ELEKTRYKA 2018
Zeszyt 1 (145) Rok LXIV

Kamil MYŚLIWIEC
Politechnika Śląska, Gliwice

Bożena WIECZOREK
Politechnika Śląska, Gliwice

ALGORYTMY ROZWIĄZYWANIA PROBLEMU KOLOROWANIA GRAFU

Streszczenie. Głównym celem pracy było zbadanie algorytmów rozwiązujących problem kolorowania grafu, kolejno: algorytmów zachłannych LF (ang. Largest First) i SFL (ang. Saturated Largest First), algorytmu genetycznego sekwencyjnego oraz równoległego. Ponadto, zaimplementowana została aplikacja działającej w środowisku przeglądarki internetowej pozwalająca na wizualizacje 3D procesu kolorowania grafu wraz z regulacją parametrów grafu (takich jak liczba wierzchołków i gęstość grafu) oraz obserwację uzyskanych wyników (czasu wykonywania algorytmu i liczby dobranych kolorów).

Słowa kluczowe: problem kolorowania grafu, algorytmy zachłanne, algorytmy genetyczne, wizualizacja 3D

GRAPH COLOURING ALGORITHMS

Summary: The main aim of the study was to examine four algorithms concerning the graph colouring problem, respectively: LF (Largest First), SFL (Saturated Largest First), genetic algorithm, both sequential and multithreaded. Additionally, an application in a web browser environment was created to 3D visualisation of the graph colouring process allowing adjustment of graph parameters (such as number of vertices and graph density) and observation of the obtained results (execution time and number of colours).

Keywords: graph colouring problem, greedy algorithms, genetc algorithms, 3D visualisation

1. SFORMUŁOWANIE ROZWIĄZYWANEGO PROBLEMU

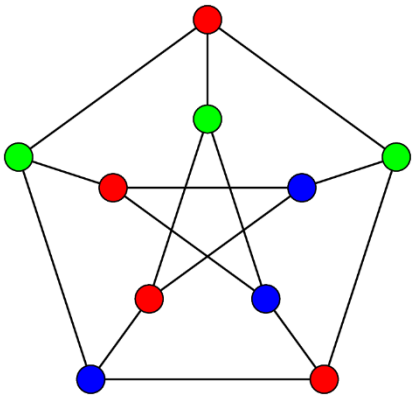
Kolorowanie grafów wykorzystywane jest w przypadku wielu skomplikowanych problemów optymalizacyjnych. Ich podstawowym celem jest wyszukanie optymalnego podziału wzajemnie wykluczających się rozwiązań wewnątrz określonego zbioru. Ze

względu na to, że algorytm kolorowania grafu należy do grupy najtrudniejszych algorytmów w kontekście złożoności obliczeniowej, jest on przedmiotem wielu badań i może pochwalić się bardzo długą, niemalże 200-letnią, historią bogatą w kilkaset wciąż otwartych problemów. Problem kolorowania grafu znany jest jako NP-zupełny [2], co oznacza, że dotychczas nie powstał algorytm, który dla każdego zbioru danych znajdzie optymalne rozwiązanie w czasie wielomianowym. Przykładami popularnych problemów, w których zastosowanie znajduje kolorowanie grafu są między innymi: odnalezienie możliwie najmniejszej liczby kolorów do pokolorowania mapy geograficznej (twierdzenie o czterech barwach [3]), tworzenie bezkolizyjnych harmonogramów (zajęć lekcyjnych, wydarzeń sportowych), planowanie zadań oraz przydział częstotliwości radiowych.

Grafem nazywamy jedną z podstawowych struktur danych używaną do modelowania wszelkiego rodzaju układów oraz sieci, które charakteryzują się skomplikowanymi zależnościami pomiędzy ich składnikami. Graf opisuje para elementów $G = (V, E)$, gdzie V oznacza zbiór wierzchołków grafu, zaś E – zbiór krawędzi. Kolorowanie grafu polega na przyporządkowaniu wierzchołkom lub krawędziom grafu kolorów w taki sposób, by żadne dwa sąsiednie wierzchołki (krawędzie) nie miały przyporządkowanego tego samego koloru i by liczba kolorów była minimalna. Szukana jest więc minimalna wartość parametru n dla następującej funkcji:

$$f: \rightarrow \{1, \ldots, n\}, n \in N, takie\ że\ \forall e = (v_1, v_2) \in E: f(v_1) \neq f(v_2) \tag{1}$$

Jeżeli dla grafu G liczba n jest minimalna, wartość parametru n nazywamy **liczbą chromatyczną** grafu G. Na rys. 1 przedstawiony został graf Petersona złożony z 10 wierzchołków oraz 15 krawędzi, pokolorowany przy użyciu 3 kolorów.



Rys. 1. Graf Petersona o liczbie chromatycznej = 3 [4]
Fig. 1. 3-chromatic Peterson graph [4]

2. OPIS ZAIMPLEMENTOWANYCH ALGORYTMÓW

2.1. Algorytmy zachłanne LF i SLF

Algorytmy LF i SLF należą do grupy algorytmów zachłannych, co oznacza, że w każdym kroku dokonują one wyboru najlepszego, częściowego rozwiązania - decyzji lokalnie optymalnej, dzięki czemu finalny rezultat może zostać osiągnięty we względnie krótkim czasie. Zadania rozwiązywane przy pomocy algorytmów zachłannych mają charakter optymalizacyjny, aczkolwiek uzyskane rozwiązanie niekoniecznie musi być optymalne. Algorytm LF możemy podzielić na dwa, następujące etapy:

1. Uporządkowanie wierzchołków grafu malejąco według ich stopni, przy czym stopień określa liczba krawędzi incydentnych z danym wierzchołkiem.
2. Pokolorowanie wierzchołków zachłannie, rozpoczynając od wierzchołka o najwyższym stopniu (zgodnie z uporządkowaną kolejnością). Kolorowanie zachłanne polega na przypisaniu rozpatrywanemu wierzchołkowi pierwszego dostępnego koloru.

Działanie algorytmu SLF sformułować można następująco: dopóki istnieje choć jeden niepokolorowany wierzchołek grafu, wyszukaj wierzchołek o największym stopniu spośród wierzchołków o maksymalnym **stopniu nasycenia**, przy czym stopień nasycenia równy jest liczbie różnych kolorów wierzchołków sąsiednich, a następnie przypisz mu pierwszy, wolny kolor.

2.2. Sekwencyjny i równoległy algorytm genetyczny

Algorytmy genetyczne należą do grupy algorytmów ewolucyjnych [1]. Wykorzystują one schemat działania polegający na symulacji procesu ewolucji i są silnie inspirowane mechanizmami obserwowanymi w przyrodzie. Pojęcia używane do opisu elementów i procesów zachodzących w algorytmach ewolucyjnych mają ścisły związek z genetyką i ewolucją. Ich cechą szczególną jest to, że korzystają jedynie z funkcji celu (nie wymagają dodatkowych informacji o rozwiązywanym problemie), pozwalającej określić jakość przetwarzanego rozwiązania. Algorytmy genetyczne są używane między innymi w przypadku problemów NP-trudnych, przy projektowaniu obwodów elektrycznych, przeszukiwaniu dużych przestrzeni rozwiązań, jak i modelowaniu procesów społecznych, ekonomicznych oraz biologicznych.

Działanie algorytmu rozpoczyna się od utworzenia **populacji**, czyli losowego zbioru osobników. W praktyce, każdy osobnik stanowi **chromosom**, który z kolei reprezentowany jest przez zestaw **genów** (zwany genotypem) oraz ocenę informującą o jakości danego rozwiązania (tak zwany stopień przystosowania osobnika). Genotyp stanowi zakodowaną informację, która pozwala na utworzenie **fenotypu**, będącego zestawem wartości odpowiadającym danemu genotypowi. Na każdy gen tworzący genotyp składa się **allel**, czyli wartość danego genu oraz **locus**, określający miejsce położenia genu w genotypie. Działanie algorytmu genetycznego można przedstawić w kilku następujących etapach:

1. Inicjalizacja populacji początkowej z losowo utworzonych osobników.
2. Ocena populacji, a następnie selekcja jednostek przy użyciu jednej z popularnych metod selekcji takich jak: ranking liniowy, koło ruletki lub turniej. Chromosomy zapewniające wyższą jakość (przystosowanie) mają znacznie większe prawdopodobieństwo wprowadzenia potomków do następnego pokolenia.
3. Genotypy wyselekcjonowanych osobników poddawane są następnie operatorom genetycznym:
   - **krzyżowaniu**, polegającym na łączeniu osobników w pary (rodziców), a następnie podziale zestawu genów obojga osobników w wylosowanym punkcie (lub przedziale) i ich wzajemne połączenie w celu utworzenia nowego genotypu,
   - **mutacji**, której zadaniem jest wprowadzenie drobnych, losowych zmian, takich jak odwracanie kolejności genów, czy zmiana bitów na wartości przeciwne.
4. W przypadku, gdy rozwiązanie nie zapewnia dostatecznie wysokiej jakości, algorytm powraca do kroku drugiego. Jeżeli osiągnięty został satysfakcjonujący rezultat, genotyp najlepszego osobnika znajdującego się w populacji to końcowy wynik działania algorytmu.

Zestaw parametrów determinujących przebieg algorytmu genetycznego został dobrany na podstawie przeprowadzonych eksperymentów. Najlepiej rokująca konfiguracja umieszczona została w tab. 1.

Tabela 1.

Zestaw parametrów algorytmu genetycznego

Nazwa parametru	**Wartość parametru**
Prawdopodobieństwo operacji krzyżowania	85%
Prawdopodobieństwo operacji mutacji	1.5%
Maksymalna liczba generacji	250
Liczba członków turnieju	5
Liczba osobników wykluczonych z selekcji	2

Algorytm kończy swoje działanie, kiedy spełniony zostaje przynajmniej jeden z dwóch, poniższych warunków:

1. Liczba generacji jest już równa maksymalnej dopuszczalnej wartości.

2. Populacja nie poprawiła wartości stopnia przystosowania (jakości populacji) w ciągu 100 generacji.

Do określenia stopnia przystosowania genotypu posłużyła funkcja jakości opisana wzorem:

$$y = -\left(\frac{c - 1 + f}{n}\right), \tag{2}$$

gdzie *c* to liczba użytych kolorów, *f* - liczba błędnych dopasowań, a *n* to liczba wierzchołków grafu.

W przypadku równoległego algorytmu genetycznego użyty został dokładnie ten sam zestaw parametrów. Dodatkowo, proces wykonywania algorytmu podzielony został na etapy, a każdemu z nich przydzielono konkretną liczbę wątków roboczych przeglądarki internetowej. Etapy wymienione w tab. 2 wykonywane są równolegle. Kiedy każdy z wątków roboczych przydzielonych do poszczególnego etapu zakończy swoje obliczenia, algorytm przechodzi do wykonywania kolejnych kroków sekwencyjnie.

Tabela 2.

Konfiguracja wątków roboczych

Nazwa etapu	**Liczba przydzielonych wątków roboczych**
Operacja krzyżowania	7
Operacja mutacji	2
Proces obliczania stopnia przystosowania	7

3. ARCHITEKTURA APLIKACJI

Podstawowym założeniem, z perspektywy architektury implementowanej aplikacji, było odseparowanie odpowiedzialności wizualizacji 3D od logiki kryjącej się za wykonaniem i monitorowaniem algorytmów. W tym celu utworzona została klasa *GraphColoring Application* pełniąca rolę **Mediatora** (wzorzec projektowy [6]). Zadaniem instancji tej klasy było zapewnienie komunikacji pomiędzy obiektem klasy *GraphComponent* w pełni odpowiedzialnej za utworzenie graficznej struktury widocznej na ekranie, a obiektem klasy *WorkerStream* silnie inspirowanej wzorcem projektowym **Obserwator** [7].

Zadaniem obiektu klasy *WorkerStream* było uruchomienie wątku roboczego przeglądarki, przesłanie pakietu danych informującego o liczbie wierzchołków, gęstości grafu oraz wybranym algorytmie i zapewnienie płynnej komunikacji do czasu zakończenia obliczeń. Dodatkowo klasa ta została wzbogacona o mechanizm buforowania odbieranych komunikatów, a czas opóźnienia jest konfigurowalny bezpośrednio z poziomu interfejsu użytkownika. Dzięki temu możliwa jest obserwacja procsu kolorowania dla małej liczby wierzchołków lub niewielkiej gęstości grafu.

Obiekt klasy *GraphComponent* odpowiedzialny jest za zarządzanie wizualizacją prezentowaną w oknie przeglądarki. Dodatkowo, nasłuchuje on podstawowych zdarzeń generowanych przez użytkownika, takich jak: poruszanie myszką oraz przybliżanie i oddalenie ekranu przy użyciu rolki myszy komputerowej, dzięki czemu może on zarządzać aktualnym położeniem kamery.

Kolejnym kluczowym elementem aplikacji jest instancja klasy *Worker*, która tworzona jest przez kontekst wątku roboczego przeglądarki. Dzięki mechanizmowi przesyłania pakietów, umożliwiona została komunikacja z natywnymi wątkami roboczymi. Każdy pakiet reprezentowany jest przez typ (w postaci tekstu) oraz dowolną paczkę danych, nieposiadającą z góry narzuconej struktury. Dzięki wykorzystaniu wątku roboczego do wykonywania algorytmu, użytkownik może swobodnie korzystać z interfejsu graficznego, między innymi poruszać się po grafie i obserwować proces kolorowania poszczególnych wierzchołków.

Wszystkie dostępne algorytmy reprezentowane są przez oddzielne klasy, a każda z nich implementuje interfejs *GraphColoringAlgorithm*. Jest to implementacja wzorca projektowego **Strategia**, dzięki czemu jedyną modyfikacją w przypadku zmiany aktualnie wybranego algorytmu, jest przekazanie instancji obiektu innej klasy [8].

Klasa *GeneticAlgorithm*, czyli implementacja najbardziej złożonego z badanych algorytmów, korzysta z obiektowych reprezentacji populacji i chromosomu oraz wzbogacona jest o zestaw metod ściśle powiązanych z procesem wykonywania algorytmu, takich jak operatory genetyczne krzyżowania, mutacji oraz selekcja nowej populacji na podstawie turnieju. W przypadku równoległej implementacji algorytmu genetycznego, operatory genetyczne wykonywane są w kontekście wątków roboczych przeglądarki znajdujących się w puli wątków.

Skrypty uruchamiane za pośrednictwem przeglądarki działają w środowisku jedno-wątkowym. Nie udostępnia ona interfejsu pozwalającego na manualne tworzenie oraz niszczenie wątków i procesów. Aby zwiększyć elastyczność i możliwości nowoczesnych aplikacji internetowych, z biegiem czasu, zdecydowano o utworzeniu mechanizmu zwanego wątkami roboczymi (ang. web workers). Funkcjonalność ta pozwala na uruchomienie skryptu pojedynczego pliku w kontekście wątku roboczego, który nie ma dostępu do pamięci globalnej (nie istnieje obszar pamięci dostępny zarówno dla wątku głównego, jak i utworzonych wątków roboczych) oraz do interfejsu użytkownika. Jedyna droga komunikacji wątku głównego z wątkiem roboczym to wymiana serializowanych pakietów danych.

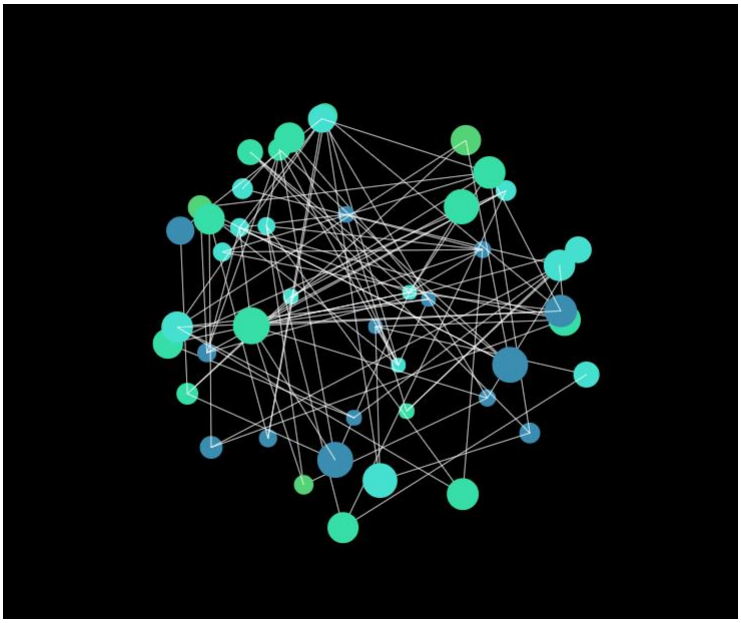
Zaimplementowana klasa *WorkerStream* odpowiedzialna jest za komunikację z natywnym wątkiem roboczym. Rozpoczęcie komunikacji poprzedzone jest wysłaniem pakietu danych zawierającego zestaw informacji determinujących działanie algorytmu, który powinien zostać uruchomiony w kontekście wątku roboczego. Następnie obiekt klasy *WorkerStream* nasłuchuje komunikatów wysyłanych przez wątek natywny, dopóki proces wykonywania algorytmu nie ulegnie zakończeniu.

Wątki robocze posiadają jedną istotną wadę - większość przeglądarek internetowych nie wspiera możliwości tworzenia kolejnych wątków bezpośrednio z poziomu wątku roboczego. Wszystkie wątki robocze muszą zostać utworzone przez wątek główny. Tworzy to zasadniczy problem z perspektywy implementacji równoległego algorytmu genetycznego, w którym konieczne było utworzenie kilkunastu wątków roboczych i zapewnienie synchronizacji pomiędzy nimi. W konsekwencji jedynym możliwym rozwiązaniem była implementacja modelu opartego o paradygmat Master-Slave, gdzie zadaniem wątku głównego poza zarządzaniem wizualizacją graficzną, jest nasłuchiwanie i grupowanie komunikatów ze wszystkich wątków roboczych [5].

4. INTERFEJS UŻYTKOWNIKA

Interfejs użytkownika podzielony jest na dwie sekcje - podgląd grafu i boczny panel konfiguracyjny. Po zakończeniu procesu kolorowania grafu, wyświetlane jest okno zawierające informacje dotyczące kolejno: użytego algorytmu, czasu wykonywania zadania i liczby dobranych kolorów.

Trójwymiarowa wizualizacja grafu odzwierciedla rzeczywiste relacje pomiędzy wierzchołkami w aktualnie przetwarzanym grafie. Domyślnie wszystkie wierzchołki pokolorowane są kolorem szarym. Na rys. 2 znajduje się pokolorowany graf złożony z 45 wierzchołków i gęstości równej 5%.



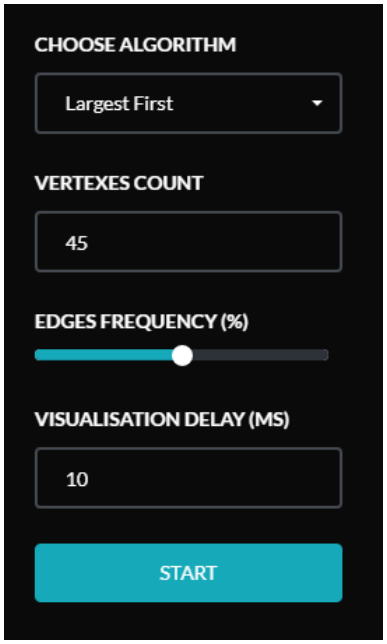
Rys. 2. Trójwymiarowa wizualizacja grafu złożonego z 45 wierzchołków i gęstości równej 5%
Fig. 2. 3D visualisation of the graph on 45 vertices and density equal to 5%

Interfejs użytkownika umożliwia swobodne poruszanie się po grafie przy użyciu myszy komputerowej. Kamera podąża za ruchem kursora, dzięki czemu możliwe jest przyjrzenie się strukturze z dowolnej perspektywy. Ponadto, istnieje możliwość przeglądania grafu bezpośrednio z jego środka (rolka w myszce komputerowej umożliwia przybliżanie, jak i oddalanie się kamery).

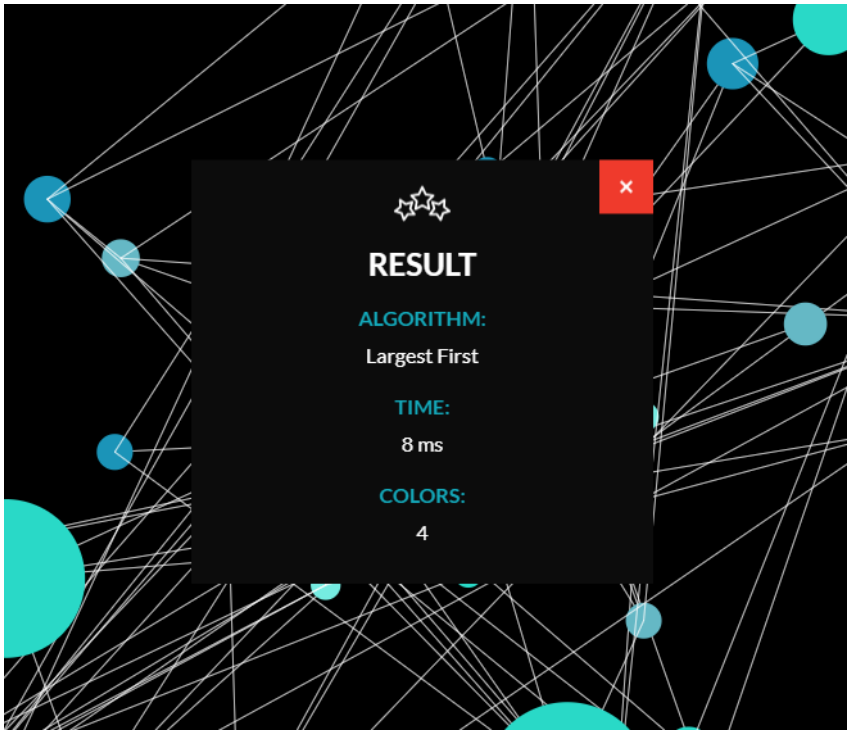
Panel konfiguracyjny (rys. 3) znajdujący się po prawej stronie ekranu umożliwia wybór:

- algorytmu,
- liczby wierzchołków,
- gęstości grafu,
- opóźnienie kolorowania grafu względem wykonywania algorytmu.

Po pokolorowaniu całego grafu, wyświetlane jest okno zawierające wyniki wykonywania algorytmu (rys. 4). Są to kolejno: nazwa użytego algorytmu, czas jego wykonywania oraz liczba dobranych kolorów.



Rys. 3. Panel konfiguracyjny
Fig. 3. Configuration panel



Rys. 4. Prezentacja wyników działania algorytmu
Fig. 4. The algorithm result presentation

5. WYNIKI PRZEPROWADZONYCH EKSPERYMENTÓW

Podstawowym celem badań było porównanie czasów działania oraz wyników otrzymanych przez poszczególne algorytmy. Dla algorytmów zachłannych wykonano po 100 eksperymentów dla losowo wygenerowanych grafów o różnych parametrach. Uśrednione wyniki dla algorytmów LF i SLF umieszczone zostały w tab. 3 i 4.

Tabela 3.

Średnia liczba kolorów (L) oraz czas wykonywania (T) algorytmu LF w zależności od liczby wierzchołków (I) oraz gęstości grafu (K)

I/K	5%		10%		25%		50%		75%		90%	
	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L
5	0,55	2,37	0,55	2,37	0,45	2,69	0,35	3,49	0,31	4,49	0,36	4,80
10	0,52	2,74	0,38	2,99	0,36	3,79	0,55	5,75	0,58	8,04	0,43	9,59
25	0,87	3,34	0,92	4,11	1,42	6,61	2,11	10,83	2,81	16,38	2,78	22,54
50	3,11	4,40	5,21	6,12	12,09	10,59	13,83	18,42	18,28	27,71	17,94	40,92
75	11,15	5,39	19,74	7,85	34,07	14,25	50,26	24,99	64,40	38,06	66,18	58,09
100	24,04	6,33	39,09	9,41	87,69	17,72	113,40	31,42	177,41	48,61	164,29	72,37
150	69,52	8,07	133,83	12,34	195,51	23,80	332,34	43,51	397,44	69,55	426,51	99,58

Tabela 4.

Średnia liczba kolorów (L) oraz czas wykonywania (T) algorytmu SLF w zależności od liczby wierzchołków (I) oraz gęstości grafu (K)

I/K	5%		10%		25%		50%		75%		90%	
	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L
5	0,45	2,33	0,27	2,46	0,19	2,80	0,18	3,53	0,14	4,52	0,16	4,88
10	0,23	2,71	0,23	0,90	0,27	3,85	0,58	5,61	0,51	8,09	0,56	9,64
25	1,59	3,17	2,07	4,18	2,75	6,80	4,86	10,78	6,47	16,72	5,85	22,40
50	8,16	4,26	13,04	6,15	30,08	11,03	47,27	18,79	47,25	28,59	65,51	41,31
75	22,9	5,44	29,79	7,99	91,98	14,64	114,51	25,83	115,52	39,96	130,92	57,98
100	37,5	6,44	78,28	9,74	143,7	17,92	272,84	32,46	320,03	50,92	320,92	73,05
150	181,1	8,23	205,0	12,8	414,9	24,32	794,7	44,83	1012,4	72,40	1212,9	100,6

Dla algorytmów genetycznych, ze względu na dłuższe czasy działania, wykonano po 10 eksperymentów dla liczby wierzchołków równej 5, 10, 25, 50, 75 i 100 oraz gęstości grafów takiej jak dla algorytmów zachłannych. Wyniki zostały przedstawione w tab. 5 i 6.

Tabela 5.

Średnia liczba kolorów (L) oraz czas wykonywania (T) sekwencyjnego algorytmu genetycznego w zależności od liczby wierzchołków (I) oraz gęstości grafu (K)

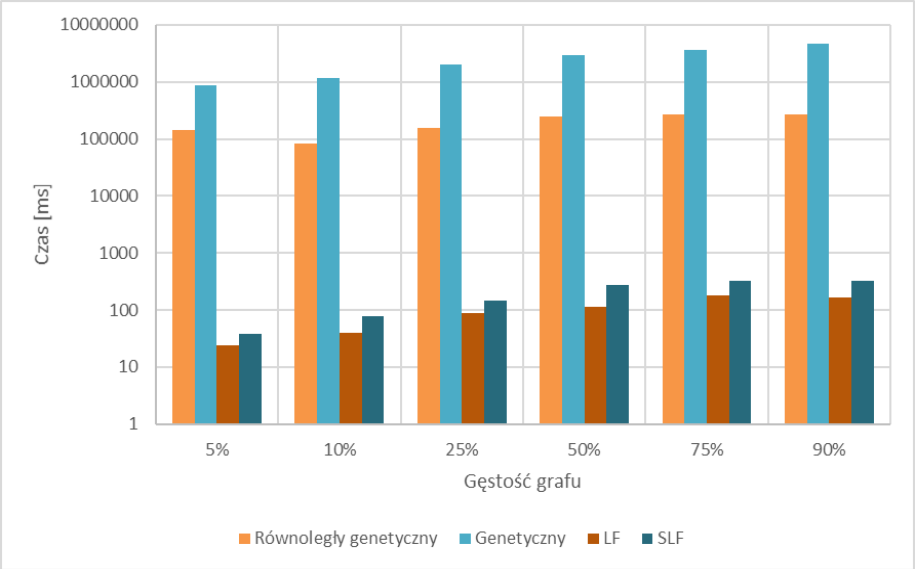
I/K	5%		10%		25%		50%		75%		90%	
	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L
5	35,14	2,96	18,91	2,84	18,21	3,30	19,65	4,08	19,85	4,10	20,83	4,54
10	259,69	2,83	201,17	3,15	198,02	4,49	319,41	6,16	337,64	8,07	234,82	8,83
25	4170	4,7	4870	5,87	7695	8,42	11758	12,4	19514	15,83	14649	20,68
50	26720	7,62	47983	9,80	88637	15,22	144883	23,87	188853	31,49	244027	38,23
75	152736	12,01	2404667	15,29	549338	19,18	952806	31,21	1059160	45,14	1086449	58,77
100	885328	17,19	1181235	21,35	2028562	32,57	2991001	45,09	3686089	58,73	4716207	70,41

Tabela 6.

Średnia liczba kolorów (L) oraz czas wykonywania (T) równoległego algorytmu genetycznego w zależności od liczby wierzchołków (I) oraz gęstości grafu (K)

I/K	5%		10%		25%		50%		75%		90%	
	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L	T [ms]	L
5	859	3,15	1073	2,81	713	3,42	419	3,28	482	3,61	219	4,22
10	957	4,04	682	3,47	822	4,26	730	6,22	650	7,87	823	8,63
25	3074	6,19	2491	6,21	3048	8,66	3056	12,65	3652	17,43	3086	20,86
50	8367	13,03	11099	9,22	12056	14,49	13110	23,04	18024	30,90	17960	39,23
75	21710	13,92	27264	17,13	35547	21,16	47933	34,71	50268	47,06	50238	57,54
100	145691	19,88	81537	19,56	155702	27,18	247270	41,12	269175	52,25	269003	66,61

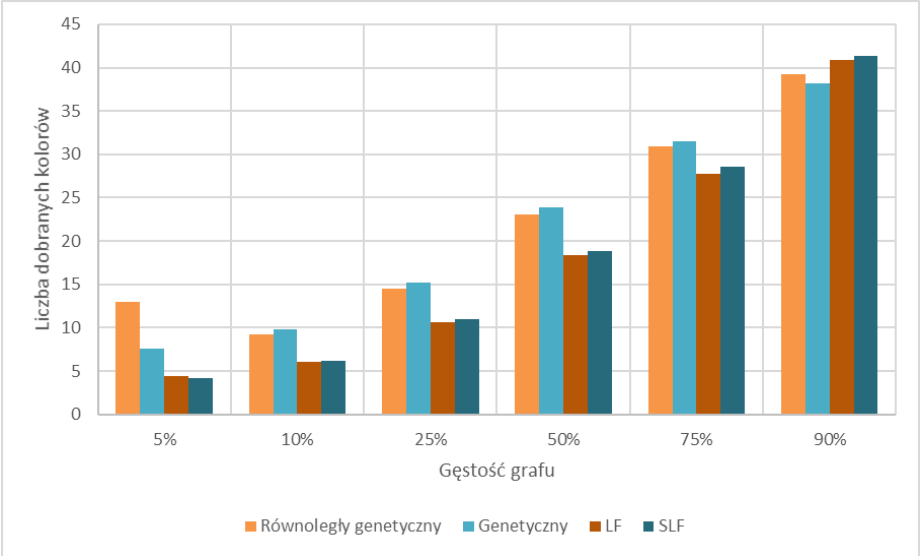
Na rys. 5 porównano czasy działania zaimplementowanych algorytmów dla grafu o 100 wierzchołkach i gęstości równej 5%, 10%, 25%, 50% 75% oraz 90%.



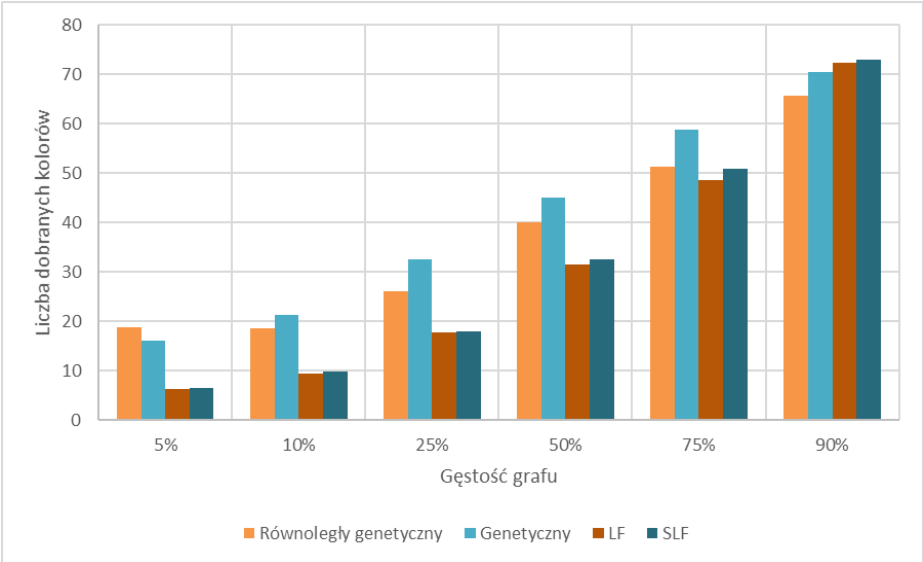
Rys. 5. Wykres zależności czasu wykonywania algorytmów od gęstości grafu dla grafu złożonego ze 100 wierzchołków
Fig. 5. Execution time vs. graph density for graph on 100 vertices

Można zauważyć, że czas działania algorytmów zachłannych jest zdecydowanie krótszy niż czas działania algorytmów genetycznych. Jest to związane ze sposobem działania algorytmów genetycznych, które by osiągnąć dobrą jakość rozwiązania muszą przetworzyć znaczną liczbę generacji (maksymalnie 250). Zgodnie z przewidywaniami, wraz ze wzrostem gęstości grafu, czasy działania algorytmów rosną. Dodatkowo, możemy zauważyć wyraźną różnicę w szybkości działania równoległego algorytmu genetycznego w porównaniu do implementacji sekwencyjnej.

Następnie porównane zostały wyniki uzyskane przez poszczególne algorytmy dla grafów o 50 i 100 wierzchołkach oraz przebadanych wartościach gęstości (rys. 6 i 7). Rysunki pokazują, że algorytmy zachłanne znajdują lepsze rozwiązania dla grafów o gęstościach równych 5%, 10%, 25%, 50% oraz 75%. Jedynie dla grafów o gęstości 90% algorytmy genetyczne dobierają mniejszą liczbę kolorów. Algorytmy LF i SLF uzyskują bardzo zbliżone do siebie wyniki, zaś pomiędzy algorytmami genetycznymi niewielką przewagę ma równoległa implementacja, która zwraca lepsze rozwiązania w 9 na 12 przypadków.



Rys. 6. Wykres zależności liczby dobranych kolorów od gęstości grafu dla grafu złożonego z 50 wierzchołków
Fig. 6. Number of colours vs. graph density for graph on 50 vertices



Rys. 7. Wykres zależności liczby dobranych kolorów od gęstości grafu dla grafu złożonego z 100 wierzchołków
Fig. 7. Number of colours vs. graph density for graph on 100 vertices

6. PODSUMOWANIE

Problem kolorowania grafu należy do grupy problemów NP-zupełnych. W konsekwencji odnalezienie optymalnego rozwiązania, czyli liczby chromatycznej jest bardzo kosztowne, a czas wykonywania algorytmu wzrasta wykładniczo wraz ze wzrostem liczby wierzchołków. Algorytmy siłowe, który zawsze odnajdują optymalne rozwiązanie, przestają być użyteczny w przypadku dużych grafów. Z tego względu, do kolorowania takich grafów nadają się jedynie algorytmy wyznaczające rozwiązania przybliżone. W pracy badaniom poddane zostały algorytmy heurystyczne LF i SLF charakteryzujące się postępowaniem zachłannym oraz algorytm genetyczny, który jest silnie inspirowany analogiami biologicznymi. W przypadku algorytmu genetycznego zbadana została zarówno implementacja sekwencyjna, jak i równoległa. Przeprowadzone eksperymenty wykazały, że algorytmy zachłanne najlepiej poradziły sobie z wykonaniem powierzonego im zadania znajdując lepsze jakościowo rozwiązania w krótkim czasie. Wyniki uzyskane przy użyciu algorytmów genetycznych były lepsze jedynie dla bardzo gęstych grafów (90%), jednak czas poświęcony na ich wyznaczenie był wielokrotnie większy niż dla algorytmów zachłannych. Oznacza to, że dla problemu kolorowania grafu algorytmy zachłanne sprawdzają się lepiej niż algorytmy genetyczne, ponieważ są w stanie odnaleźć akceptowalne rozwiązanie w znacznie krótszym czasie.

BIBLIOGRAFIA

1. Figielska E. „Algorytmy ewolucyjne i ich zastosowania”, Biblioteka Narodowa, 2006
2. https://pl.wikipedia.org/wiki/Proble_NP-zupełny
3. https://pl.wikipedia.org/wiki/Twierdzenie_o_czterech_barwach
4. https://graphtheoryinlatex.wordpress.com/2010/02/18/a-coloring-of-the-petersen-graph-2/
5. https://en.wikipedia.org/wiki/Master/slave_(technology)
6. https://pl.wikipedia.org/wiki/Mediator_(wzorzec_projektowy)
7. https://msdn.microsoft.com/pl-pl/library/ee850490(v=vs.110).aspx
8. http://blog.helion.pl/wzorzec-projektowy-strategii/

Kamil MYŚLIWIEC
Politechnika Śląska
Wydział Elektryczny, Instytut Elektrotechniki i Informatyki
Akademicka 10
44-100, Gliwice
mail@kamilmysliwiec.com

Dr inż. Bożena WIECZOREK
Politechnika Śląska
Wydział Elektryczny, Instytut Elektrotechniki i Informatyki
Akademicka 10
44-100, Gliwice
bozena.wieczorek@polsl.pl