WSI LAB 2
Algorytmy genetyczne i ewolucyjne

Będkowski Patryk, 310603

I. WSTĘP

Niniejszy raport przedstawia ćwiczenie polegające na implementacji algorytmu genetycznego do rozwiązania następującego problemu:

Układamy grafik dla studentów na zdalnym nauczaniu. Mamy n=25 przedmiotów (wierzchołki w grafie) i uczelnia najchętniej zrobiłaby je tego samego dnia. Niestety, jeżeli jakiś student zapisany jest na dwa przedmioty, to nie mogą one odbyć się o tej samej godzinie (student reprezentuje krawędź w grafie). Szukamy sposobu jak zorganizować wszystkie zajęcia w jak najmniejszej liczbie okien czasowych - innymi słowy jak pokolorować wierzchołki w grafie, by te same kolory nie były połączone krawędzią. (graph coloring problem)

II. URUCHOMIENIE

Do uruchomienia skryptu potrzebne są następujące narzędzia:

- Python w wersji 3.8.8
- Moduły:
    - networkx 2.5
    - pandas 1.2.4
    - numpy 1.20.1

Aby uruchomić skrypt należy użyć polecenia:

```
python main.py
```

III. PRZYGOTOWANE PLIKI

W załączniku raportu przesyłam plik .zip wraz z implementacją wymaganego algorytmu. Znajduje się on wraz z funkcjami pomocniczymi w pliku `main.py`. Zawiera on implementację klasycznego algorytmu ewolucyjnego w formie klasy. Do tworzenia kolejnych populacji wykorzystuje się selekcję turniejową (turnieje dwuosobowe), mutację oraz sukcesję generacyjną. Owa klasa została skonstruowana tak, aby mogła przyjmować dowolne parametry.
Każde uruchomienie skryptu powoduje utworzenie raportu w formie pliku .txt wraz z wynikami jego działania oraz parametrami uruchomienia. Dodatkowo skrypt tworzy wykresy zapisywane w katalogu zawierającym owy plik.

IV. ROZWIĄZANIE

Zadanie polega na minimalizacji kolorów wierzchołków dla różnych grafów nieskierowanych. W plikach źródłowych znajdują się funkcje generujące owe grafy w sposób losowy. W kodzie źródłowym zostały zdefiniowane następujące typy grafów, które można podać jako argument klasy:

Typ grafu	Nazwa reprezentująca dany typ w skrypcie
pełny	complete
dwudzielny	bipartite
losowy (60% grafu pełnego)	random
duże skupiska grup (15% wierzchołków jest połączona z 50% reszty wierzchołków)	random groups

(tabela 1.1) Typy grafów

Jeżeli do programu nie zostanie podany typ grafu, zostanie załadowany domyślny graf typu: *duże skupiska grup*.

V. ROZWIĄZANIE

Zaimplementowany algorytm pozwala znaleźć minimalną ilość kolorów (przy ograniczonym budżecie) potrzebną do pokolorowania grafu w taki sposób, żeby dwa połączone wierzchołki nie były tego samego koloru (graph coloring problem).

Wyniki algorytmu genetycznego zależą głównie od następujących parametrów:

- maksymalna ilość iteracji
- wielkość populacji
- prawdopodobieństwo mutacji osobnika
- prawdopodobieństwo mutacji genu

Parametry, które mają pośredni wpływ na wyniki algorytmu to:

- typ użytego grafu (tabela 1.1)
- „losowość" zbudowanego grafu
- maksymalna dopuszczalna ilość iteracji bez poprawy
- ilość wierzchołków rozwiązywanego grafu

Ponieważ problem dotyczył grafu o ilości wierzchołków równej 25, nie będę rozważał jak różna ilość wierzchołków wpływa na wyniki.

Osobnik jest przedstawiony jako sekwencja liczb całkowitych. Ilość tych liczb równa jest liczbie wierzchołków. Pojedyncza liczba sekwencji reprezentuje kolor danego wierzchołka w taki sposób, że pierwsza liczba sekwencji odpowiada kolorowi pierwszego wierzchołka, druga odpowiada kolorowi drugiego wierzchołka, itp.

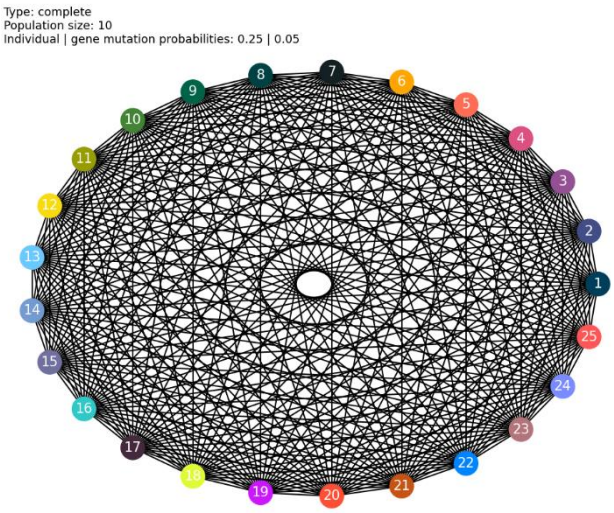
Pula kolorów możliwych do pokolorowania wierzchołków stanowi liczby całkowite od 1 do N (gdzie N to ilość wierzchołków grafu).

Przykładowego osobnika możemy przedstawić jako sekwencję: *15542*.
Wtedy wierzchołek numer *2* oraz *3* są tego samego koloru równego *5*.

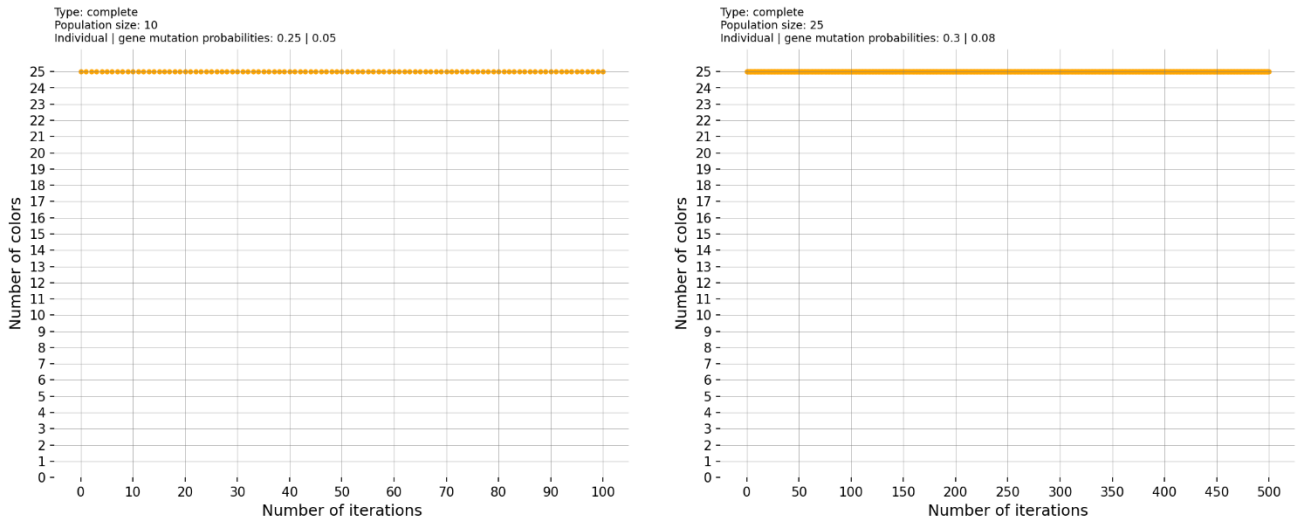
Przykład niedozwolonego osobnika możemy przedstawić jako sekwencję: *12632*.
Wierzchołek numer *3* posiada kolor *6*, który nie należy do zbioru kolorów dostępnych dla owego grafu, ponieważ jego ilość wierzchołków wynosi pięć.

1. Graf pełny

Graf pełny to taki, w którym dla każdej pary węzłów istnieje krawędź je łącząca. Nie jest to zaskoczeniem, że dla danego grafu niezależnie od wprowadzonych parametrów, algorytm nie zminimalizuje ilości kolorów potrzebnych do pokolorowania jego węzłów.



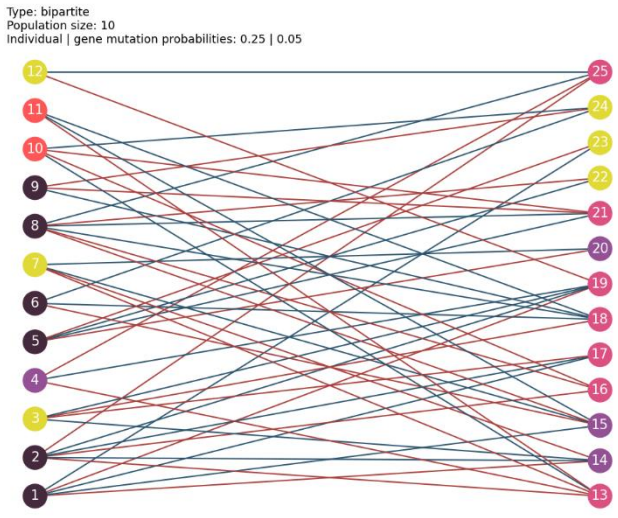
(wykres 1.1) Reprezentacja grafu pełnego



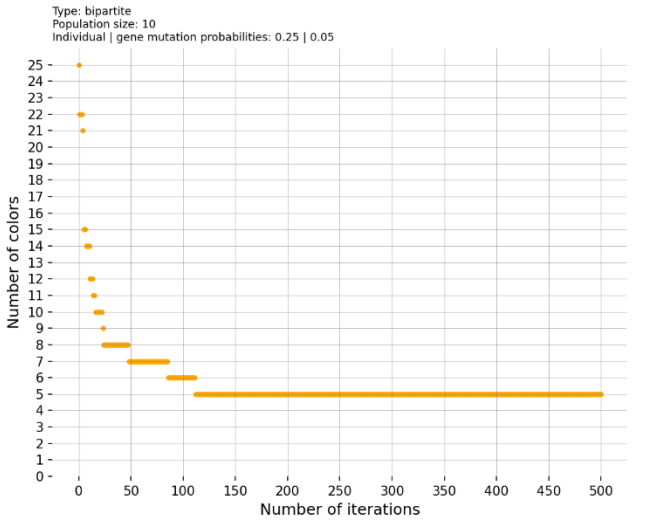
(wykres 1.2 – 1.3) Zależność kolejnych iteracji algorytmu i ilości zminimalizowanych kolorów dla różnych parametrów

2. Graf dwudzielny

Wierzchołki grafu dwudzielnego można podzielić na dwa rozłączne zbiory. Wierzchołki należące do jednego zbioru mogą się łączyć krawędziami wyłącznie z drugiego zbioru i na odwrót. Wiadomo, że wierzchołki grafu tego typu możemy przedstawić za pomocą minimalnie dwóch różnych kolorów aby rozwiązać nasz problem.
Algorytm dzieli poszczególne wierzchołki na rozłączne zbiory U i V w sposób losowy.



(wykres 2.1) Graf dwudzielny



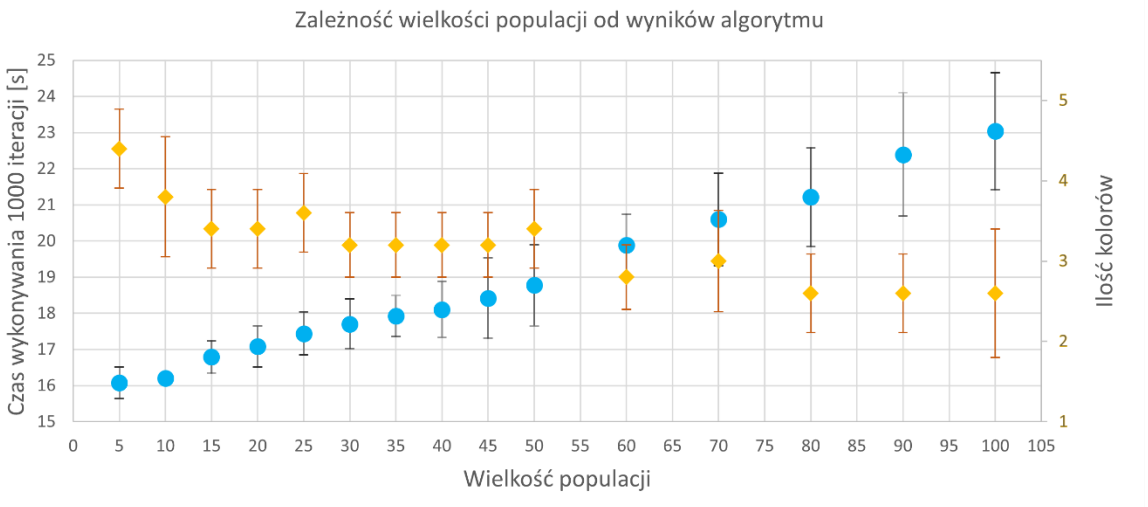
(wykres 2.2) Zależność ilości zminimalizowanych kolorów w kolejnych iteracjach

W poniższych testach algorytm został uruchomiony z parametrami [1] (test ma na celu badanie zmiany jednego z parametrów):

- maksymalna ilość iteracji: 1000
- wielkość populacji: 20
- prawdopodobieństwo mutacji osobnika: 0.3
- prawdopodobieństwo mutacji genu: 0.05
- typ grafu: bipartite

Aby zbadać losowość algorytmu, każdy z poniższych testów został uruchomiony 25 razy. Poniższe wykresy przedstawiają średnie uzyskane wartości wraz z odchyleniem standardowym.

Zbadano wpływ liczby osobników populacji na jakość uzyskanych rozwiązań. Algorytm uruchomiono z powyższymi parametrami [1]:



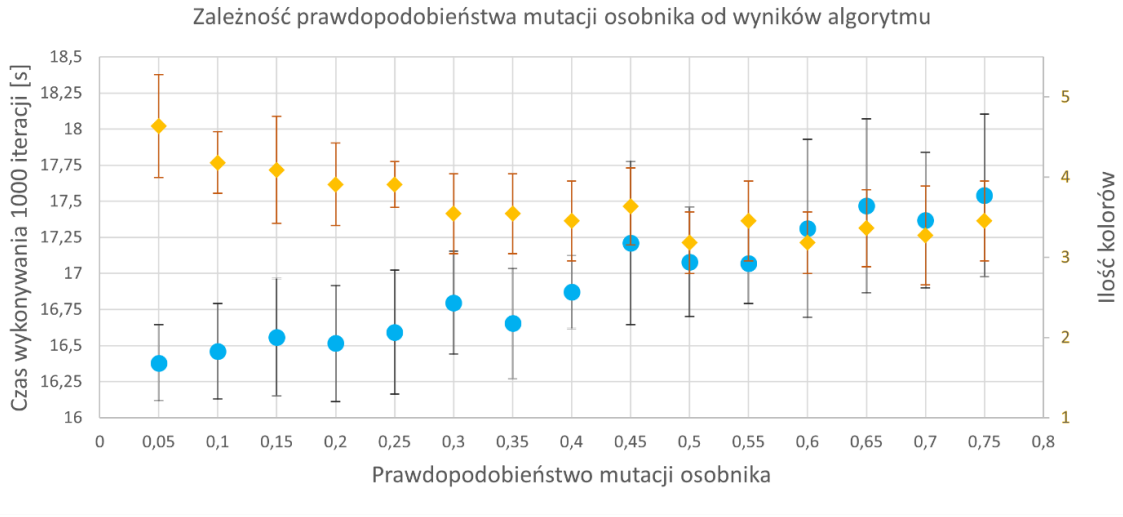
(wykres 2.3) Zależność wielkości populacji od wyników algorytmu

Wielkość populacji	Czas wykonywania		Zminimalizowana ilość kolorów	
	Minimalny	Maksymalny	Minimalna	Maksymalna
5	15,52	16,82	4	5
10	16,09	16,31	3	5
15	16,43	17,61	3	4
20	16,63	18,08	3	4
25	16,86	18,39	3	4
30	17,09	18,81	3	4
35	17,36	18,86	3	4
40	17,50	19,57	3	4
45	17,33	20,57	3	4
50	18,07	21,01	3	4
60	18,91	21,42	2	3
70	19,48	22,88	2	4
80	20,23	23,90	2	3
90	21,01	25,56	2	3
100	21,74	26,11	2	4

(tabela 2.1) Minimalne, maksymalne odczyty pomiarów

Czas wykonywania tysiąca iteracji rośnie wraz ze wzrostem populacji. Jednocześnie im większa populacja tym algorytm lepiej minimalizuje ilość kolorów w ograniczonej liczbie iteracji. Dodatkowo większe populacje skutkują większą dysproporcją czasów wykonywania algorytmu.

Aby zbadać wpływ zmiany parametrów, algorytm został uruchomiony z następującymi parametrami [1], do parametrów poddawanych modyfikacji należą *prawdopodobieństwo mutacji osobnika* oraz *prawdopodobieństwo mutacji genu*.

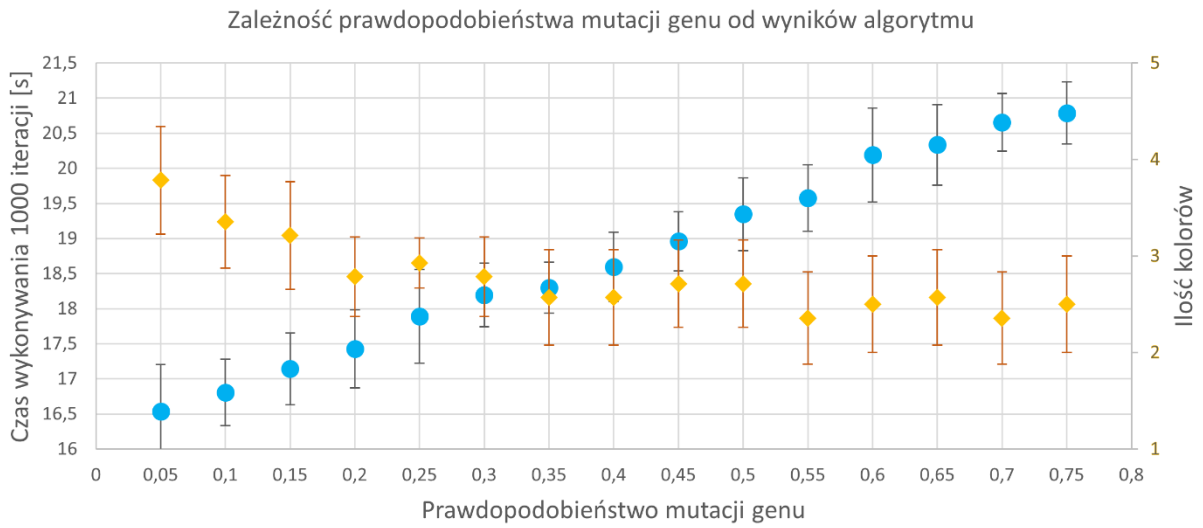


(wykres 2.4) Zależność prawdopodobieństwa mutacji osobnika od wyników algorytmu dla parametrów [1]

Prawdopodobieństwo mutacji osobnika	Czas wykonywania		Zminimalizowana ilość kolorów	
	Minimalny	Maksymalny	Minimalna	Maksymalna
0,05	15,83	16,80	4	6
0,10	15,73	16,87	4	5
0,15	15,80	17,32	3	5
0,20	15,86	17,31	3	5
0,25	15,78	17,19	3	4
0,30	15,98	17,21	3	4
0,35	15,95	17,18	3	4
0,40	16,49	17,35	3	4
0,45	16,36	18,11	3	4
0,50	16,70	18,12	3	4
0,55	16,62	17,47	3	4
0,60	16,60	19,07	3	4
0,65	16,66	18,79	3	4
0,70	16,79	18,56	2	4
0,75	17,00	18,93	3	4

(tabela 2.2) Minimalne, maksymalne odczyty pomiarów

Z powyższego wykresu wynika, iż im większe prawdopodobieństwo mutacji danego osobnika, tym algorytm potrzebuje więcej czasu aby znaleźć rozwiązanie w 1000 iteracji. Dodatkowo niższe wartości tego prawdopodobieństwa skutkują gorszym wynikiem (większa ilość zminimalizowanych kolorów) algorytmu.



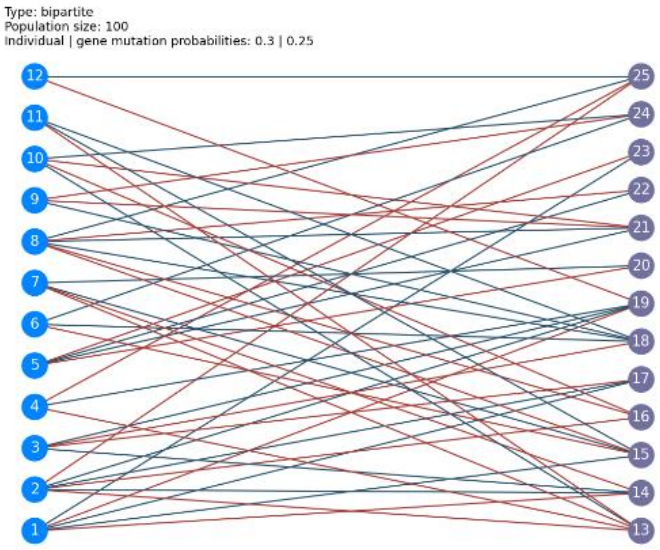
(wykres 2.5) Zależność prawdopodobieństwa mutacji genu od wyników algorytmu dla parametrów [1]

Prawdopodobieństwo mutacji genu	Czas wykonywania		Zminimalizowana ilość kolorów	
	Minimalny	Maksymalny	Minimalna	Maksymalna
0,05	15,93	17,87	3	5
0,10	16,23	17,63	3	4
0,15	16,58	18,18	2	4
0,20	16,85	18,82	2	3
0,25	17,11	19,34	2	3
0,30	17,47	18,89	2	3
0,35	17,65	19,00	2	3
0,40	17,43	19,58	2	3
0,45	18,43	20,22	2	3
0,50	18,64	20,72	2	3
0,55	18,98	20,79	2	3
0,60	19,38	21,60	2	3
0,65	19,77	21,77	2	3
0,70	20,17	21,68	2	3
0,75	20,37	21,76	2	3

(tabela 2.3) Minimalne, maksymalne odczyty pomiarów

Czas wykonywania algorytmu rośnie wraz ze wzrostem prawdopodobieństwa mutacji pojedynczego genu. Widzimy również, że wzrost wartości tego prawdopodobieństwa skutkuje lepszym minimalizowaniem ilości użytych kolorów.

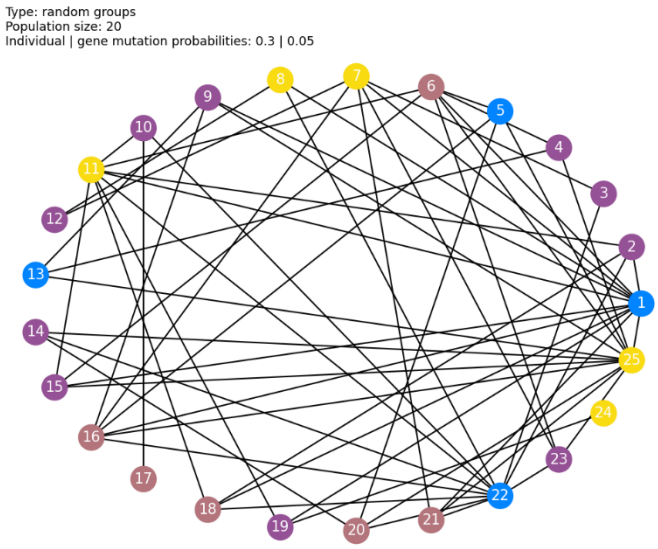
Poniższy wykres pokazuje wyniki algorytmu po 2000 iteracjach dla podanych parametrów:



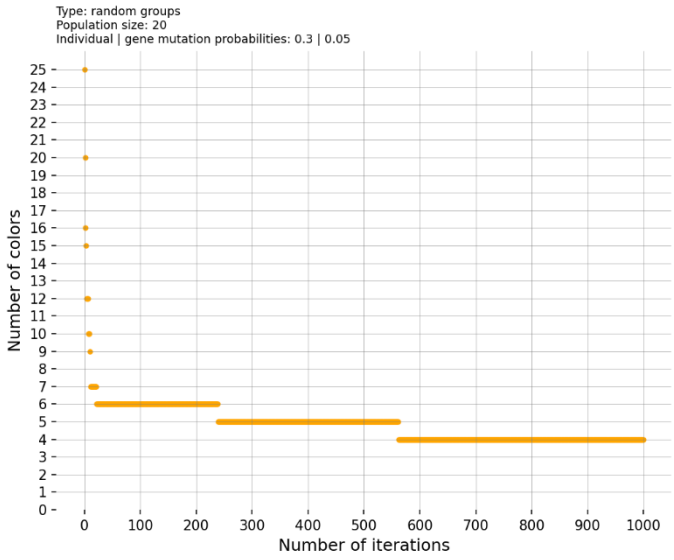
(wykres 2.6) W pełni zminimalizowany graf dwudzielny

3. Graf duże skupiska grup

Ten typ grafu opisuje połączenia między węzłami jako lokalne grupy dużego zagęszczenia połączeń. 15% wierzchołków zostaje wybrana i połączona w sposób losowy z 50% wierzchołków całego grafu. Przykładowy wynik algorytmu dla grafu tego typu:



(wykres 3.1) Graf przedstawiający skupiska dużych grup



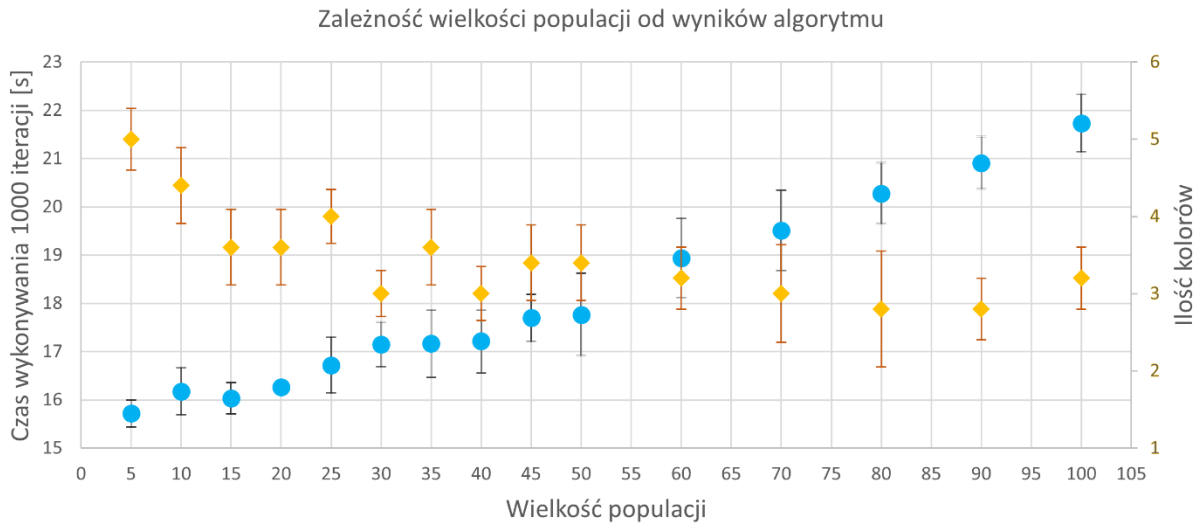
(wykres 3.2) Zależność ilości zminimalizowanych kolorów w kolejnych iteracjach

W poniższych testach algorytm został uruchomiony z parametrami [2] (test ma na celu badanie zmiany jednego z parametrów):

- maksymalna ilość iteracji: 1000
- wielkość populacji: 10
- prawdopodobieństwo mutacji osobnika: 0.25
- prawdopodobieństwo mutacji genu: 0.08
- typ grafu: random groups

Aby zbadać losowość algorytmu, każdy z poniższych testów został uruchomiony 25 razy. Poniższe wykresy przedstawiają średnie uzyskane wartości wraz z odchyleniem standardowym.

Zbadano wpływ liczby osobników populacji na jakość uzyskanych rozwiązań. Algorytm uruchomiono z powyższymi parametrami [2]:



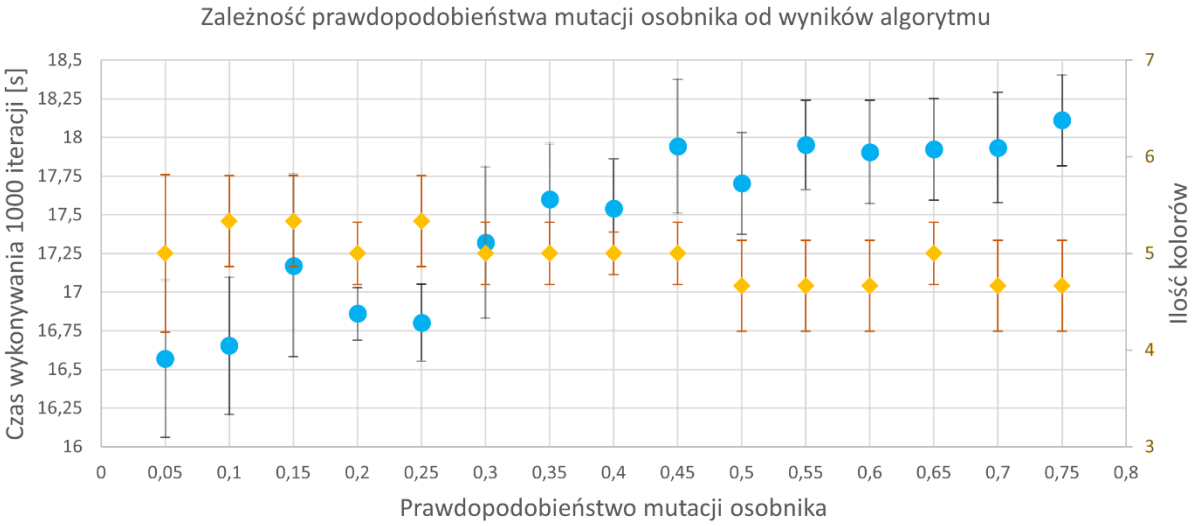
(wykres 3.3) Zależność wielkości populacji od wyników algorytmu

Wielkość populacji	Czas wykonywania		Zminimalizowana ilość kolorów	
	Minimalny	Maksymalny	Minimalna	Maksymalna
5	15,24	16,03	5	6
10	15,80	17,12	4	5
15	15,64	16,57	3	4
20	16,13	16,37	3	4
25	15,73	17,44	3	4
30	16,35	17,68	3	4
35	16,24	18,36	3	4
40	16,47	18,34	3	4
45	16,85	18,21	3	4
50	16,36	18,81	3	4
60	17,49	19,98	3	4
70	18,48	20,51	2	4
80	19,37	21,35	2	4
90	19,90	21,40	2	3
100	20,64	22,33	3	4

(tabela 3.1) Minimalne, maksymalne odczyty pomiarów

Czas wykonywania tysiąca iteracji rośnie wraz ze wzrostem populacji. Możemy również zauważyć, że większe populacje skutkują większą dysproporcją uzyskanych kolorów po minimalizacji.

Aby zbadać wpływ zmiany parametrów, algorytm został uruchomiony z następującymi parametrami [2], do parametrów poddawanych modyfikacji należą *prawdopodobieństwo mutacji osobnika* oraz *prawdopodobieństwo mutacji genu*.

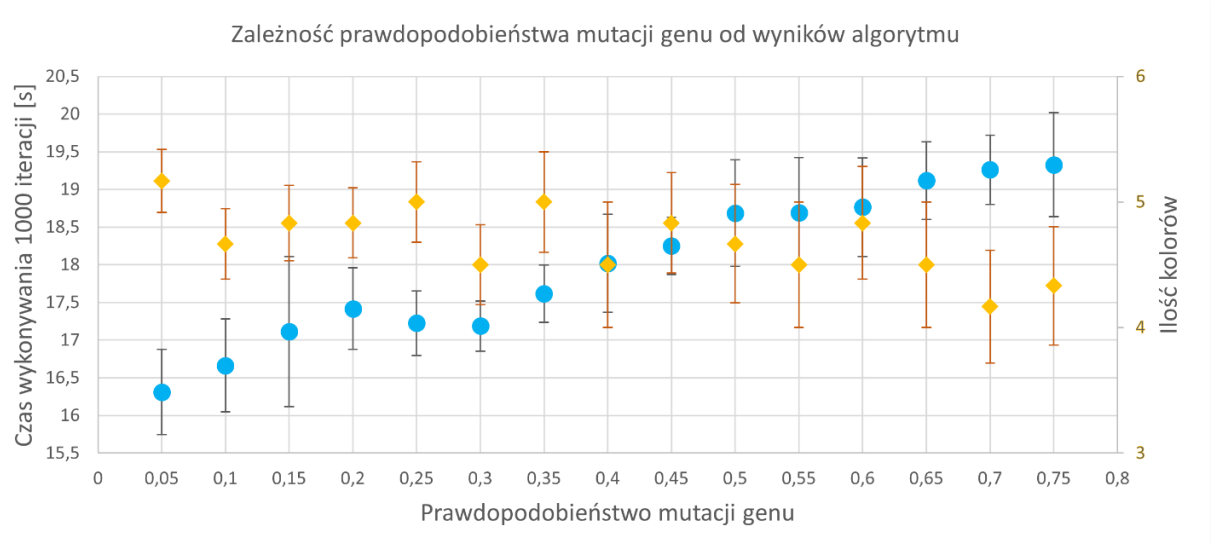


(wykres 3.4) Zależność prawdopodobieństwa mutacji osobnika od wyników algorytmu dla parametrów [2]

Prawdopodobieństwo mutacji osobnika	Czas wykonywania		Zminimalizowana ilość kolorów	
	Minimalny	Maksymalny	Minimalna	Maksymalna
0,05	15,85	16,95	4	6
0,10	16,09	17,18	5	6
0,15	16,50	17,94	5	6
0,20	16,62	16,99	5	5
0,25	16,57	17,15	5	6
0,30	16,65	17,81	5	5
0,35	17,20	18,08	5	5
0,40	17,10	17,84	5	5
0,45	17,58	18,55	5	5
0,50	17,25	18,02	4	5
0,55	17,58	18,28	4	5
0,60	17,63	18,37	4	5
0,65	17,66	18,39	5	5
0,70	17,43	18,21	4	5
0,75	17,78	18,50	4	5

(tabela 3.2) Minimalne, maksymalne odczyty pomiarów

Z powyższego wykresu wynika, że niższe wartości prawdopodobieństwa mutacji osobnika skutkują lepszymi wynikami czasowymi, lecz dla szczególnie niskich wartości (0,05, 0,10 oraz 0,15), czas wykonania osiąga bardziej różnorodne wartości dla różnych uruchomień algorytmu. Dodatkowo w przeciwieństwie do wielkości populacji, zmiana parametru prawdopodobieństwa mutacji wpływa na bardziej różnorodne wartości zminimalizowanych kolorów dla różnych uruchomień algorytmu.



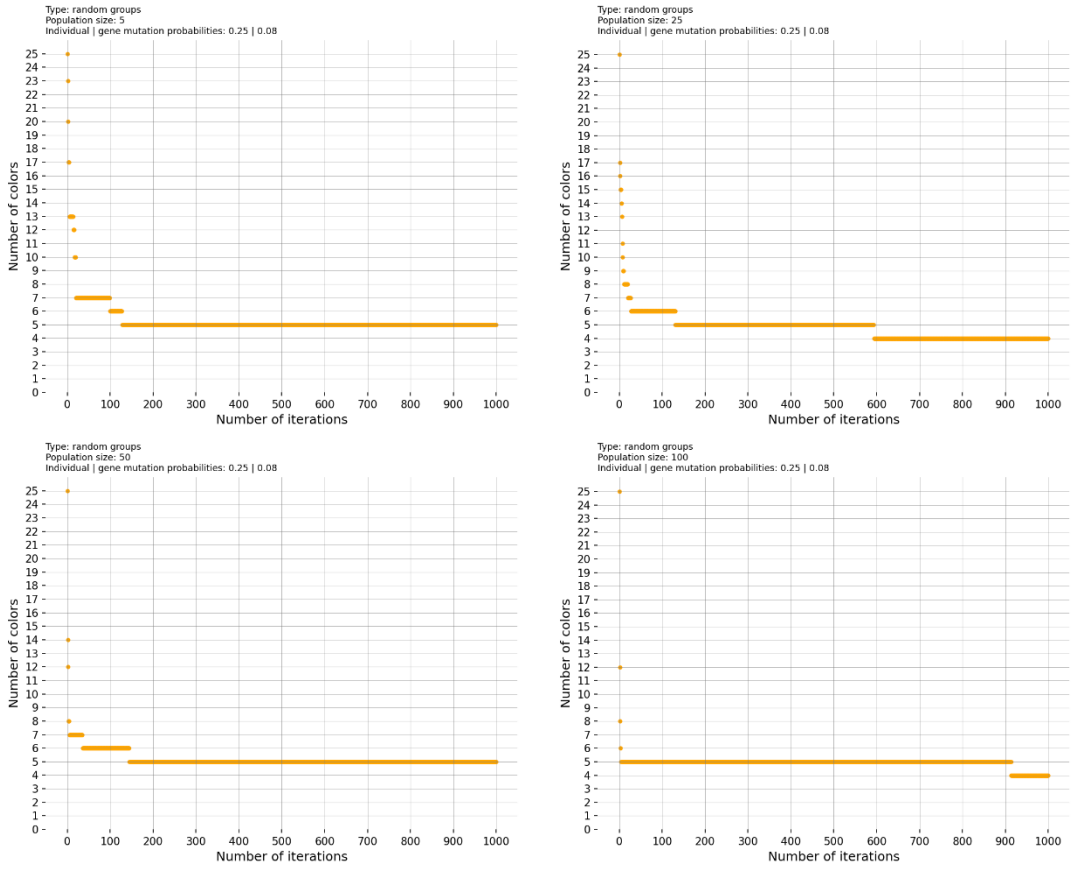
(wykres 3.5) Zależność prawdopodobieństwa mutacji genu od wyników algorytmu dla parametrów [2]

Prawdopodobieństwo mutacji genu	Czas wykonywania		Zminimalizowana ilość kolorów	
	Minimalny	Maksymalny	Minimalna	Maksymalna
0,05	15,32	17,08	5	6
0,10	15,81	17,41	4	5
0,15	15,84	18,33	4	5
0,20	16,81	18,33	4	5
0,25	16,74	18,05	5	5
0,30	16,71	17,49	4	5
0,35	17,07	18,21	5	5
0,40	16,71	18,73	4	5
0,45	17,84	18,75	4	5
0,50	17,90	19,55	4	5
0,55	17,87	19,63	4	5
0,60	18,04	19,65	4	5
0,65	18,46	20,13	4	5
0,70	18,48	20,06	4	5
0,75	18,38	20,52	4	5

(tabela 3.3) Minimalne, maksymalne odczyty pomiarów

Czas wykonywania algorytmu rośnie wraz ze wzrostem prawdopodobieństwa mutacji pojedynczego genu. Widzimy również, że wzrost wartości tego prawdopodobieństwa skutkuje lepszym minimalizowaniem ilości użytych kolorów. Jednakże wraz ze wzrostem wartości tego parametru, rośnie również odchylenie standardowe, mówiące o tym, że w różnych uruchomieniach otrzymujemy bardziej zróżnicowane wyniki minimalizacji ilości kolorów w grafie.

Sprawdźmy również jak wielkość populacji wpływa na ilość zminimalizowanych kolorów w kolejnych iteracjach. Algorytm został uruchomiony dla czterech różnych wartości wielkości populacji, reszta parametrów została podana powyżej [2]:



(wykres 3.6) Zależność ilości zminimalizowanych kolorów w kolejnych iteracjach dla różnych wielkości populacji

Powyższe wykresy wskazują na to, ze duża populacja może nam przyspieszyć minimalizację ilości kolorów, lecz potrzebuje ona więcej iteracji na to żeby znaleźć bardziej optymalne rozwiązanie. Patrząc na populację o rozmiarze 25 i 100 widzimy,

że ta pierwsza początkowo potrzebowała więcej iteracji na minimalizację ilości kolorów, lecz finalnie w przeciwieństwie do populacji o rozmiarze 100, potrzebowała mniej iteracji aby znaleźć minimalną ilość kolorów równą 4.

4. Generowany raport

Pojedyncze uruchomienie algorytmu powoduje wygenerowanie raportu zadania minimalizacji kolorów wierzchołków zapisanego w postaci pliku .txt wraz z użytymi parametrami oraz datą wykonania. Przykładowy raport został przedstawiony poniżej:

```
Algorithm found the best individual in 1000 iterations.
Execution time 17.37 [s]
At the beginning graph had: 25 unique colors
Graph was minimized to only: 5 unique colors
Best individual is best represented by the sequence:
[5, 3, 25, 5, 5, 3, 19, 3, 25, 5, 25, 25, 19, 5, 19, 3, 19, 23, 19, 25, 25, 19, 5, 25, 23]
Vertices of generated graph:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
Edges of generated graph:
[(2, 18), (3, 7), (4, 13), (5, 16), (7, 21), (8, 12), (9, 16), (10, 11), (11, 2), (12, 7),
(13, 9), (14, 20), (15, 11), (16, 7), (17, 10), (18, 11), (19, 11), (20, 5), (21, 1), (23, 7),
(24, 19), (22, 20), (22, 16), (22, 11), (22, 2), (22, 18), (22, 21), (22, 8), (22, 23), (22, 3),
(22, 14), (22, 10), (22, 10), (6, 5), (6, 11), (6, 5), (6, 1), (6, 11), (6, 15), (6, 25), (6, 15),
(6, 11), (6, 4), (6, 23), (6, 23), (25, 21), (25, 13), (25, 20), (25, 9), (25, 13), (25, 14), (25, 5),
(25, 15), (25, 23), (25, 1), (25, 16), (25, 4), (1, 11), (1, 19), (1, 9), (1, 16), (1, 21), (1, 15), (1, 7),
(1, 18), (1, 18), (1, 19), (1, 2), (1, 8)]
```

(wykres 4.1) Raport wygenerowany przez algorytm

Raport przedstawia:

- ilość iteracji wykonanej przez algorytm
- czas wykonania
- początkową ilość kolorów
- zminimalizowaną ilość kolorów
- najlepszego osobnika znalezionego przez algorytm
- ponumerowane wierzchołki grafu
- krawędzie grafu w postaci pary dwóch połączonych wierzchołków

5. Podsumowanie

Zaimplementowany przeze mnie algorytm z powodzeniem radzi sobie w rozwiązywaniu problemu kolorowania grafu. Algorytm ten możemy wykorzystać do rozwiązania wielu problemów, nie tylko do poruszanego problemu ułożenia harmonogramu egzaminów.
Z powyższej analizy wynika, że typ grafu oraz różne parametry wpływają na jakoś uzyskiwanych wyników. Jednakże generowane wykresy oraz raport, skutecznie pomagają znaleźć odpowiednie parametry do naszych potrzeb i ograniczonych zasobów obliczeniowych.