Wstęp do sztucznej inteligencji
Sprawozdanie 2

Jan Kwiatkowski 299374

Marzec 2023

1 Treść zadania

Tematem drugich ćwiczeń są algorytmy genetyczne i ewolucyjne. Zadaniem jest zaimplementować klasyczny algorytm ewolucyjny bez krzyżowania, z selekcją turniejową i sukcesją generacyjną. W raporcie należało wskazać jak zmiana liczby osobników w populacji wpływa na jakość uzyskanych rozwiązań przy ograniczonym budżecie. Opisano także zachowanie algorytmu dla różnych rodzajów danych wejściowych oraz wpływ zmiany parametrów. Przykładowe zbiory danych i ich generatory skonstruowano samemu na potrzebę zadania.

1.1 Problem

Z powodu podwyżek cen prądu w Farszawie, zmniejszono nakłady na oświetlenie ulic tego wspaniałego miasta. W mieście znajduje się n=25 placów (wierzchołki w grafie), z których można dotrzeć do sąsiadujących placów (istnieje krawędź w grafie). Latarnia stojąca na placu jest w stanie pokryć wszystkie krawędzie biegnące od tego placu do wszystkich jego sąsiadów. Które lampy powinniśmy włączyć na noc by uzyskać jak największe pokrycie wszystkich ulic (vertex cover problem). Sugerowane grafy: graf pełny, graf dwudzielny, graf losowy (graf pełny z usuniętymi 50-70

Do rozwiązania powyższego, grafowego problemu NP-zupełnego zaimplementowano:

- algorytm generacyjny bez krzyżowania, z selekcją turniejową i sukcesją generacyjną.

2 Vertex cover problem - założenia

Problem pokrycia wierzchołków jest problemem, polegającym na znalezieniu najmniejszego zestawu wierzchołków w grafie, który obejmuje wszystkie krawędzie grafu.

Biorąc pod uwagę graf nieskierowany $G = (V, E)$, pokrycie wierzchołków jest podzbiorem wierzchołków $C \subseteq V$ takim, że dla każdej krawędzi $(u, v) \in E$ przynajmniej jedna z u lub v należy do C. rozmiar pokrycia wierzchołków to liczba wierzchołków w C.

Problem pokrycia wierzchołków polega na znalezieniu pokrycia wierzchołków o minimalnym rozmiarze dla danego grafu G. Wiadomo, że problem jest NP-zupełny, co oznacza, że jest mało prawdopodobne, aby istniał algorytm czasu wielomianowego, który rozwiązuje problem dla wszystkich instancji.

W związku z powyższym funkcja oceny będzie 'karała' za dużą liczbę wybranych wierzchołków (równych 1), ale jednocześnie 'premiowała' wyniki dla których znaleziono dużą liczbę pokrytych krawędzi.

3 Algorytm genetyczny

Algorytm ten służy do rozwiązywania problemów optymalizacyjnych. Algorytmy genetyczne są inspirowane procesami zachodzącymi w naturze, takimi jak ewolucja biologiczna, a ich celem jest znalezienie najlepszego (lub jednego z najlepszych) rozwiązania danego problemu w populacji.

W algorytmie tym rozwiązania problemu reprezentowane są przez osobniki w populacji, a każdy osobnik ma przypisaną wartość jakości rozwiązania.

Algorytm w badanym przypadku działa na zasadzie iteracyjnego przeszukiwania przestrzeni rozwiązań poprzez mutację osobników (zazwyczaj występuje jeszcze krzyżowanie) oraz ich późniejszą selekcję.

3.1 Zasada działania

Na początku algorytm tworzy losową populację kandydatów na rozwiązanie problemu. W każdej iteracji algorytmu (w każdym pokoleniu) losowane jest w turnieju potomstwo P_{k+1}('Tt' w programie) z pokolenia P_k, które reprezentuje nowe rozwiązania. Następnie przy pomocy mechanizmu mutacji (to znaczy losowej zmiany wartości jednej lub kilku cech osobnika) pokolenie jest modyfikowane. W kolejnym kroku następuje selekcja, która polega na wyborze najlepszych osobników z populacji na podstawie ich wartości funkcji celu (działa w sytuacji gdy liczba osobników w 'Tt' jest większa od populacji początkowej). Tak wyznaczony nowy zbiór zostaje zachowany do kolejnej iteracji. Na końcu zwracany jest najlepszy element z populacji to znaczy ten, który ma największe pokrycie krawędzi i spośród tych osobników ma najmniejszą liczbę wybranych wierzchołków.

Algorithm 1 Algorytm genetyczny

```
1: function GeneticAlgorithm(())
2:     Inicjalizuj populacje P o rozmiarze N
3:     Oceń populację
4:     for od 1 do LiczbaGeneracji: do
5:         Przeprowadź turniej w którym wybierzesz nowych osobników
6:         Przeprowadź mutację osobników
7:         Oceń zmutowanych osobników
8:         Przeprowadź selekcję z której ewentualnie wybierzesz N najlepszych
   osobników
9:     end for
10:    Zwróć najlepszego osobnika z populacji
11: end function
```

Parametry wejściowe:

- *edges*– krawędzie grafu
- *nodes*– wierzchołki grafu
- *num_generations*– liczba iteracji (pokoleń)
- *mutation_probability*– prawdopodobieństwo zajścia mutacji
- *mutation_rate*– siła mutacji
- *population_size*– rozmiar populacji
- *tournament_size*– liczba osobników brana do turnieju
- *selection_size*– ilość przeprowadzonych turnieji (liczba nowych osobników)

3.2 Poszczególne metody

- **mutation**(**individual**, **mutation_rate**)– metoda ta przyjmuje dwa parametry: wektor binarny (osobnik populacji) oraz siłę mutacji(im większy jest ten współczynnik, tym większa jest szansa na mutację.). Metoda ta jest wywoływana w momencie spełnienia warunku zajścia mutacji. Następnie z rozkładu normalnego (gdzie: średnia = mutation_rate x len(individual) i odchyleniem standardowe = 1) losowana jest liczba genów do mutacji, które następnie są losowo mutowane to znaczy ich pierwotna wartość jest odwracana.

- **tournament_selection**(**population**, **fitness_scores**, **tournament_size**, **selection_size**)– metoda ta przyjmuje następujące parametry:
  -**population**: lista osobników,
  -**fitness_scores**: lista wartości funkcji dopasowania dla każdego z osobników z populacji,
  -**tournament_size**: liczba osobników biorących udział w pojedyńczym turnieju (im większa wartość tego parametru, tym większe znaczenie będzie miał najlepszy z turniejowiczów),
  -**selection_size**: sumaryczna liczba osobników wybranych w turnieju(liczba turniejów).
  Metoda selekcji turniejowej polega na losowym wyborze z populacji kilku osobników (turniejowiczów), a następnie wyborze najlepszego z nich jako jednego z wybranych do kolejnej generacji. Powtarzamy ten proces tak długo, aż uzyskamy odpowiednią liczbę osobników.

  Na końcu funkcja zwraca listę osobników, którzy przeszli selekcję.

- **calculate_fitness**(**individual**, **edges**)– metoda ta przyjmuje jako argumenty: osobnika populacji (parametr 'individual' w postaci listy binarnej) oraz listę krawędzi grafu. Następnie obliczany jest koszt tego osobnika (kwadrat sumy elementów grafów = 1), liczba krawędzi pokrytych przez tego osobnika oraz łączna wartość funkcji oceny. Zwracany wynik jest oceną danego osobnika.

- **genetic_algorithm**()– jest to główna metoda klasy GeneticAlgorithm działająca zgodnie z zaprezentowanym wyżej pseudokodem. Selekcja osobników wybieranych do następnej populacji, polega na wybraniu najlepszych osobników na podstawie ich oceny wyznaczonej przez funkcje przystosowania. Jako wynik działania tego algorytmu zwracany jest najlepszy osobnik osobnik w populacji 'population', a następnie wśród osobników, którzy są równi najlepszemu, wybierany jest ten, który ma najmniejszą sumę wartości w wektorze.

4 Eksperymenty

Eksperymenty przeprowadzono dla takich parametrów:

- turniej o stałym rozmiarze równym 2 (tournament_size = 2)
- siła mutacji stała i równa 0,01 (mutation_rate = 0,01)
- 3 typy grafów (pełny, dwudzielny, losowy)
- rozmiar populacji równy 30 oraz 100 (population_size)
- ilość przeprowadzonych turniejów równa 30 oraz 100 (selection_size)
- dla każdego grafu sprawdzono algorytm dla liczby iteracji równej 50,100,200,500,1000 (num_generations) oraz do sprawdzenia wpływu wielkości populacji 20,40,60,80,100
- i prawdopodobieństwa mutacji równego 0,01 i 0,1 oraz 0,4(mutation_probability)

Dla każdego zestawu parametrów uruchomiano algorytm 51 razy i wyznaczano:

- wartość średnią pokrycia grafu z 51 uruchomień
- wartość minimalną pokrycia grafu z 51 uruchomień
- wartość maksymalną pokrycia grafu z 51 uruchomień
- wariancję dla parametru pokrycia grafu z 51 uruchomień

Parametr pokrycia grafu jest stosunkiem pokrytych wierzchołków do sumy wszystkich wierzchołków w grafie.

4.1 Użyte przy implementacji biblioteki

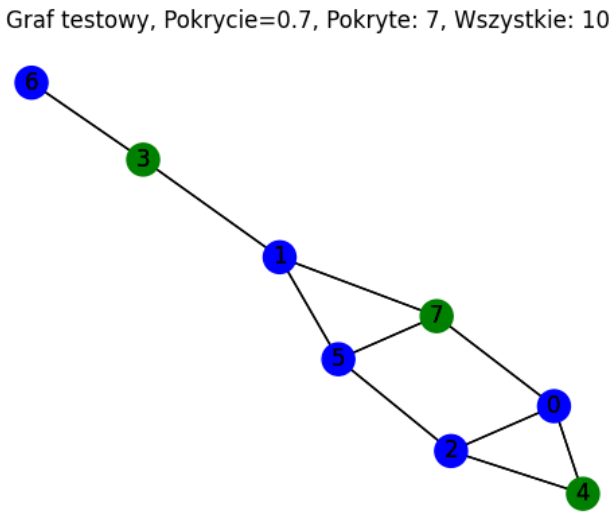
- numpy,
- matplotlib - wykresy,
- random - losowanie punktów początkowych,
- networkx - grafy

4.2 Testy poprawności działania

W pliku 'test_Methods' znajdują się podstawowe testy poprawności działania zaimplementowanych w klasie 'GeneticAlgorithm' metod.

Sprawdzono przy pomocy biblioteki networkx oraz matplotlib poprawność działania algorytmu genetycznego. W ramach testów przeprowadzono 5 uruchomień dla podstawowych parametrów algorytmu, dla każdego typu grafu. Wyniki przedstawiono na rysunkach. Implementacja tych testów znajduje się w pliku 'tests'.

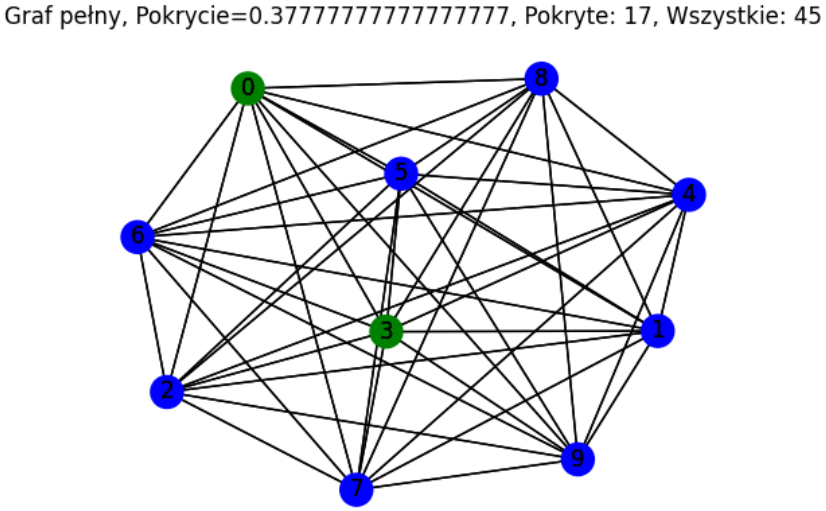
Wyniki przedstawiono na rysunkach: 1, 2, 3, 4



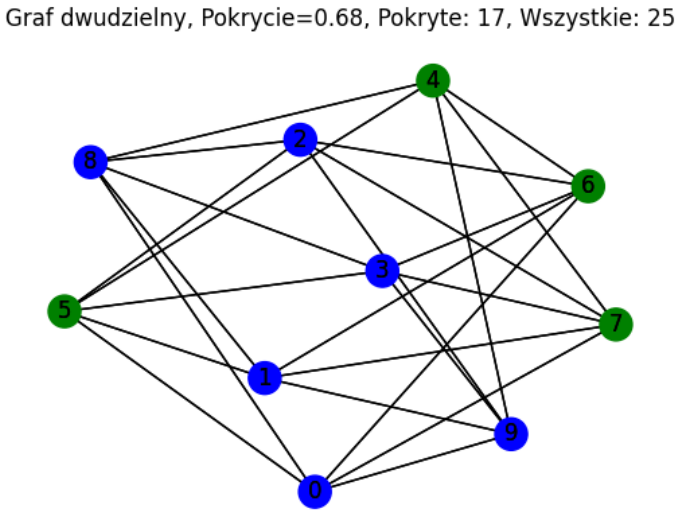
Rys. 1: Graf testowy

5 Wyniki dla różnego prawdopodobieństwa mutacji

Wyniki dla prawdopodobieństwa mutacji równego 0,01 przedstawiono na rysunku: 5,6 ,7
Wyniki dla prawdopodobieństwa mutacji równego 0,1 przedstawiono na rysunku: 8,9 ,10
Wyniki dla prawdopodobieństwa mutacji równego 0,4 przedstawiono na rysunku: 11,12 ,13
Najlepsze wyniki obserwuje się dla liczby iteracji równej 400-600. Dla małej

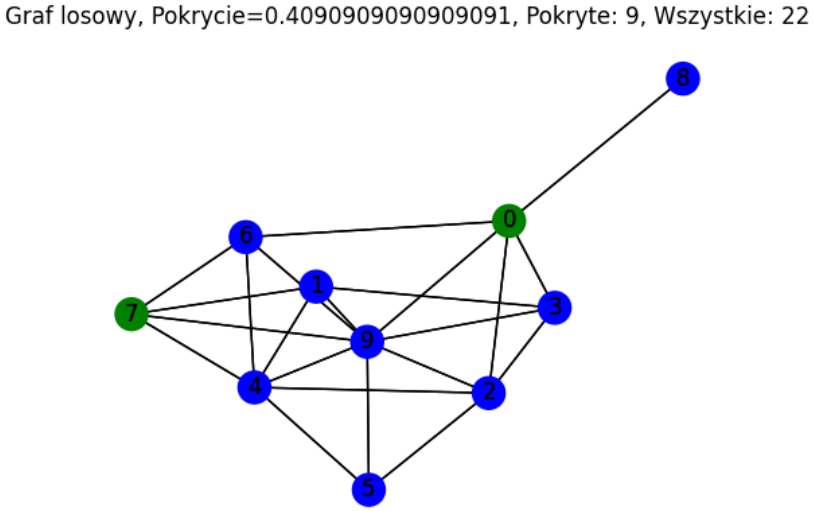


Rys. 2: Graf pełny



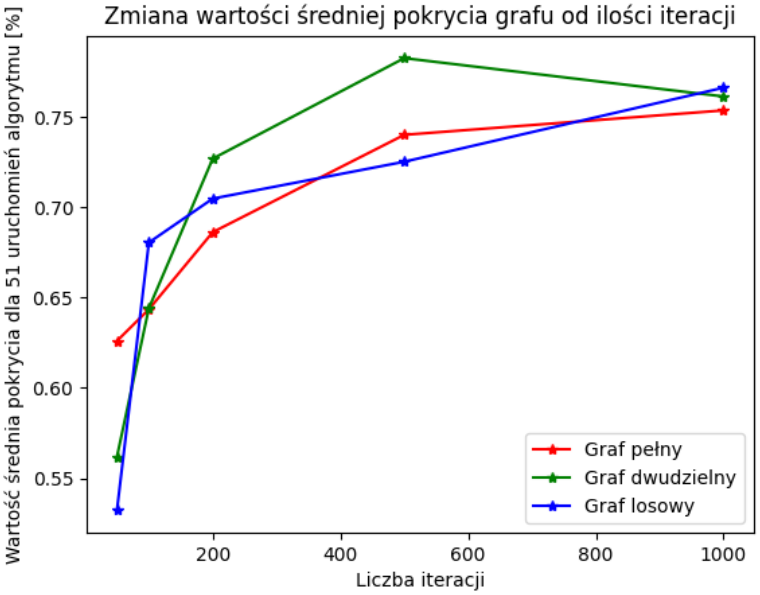
Rys. 3: Graf dwudzielny

liczby iteracji obserwuje się dużą wariancję, która wykładniczo maleje wraz ze wzrostem liczby iteracji.
Dla grafu losowego średnie wartości jego pokrycia zachowują się w sposób bar-

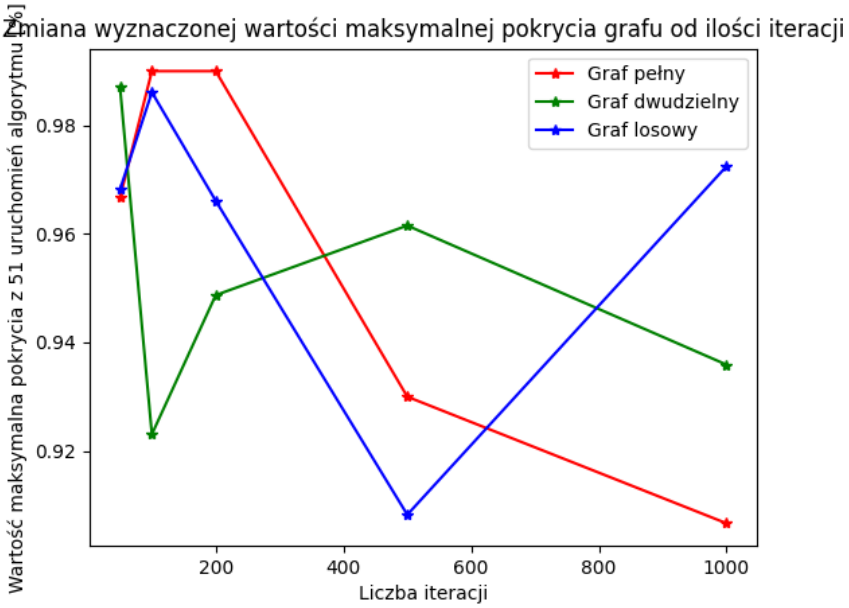


Rys. 4: Graf losowy

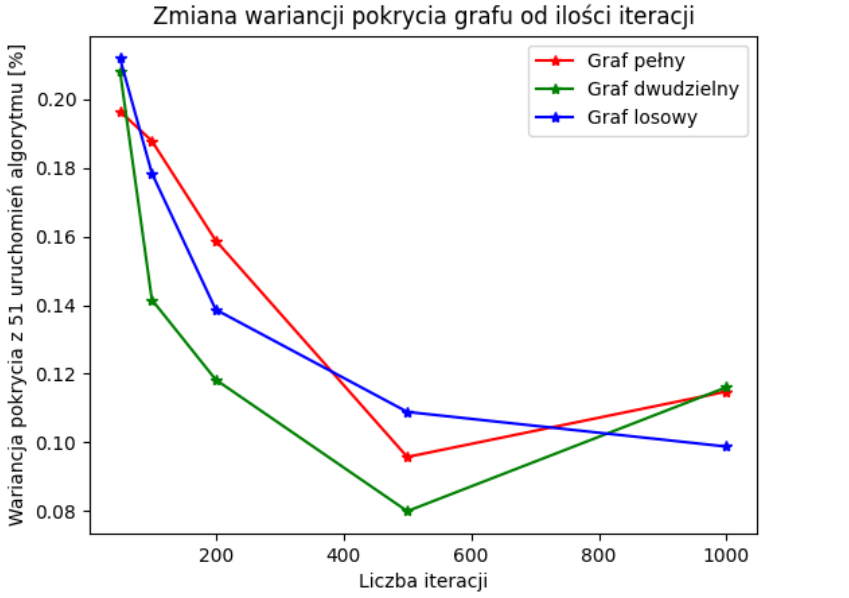
dziej losowy niż dla grafu pełnego lub dwudzielnego.
Wraz ze wzrostem mutacji zmienia się charakter wyznaczanych funkcji.



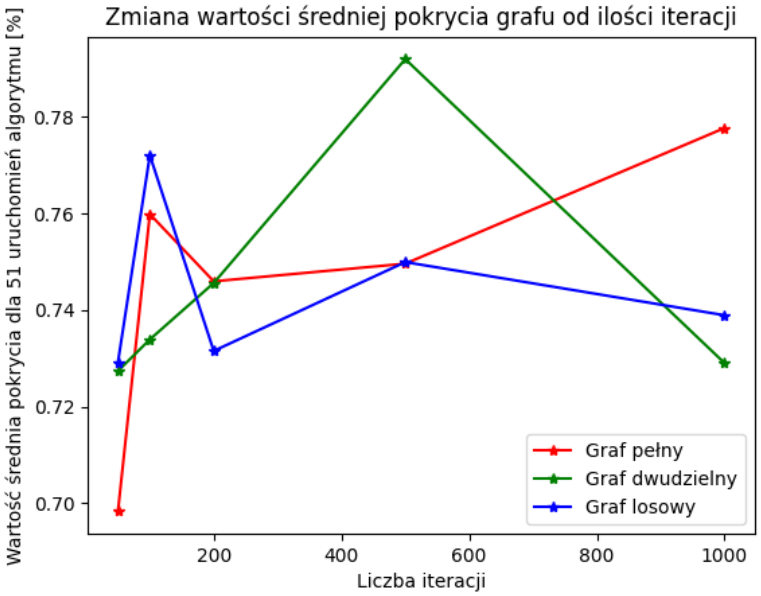
Rys. 5: Wyniki dla prawdopodobieństwa mutacji równego 0,01



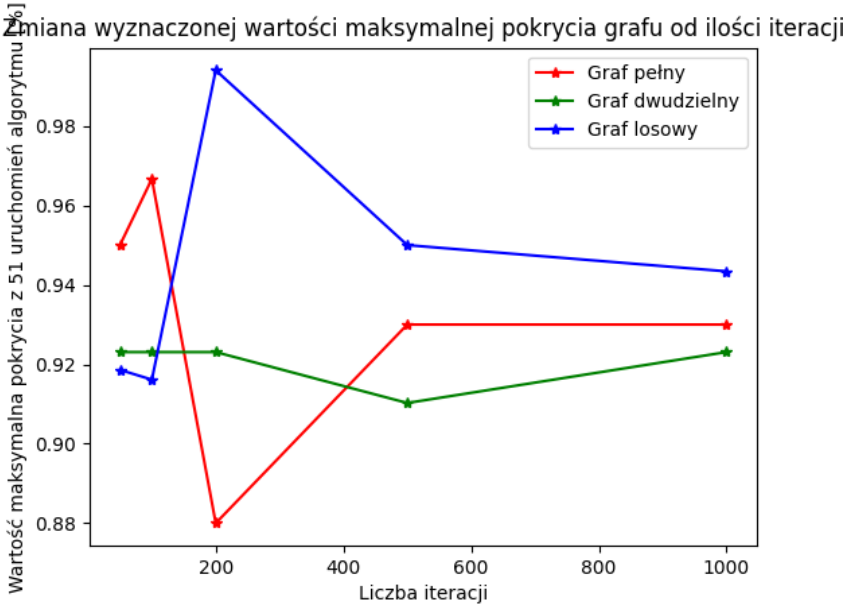
Rys. 6: Wyniki dla prawdopodobieństwa mutacji równego 0,01



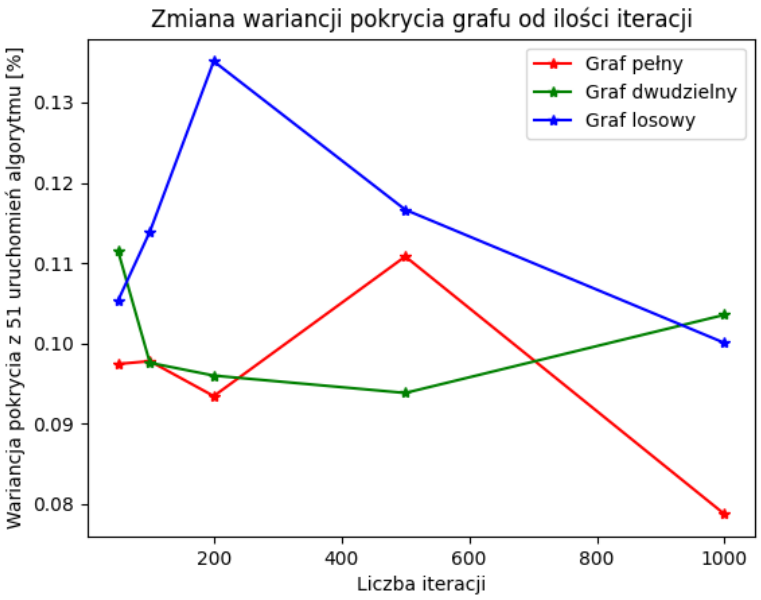
Rys. 7: Wyniki dla prawdopodobieństwa mutacji równego 0,01



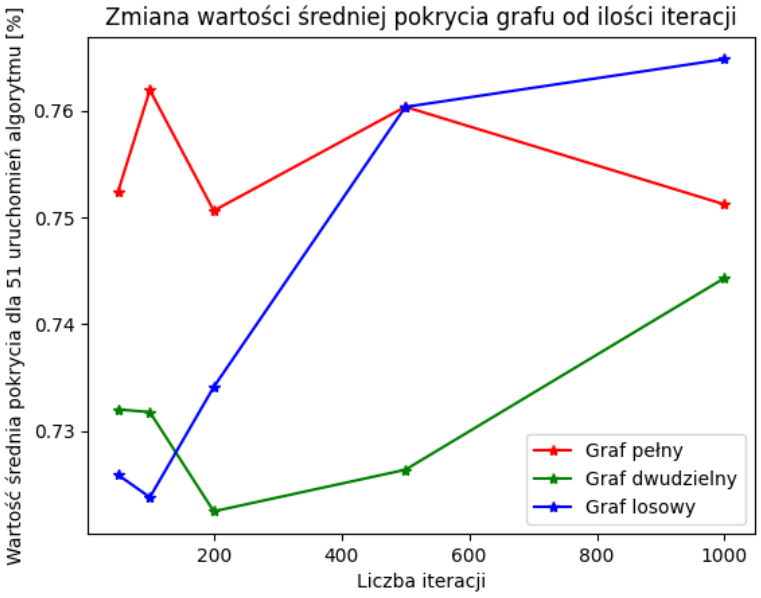
Rys. 8: Wyniki dla prawdopodobieństwa mutacji równego 0,1



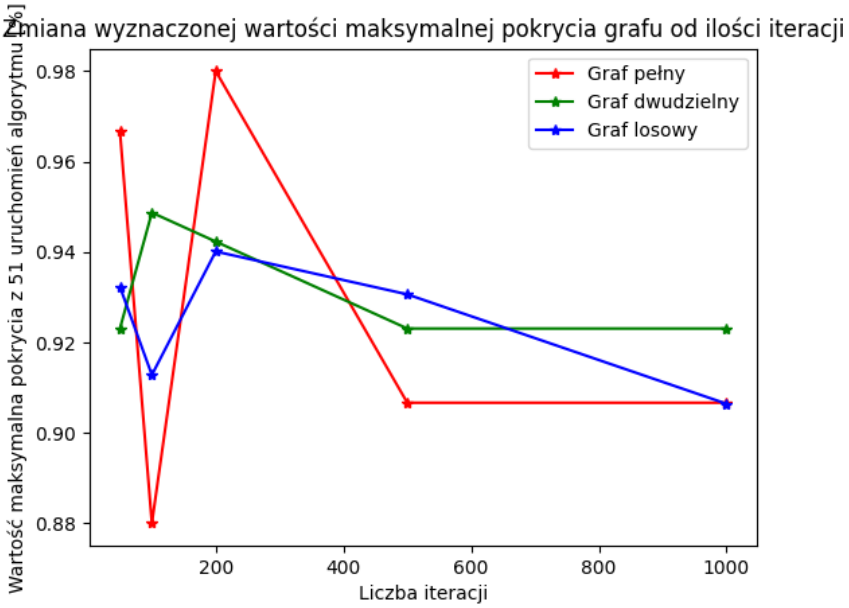
Rys. 9: Wyniki dla prawdopodobieństwa mutacji równego 0,1



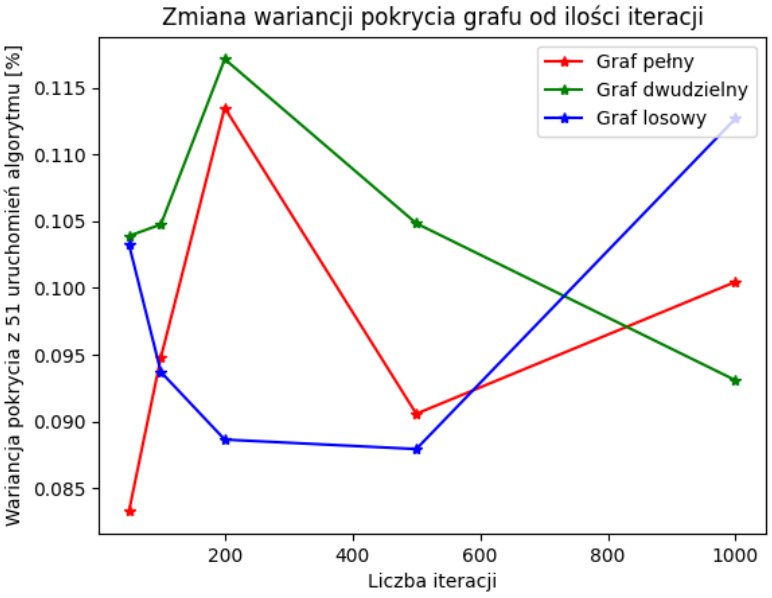
Rys. 10: Wyniki dla prawdopodobieństwa mutacji równego 0,1



Rys. 11: Wyniki dla prawdopodobieństwa mutacji równego 0,4



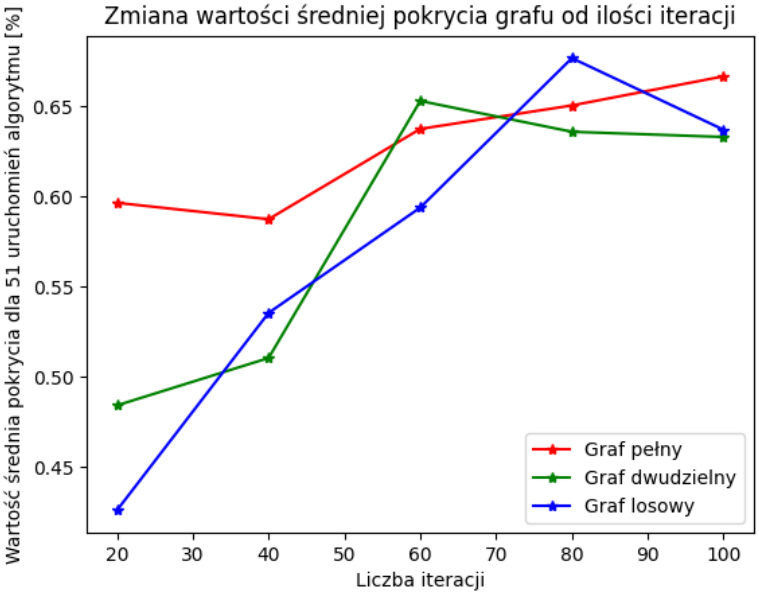
Rys. 12: Wyniki dla prawdopodobieństwa mutacji równego 0,4



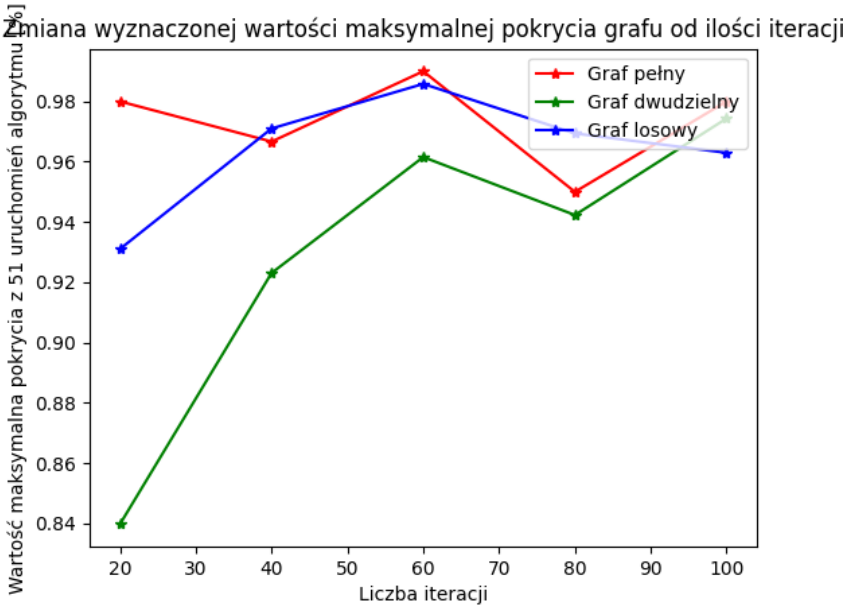
Rys. 13: Wyniki dla prawdopodobieństwa mutacji równego 0,4

6 Wyniki dla różnej populacji

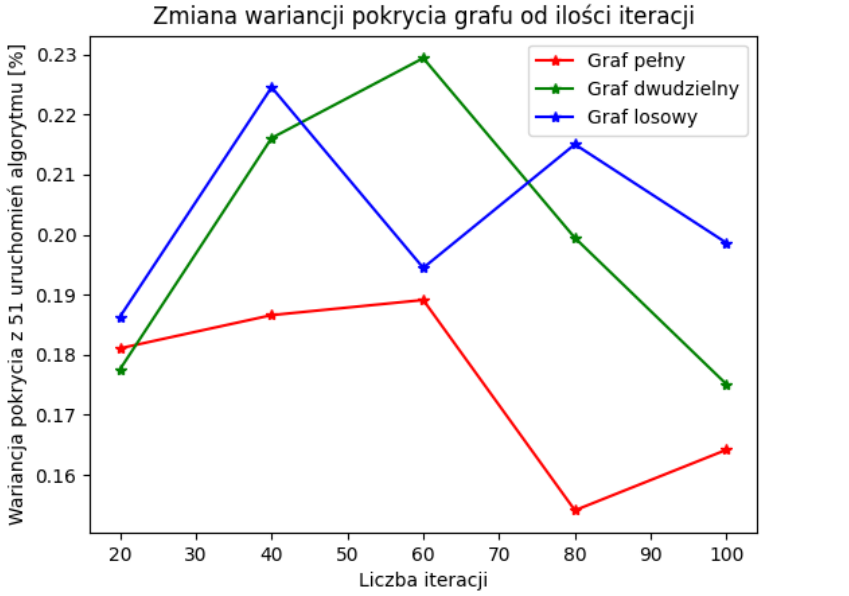
Eksperymenty przeprowadzano dla populcji równej 30 osobników i liczby iteracji równej 20,40,60,80,100 i prawdopodobieństwie mutacji równego 0,01. Wyniki przedstawiono na rysunkach:14,15 ,16
Wyniki dla populacji równej 100 osobników i liczby iteracji równej 20,40,60,80,100 i prawdopodobieństwie mutacji równego 0,01 przedstawiono na rysunkach:17,18 ,19
Dla dużej liczności populacji algorytm powinien szybciej znaleźć optymalne pokrycie grafu jednak porównując rysunek 14 z rysunkiem 17 takiej zależności nie widać. Dla dużej liczności populacji ilość iteracji nie ma aż takiego wpływu na co wskazuje płaski charakter funkcji z rys. 17.



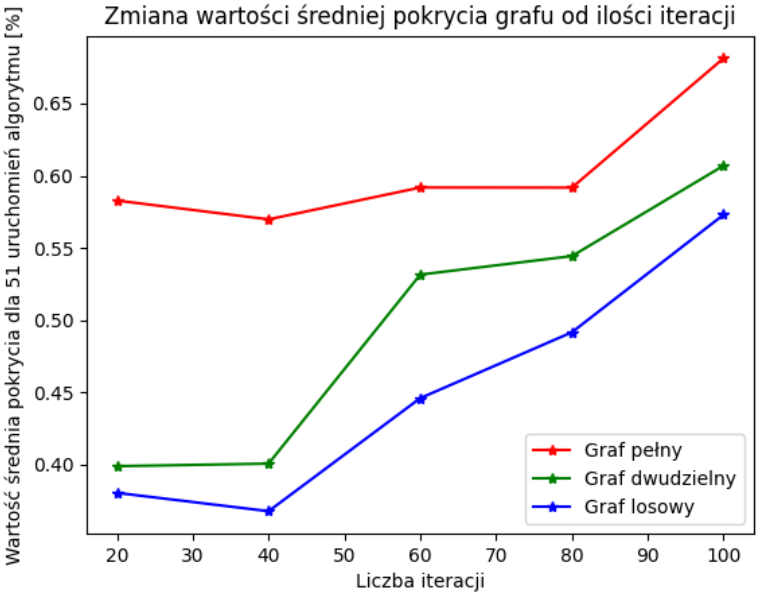
Rys. 14: Wyniki dla populacji równej 30



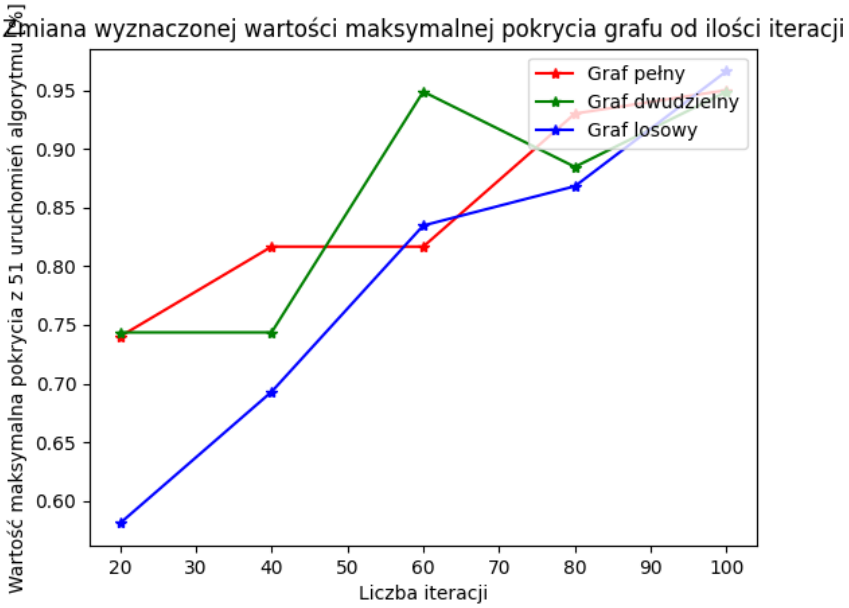
Rys. 15: Wyniki dla populacji równej 30



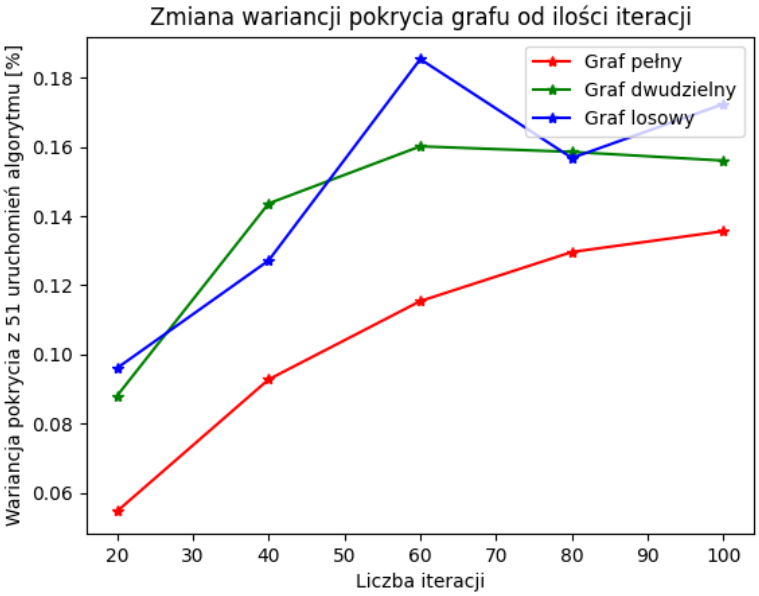
Rys. 16: Wyniki dla populacji równej 30



Rys. 17: Wyniki dla populacji równej 100



Rys. 18: Wyniki dla populacji równej 100



Rys. 19: Wyniki dla populacji równej 100