Rozwiązywanie łamigłówki Kuromasu algorytmem genetycznym

Sebastian Rychert

April 2022

Spis treści

1 Temat pracy

Niniejsza praca skupia się na przedstawieniu sposobu rozwiązania zagadki Kuromasu za pomocą algorytmy genetycznego oraz porównaniu wyników w zależności od stopnia skomplikowania łamigłówki. Najważniejsze kryteria to skuteczność algorytmu oraz czas potrzebny na znalezienie rozwiązania. Praca zawiera wstawki kodu źródłowego z języka python w wersji 3.8. Wykorzystano bibliotekę PyGAD.

2 Opis gry

2.1 Zasady

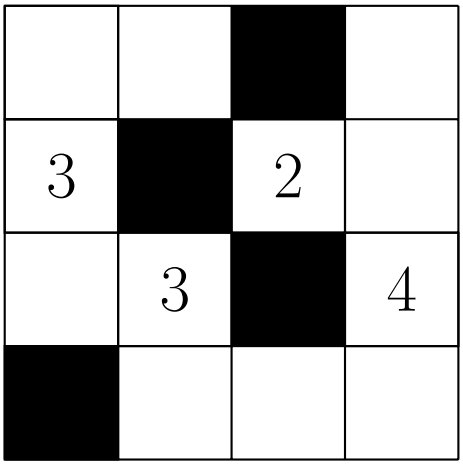
W Kuromasu gra się na prostokątnej siatce. Na początku wszystkie komórki są białe, a niektóre mają w sobie liczby. Celem gry jest określenie koloru każdej komórki (biała lub czarna).

Poniższe zasady określają sposób wyboru koloru dla każdej komórki:

- Każda liczba na planszy reprezentuje liczbę białych komórek, które można zobaczyć z tej komórki, łącznie z nią. Komórkę można zobaczyć z innej komórki, jeśli obie komórki znajdują się w tym samym wierszu lub kolumnie i nie ma między nimi czarnych komórek w tym wierszu lub kolumnie.
- Ponumerowane komórki nie mogą być czarne.
- Żadne dwie czarne komórki nie mogą przylegać do siebie w pionie lub poziomie.
- Wszystkie białe komórki muszą być połączone poziomo lub pionowo.

2.2 Przykład rozwiązania

3		2	
	3		4



3 Algorytm

3.1 Reprezentacja planszy

Plansza w programie jest reprezentowana jako dwuwymiarowa tablica numpy. Dane potrzebne do jej utworzenia czytane są z pliku "puzzles.json" pod konkrentynm kluczem np. "0". Puste pola są zerami.

```
# load puzzles from file
puzzles = {}
with open("puzzles.json", "r") as f:
    puzzles = json.load(f)

# make a board for puzzle - args.p is puzzle id flag e.g. "0"
board = np.array(puzzles[args.p])
```

3.2 Parametry

Aby uruchomić algorytm genetyczny musimy zdefiniowac szereg parametrów.

Pierwszym z nich jest ***gene_space***, który określa możliwe geny do wygenerowania w rozwązaniu. Nasz algorytm będzie generował liczby binarne, to znaczy 0 lub 1.

Następnie ustawiamy ***num_genes***, odpowiadający za długość chromosomu. Wartość taka jak liczba pustych pól w łamigłówce.

Kolejnym ważnym parameterm jest ***sol_per_pop***, odpowiadający za wielkość populacji, czyli inaczej liczbę chromosomów w danej generacji.

Trzy kolejne parametry są ustawiane jako mały procent wielkości populacji.
num_generations - Liczba generacji
num_parents_mating - Ile wybrać rodziców do rozmnażania
keep_parents - Ile rodziców zachować

Określimy także punkt zatrzymania algorytmu w przypadku braku postępów przez znaczną część pokoleń - ***saturate*** - oraz procent szansy na mutacje ***mutation_percent_genes***

```
gene_space = [1, 0]

# calc gene length from board
num_genes = 0
for i in board.flatten():
    if i == 0:
        num_genes += 1

# population size
sol_per_pop = board.shape[0] * board.shape[1] * 10
num_generations = int(2 * sol_per_pop)
num_parents_mating = int(0.25 * sol_per_pop)
keep_parents = int(0.01 * sol_per_pop)
s = int(0.02 * num_generations)
saturate = "saturate_{s}".format(s=15 if s < 15 else s)

# sss = steady, rws=roulette, rank = rankingowa, tournament = turniejowa
parent_selection_type = "sss"
crossover_type = "single_point"
mutation_type = "random"
# small boards - smaller mutation percent and bigger for bigger boards
mutation_percent_genes = 120 / num_genes if num_genes <= 50 else 100 * (2 / num_genes)
```

3.3 Funkcja przystosowania – (ang. fitness function)

Jest to główny element algorytmu który musimy zdefiniować. Na podstawie wyniku funkcji ewaluujemy poprawność rozwiązania dla każdego chromosomu. Celem algorytmu jest osiągnięcie wartości 0. Punkty ujemne przydzielamy w liczbie:

- Wynik z absolutnej różnicy wartości każdej ponumerowanej komórki z liczbą komórek, które widzi w pionie i poziomie.
- 2 punkty za każdą parę czarnych komórek (para - 2 komórki sąsiadujące w pionie lub poziomie)
- Potęga o podstawie równej 2 oraz wykładniku będącym liczbą wysp, czyli odseparownaych od reszty skupisk białych komórek, pomniejszony o 1. Wynik potęgowania zmniejszamy o 1 ($2^1 - 1 = 0$)

```
def fitness_func(solution, solution_idx):
    # board for solution
    board_s = makeBoardWithSoution(solution)
    points = 0
    # deep first search object
    dfs = DFS()
    # find starting point for dfs
    firstA = np.argwhere(board_s != 1)[0]
    first = (firstA[0], firstA[1])

    # calculate points
    for idx, x in np.ndenumerate(board_s):
        points -= whiteCellsPoints(x, idx, board_s)
        points -= blackCellsPoints(x, idx, board_s)

    # check if there is only one island else give punishment points
    islands = dfs.getNumberOfIslands(board_s, first)
    points -= 2 ** (islands - 1) - 1

return points
```

4 Wyniki

4.1 Sposób pomiaru

Algorytm został uruchomiony na 10 różnych łamigłówkach o rosnącym stopniu trudności; zaczynając na planszy o wymiarach 4x4, a kończąc na 11x11. Każda plansza była poddana próbie rozwiązania 10 razy. Dla każdej planszy zapisany został procent sukcesu, średni czas rozwiązania oraz średni numer generacji rozwiązania. Próby zakończone porażką, to znaczy wartością fitness różną od 0, zostały pominięte.

Aby zwiększyć predkość przeprowadzania testów została użyta technika przetwarzania wieloprocesorowego.

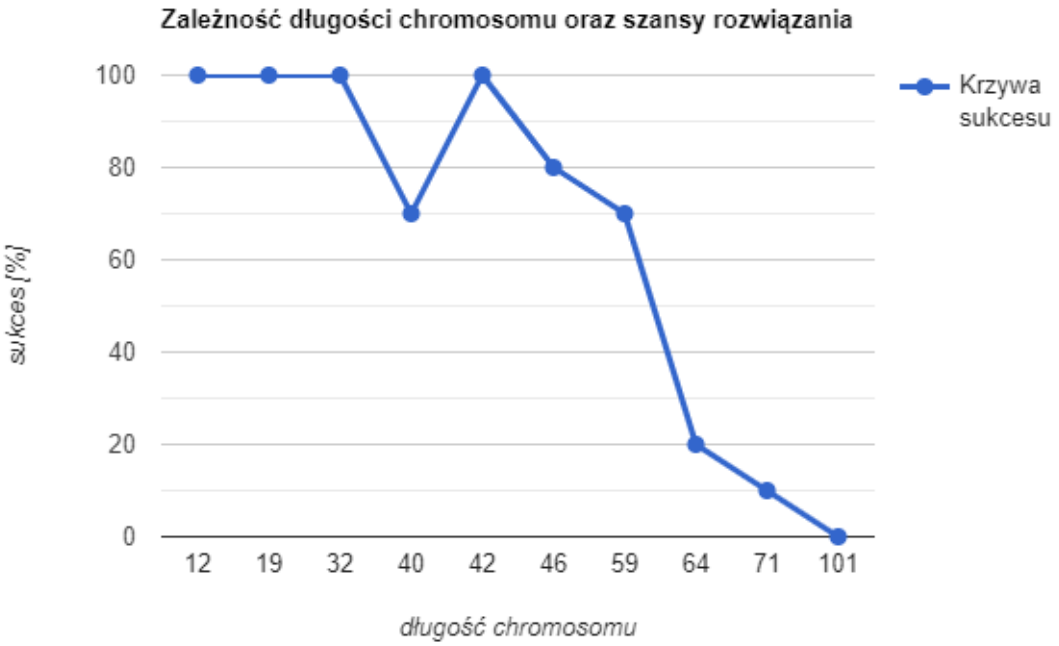
```
results = []
with concurrent.futures.ProcessPoolExecutor() as executor:
    # make list of len = runs
    l = list(range(0, runs))
    # run the algorithm for the number of processes equal to "runs"
    pool = executor.map(runGA, l)
    for res in pool:
        results.append(res)
```

4.2 Tabela

Tabela 1: Wyniki testów

puzzle	rozmiar	długość chromosomu	procent sukcesu	średni czas sukcesu [s]	średnia generacja sukcesu
0	4x4	12	100%	0.179	65
1	5x5	19	100%	1.004	111
2	7x6	32	100%	3.535	191
3	7x7	40	70%	5.218	184
4	8x7	42	100%	6.712	171
5	8x7	46	80%	9.900	311
6	9x9	59	70%	53.922	307
7	9x9	64	20%	69.261	609
8	9x9	71	10%	50.431	8
9	11x11	101	0%	-	-

4.3 Wykresy


Zależność długości chromosomu oraz szansy rozwiązania
100
80
60
40
20
0
sukces [%]
12
19
32
40
42
46
59
64
71
101
długość chromosomu
Krzywa sukcesu


Zależność długości chromosomu oraz czasu rozwiązania
80
60
40
20
0
czas [s]
12
19
32
40
42
46
59
64
71
długość chromosomu
Krzywa czasu

4.4 Podsumowanie

Przeprowadzone testy wskazują iż algorytm znajduje rozwiązanie w mniejszych oraz średnich planszach, ale nie radzi sobie już z większymi planszami. Punktem diametralnego spadku procentu sukcesu wydaje się być długość chromosomu równa 60. Algorytm powyżej tej długości o wiele częściej wpada w minima lokalne, które nie pasują ostatecznie do prawidłowego rozwiązania.
Wyniki planszy o id "8" sugerują, iż jedynie początkowa "szczęśliwa" kombinacja jest wstanie doprowadzić do rozwiązania na większej planszy - świadczy o tym bardzo mała liczba generacji równa 8.

Literatura

[1] Wikipedia strona Kuromasu `https://en.wikipedia.org/wiki/Kuromasu`

[2] PyGad Dokumentacja `https://pygad.readthedocs.io/en/latest/`

[3] Katalog łamigłówek `https://www.math.edu.pl/kuromasu`