | 84.88% | 93.28% | 91.28% | 92.96% |
| Trudny | Procent poprawnych rozwiązań | 97% | 95% | 93% | 59% | 50% |
| Minimalny czas | 7.56s | 15.21s | 12.81s | 2.25s | 7.6s |
| Maksymalny czas | 185.13s | 168s | 185.55s | 210.44s | 202.6s |
| Średni  czas | 42.4s | 52.81s | 65.76s | 44.24s | 66.35s |
| Minimalna ilość generacji | 37 | 70 | 63 | 11 | 77 |
| Maksymalna ilość generacji | 949 | 722 | 866 | 991 | 878 |
| Średnia ilość generacji | 197 | 256 | 317 | 225 | 364 |
| Minimalne podobieństwo | 60% | 52% | 76% | 32% | 36% |
| Maksymalne podobieństwo | 100% | 100% | 100% | 100% | 100% |
| Średnie podobieństwo | 99.04% | 98.2% | 99.12% | 80.64% | 86.2% |

## Wnioski

Zwiększanie rozmiaru monogramu wpływa liniowo na średni czas rozwiązywania. Zmodyfikowana funkcja fitness redukuje czas rozwiązywania większych monogramów jednak wydłuża czas rozwiązania mniejszych. Wykres dla tego przypadku bardziej przypomina funkcję logarytmiczną. Jednocześnie nie wypływa znacząco na ilość rozwiązanych monogramów.

Zwiększenie ilości kolorów monogramu nie wpływa znacząco na czas rozwiązania monogramu. Zauważyć można że minimalny a ilość generacji wymaganych do rozwiązania monogramu znacząco rośnie. Zmodyfikowanie metody fitness nie niezauważalnie wpływa na średnie wyniki.