Politechnika Warszawska
Wydział Matematyki i Nauk Informacyjnych

Chromatyczna Teoria Grafów

Dokumentacja projektowa wstępna

Chimedshirchin Batjargal, Mateusz Rymuszka

23 marca 2019

Spis treści

1 Abstrakt

Przedmiotem projektu realizowanego w ramach przedmiotu Chromatyczna Teoria Grafów przez autorów tego dokumentu jest analiza problemu kolorowania warstwowego grafu. Zespół przygotuje, zaimplementuje oraz przetestuje działanie trzech algorytmy, które będą starały się pokolorować grafy wielowarstwowe w lepszy sposób niż naiwny algorytm duplikacji koloru na wiele warstw. W dokumentacji przedstawi teoretyczny opis problemu wraz z proponowanymi algorytmami, a następnie opisze sposób działania programu i przedstawi raport z testów na wybranych rodzajach grafów, podsumowując to wszystko wnioskami płynącymi z obserwacji.

2 Opis teoretyczny problemu

Definicja 1. ***Kolorowaniem*** p***-warstwowym*** *grafu* G *nazywamy takie przyporządkowanie* $c : v \to 2^C$*, gdzie każdemu wierzchołkowi* $v \in V(G)$ *przypisujemy podzbiór* C' *zbioru kolorów* C *taki, że* $|C'| = p$.

Definicja 2. *Kolorowanie* p*-warstwowe grafu* G *nazywamy* ***właściwym (poprawnym, optymalnym)****, jeżeli dla dowolnego* $v \in V(G)$ *przecięcia zbioru kolorów tego wierzchołka i kolorów każdego jego sąsiada są zbiorami pustymi, tzn.*

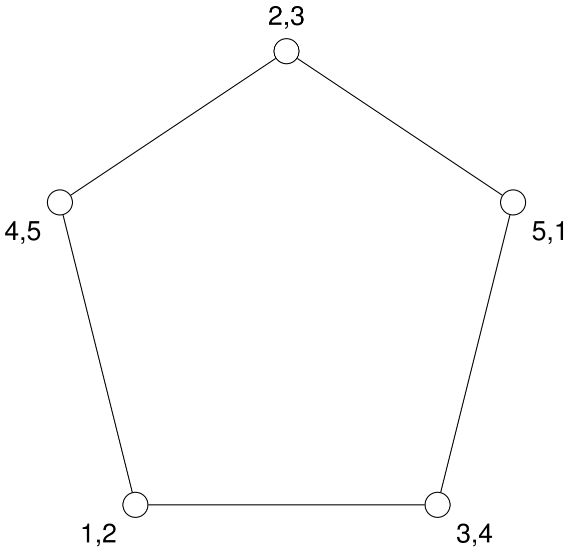
$$\forall u, v \in V(G) \quad \{u, v\} \in E(G) \implies c(u) \cap c(v) = \emptyset$$

Ogólnie rzecz ujmując, kolorowanie wielowarstwowe grafu jest rozszerzeniem zwykłego kolorowania grafu na wiele wymiarów. Kolorowanie jednowarstwowe jest tożsame z klasyczną definicją kolorowania wierzchołkowego grafu.

Definicja 3. p***-warstwową liczbą chromatyczną*** *grafu* G *nazywamy najmniejsze* q *takie, że istnieje poprawne* p*-warstwowe pokolorowanie grafu* G *używające* q *kolorów.*
Chromatycznym współczynnikiem p***-warstwowym*** *grafu* G *nazywamy stosunek* p*-warstwowej liczby chromatycznej do liczby warstw.*

Okazuje się, że $\chi_p G \leq p\chi G$. Gdy weźmiemy bowiem dobre χG-pokolorowanie $c : V \to C$ grafu G i dla każdego $v \in V$ przypiszemy mu $c'(v) = \{r \cdot |C| + c(v) : r \in \{1, ..., p\}\}$, to uzyskamy $p\chi G$-pokolorowanie p-warstwowe. Nazwijmy je pokolorowaniem p-warstwowym grafu G **równoległym do klasycznego**.

Przykładem grafu, który dla którego zachodzi ostra nierówność, jest nieparzysty cykl o wielkości przekraczającej trzy wierzchołki. Takie kolorowanie będziemy nazywać **lepszym od klasycznego**.



Rysunek 1: Przykład 2,5-pokolorowania 2-warstwowego dla grafu C_5

Problem polega na znalezieniu takiego p-warstwowego pokolorowania dla zadanego p, aby dla pewnych grafów (takich, gdzie jest to możliwe) $\chi_p G < p\chi G$.

3 Proponowane algorytmy

Ponieważ problem znalezienia p-warstwowego grafu jest NP-trudny (można to wykazać, rozwijając graf warstwowo, gdzie każdy wierzchołek tworzy p-krotną klikę, zachowując połączenia z wierzchołkami w każdej z warstw), nie możemy znaleźć takiego pokolorowania w czasie wielomianowym. Zespół skupi się na poszukiwaniu algorytmu aproksymacyjnego, który przy zastosowaniu odpowiedniego podejścia do problemu, a być może pewnych heurystyk, będzie w stanie osiągać rezultat dla niektórych grafów.

Wspólnym mianownikiem tych algorytmów powinna być troska, aby starać się kolorować ćhaotycznie", tzn. nie powtarzać tych samych zbiorów kolorów w wierzchołkach niezależnych. Taki układ może nas potem zmusić do użycia większej liczby kolorów, a w efekcie uzyskamy pokolorowanie na $p \cdot \chi G$ kolorów.

3.1 Zmodyfikowany algorytm AMIS

Algorytm AMIS (*ang. approximately maximum independent set*) jest jednym z efektywniejszych algorytmów wielomianowych kolorowania grafu. W klasycznym problemie, polega on na wyborze spośród niepokolorowanych wierzchołków zbioru niezależnego i pokolorowanie go na nowy kolor.

Algorytm można byłoby zaaplikować bez większych zmian (dopóty wierzchołek uznajemy za niepokolorowany, dopóki nie posiada zbioru p kolorów przypisanych do niego). Istnieje jednak ryzyko, że wybierając te same zbiory niezależne w kolejnych iteracjach algorytmu znajdowania zachłannego zbiorów niezależnych, uzyskalibyśmy pokolorowanie równoległe do klasycznego. Zależy nam zatem, aby w kolejnych iteracjach zbiory niezależne jednocześnie nie były tożsame oraz nie były rozłączne.

Zaproponowany algorytm będzie korzystał ze zoptymalizowanej wersji algorytmu GIS (*ang. greedy independent sets*), w którym zamiast wyboru jednego wierzchołka, będziemy wybierać dla każdego wierzchołka, czy powinien się on znaleźć w nowym zbiorze wierzchołków niezależnych. Będziemy również pamiętać dla każdego wierzchołka, ile razy znalazł się on już w zbiorze niezależnym. Po wyborze wierzchołków do naszego zbioru niezależnego, będziemy analizować wszystkie krawędzie między kandydatami i wyrzucać jeden z wierzchołków. Będzie to wierzchołek, który znalazł się więcej lub mniej razy w zbiorze niezależnym (wyboru dokonujemy na zmianę).

3.1.1 Pseudokod

```
Wejście: graf G i tablica wystąpień w zbiorach niezależnych wierzchołków f

function GreedyIndependentSet(G, f)
    I := ∅
    old := true

    for all v ∈ V(G) do
        if unif(0,1) ≤ deg_G^{-1} v then
            I := I ∪ {v}
        end if
    end for
    for all {u,v} ∈ E(G) do
        if u,v ∈ I then
            if f(u) ≤ f(v) ⟺ old then
                I := I − {u}
            else
                I := I − {v}
            end if
            old := ¬ old
        end if
    end for
    return I ∪ GreedyIndependentSet(G[V(G) − I], f)
end function
```

```
Wejście: graf G

function Color(G)
    colored := 0
    color := 1
    c := []
    f := []

    for all v ∈ V(G) do
        c(v) := ∅
        f(v) := 0
    end for
    while colored < p · |V(G)| do
        I := GreedyIndependentSet(G [{v ∈ V(G) : |c(v)| < p}], f)
        for all v ∈ I do
            c(v) := c(v) ∪ {color}
            f(v) := f(v) + 1
            colored := colored +1
        end for
    end while
    return (c, color)
end function
```

3.2 Zmodyfikowany algorytm DSATUR

Algorytm DSATUR należy do rodziny algorytmów sekwencyjnych. W klasycznym problemie, polega on na wyborze spośród niepokolorowanych wierzchołków zbioru niezależnego i pokolorowanie go na nowy kolor. Zespół postanowił nieco zmodyfikować ten algorytm i zbadać jego działanie dla problemu kolorowania wielowarstwowego.

Saturację w klasycznej wersji tego algorytmu traktowaliśmy jako liczbę unikalnych kolorów w sąsiednich pokolorowanych wierzchołkach. Aby algorytm działał dla wielu warstw, jednocześnie uwzględniając kolory znajdujące się u sąsiadów oraz w wierzchołku, należy zdefiniować nowy porządek.

W celu analizy problemu, możemy rozpatrywać saturację wielorako; zdefiniujmy:

- saturację zewnętrzną, będącą ilością unikalnych kolorów użytych do pokolorowania wierzchołków sąsiednich, tzn.
$$S_G^{OUT}(v) = \left| \bigcup_{u \in N_G(v)} c(u) \right|$$
- saturację wewnętrzną, będącą ilością kolorów użytych do pokolorowania wierzchołka, tzn.
$$S_G^{IN}(v) = |c(v)|$$
- saturację całkowitą, będącą ilością unikalnych kolorów użytych do pokolorowania wierzchołków sąsiednich, tzn.
$$S_G(v) = \left| \left(\bigcup_{u \in N_G(v)} c(u) \right) \cup c(v) \right|$$

Ponieważ $\forall u, v \in V(G) \quad u, v \in E(G) \implies c(u) \cap c(v) = \emptyset$ dla poprawnego p-warstwowego pokolorowania c grafu G, mamy $S_G(v) = S_G^{OUT}(v) + S_G^{IN}(v)$.

W każdym przypadku saturacją będzie saturacja całkowita i po niej będziemy w pierwszej kolejności sortować. W razie remisów wybieramy wierzchołek o większej saturacji wewnętrznej, gdyż później, gdy jego sąsiedzi mogą dostać kolejne kolory i będzie go ciężej pokolorować.

3.2.1 Pseudokod

Wejście: graf G

colored $:= 0$
max_color $:= 0$

function COLOR(G)
    $c := []$
    $s := []$

    for all $v \in V(G)$ **do**
        $c(v) = \varnothing$
        $s(v) = \varnothing$
    end for

    $v := rand(\{v \in V(G) : \forall u \in V(G) \deg_G v \geq \deg_G u\})$
    COLORVERTEX(v, &c, &s)

    while colored $< p \cdot |V(G)|$ **do**
        $v := rand(\{v \in V(G) : \forall u \in V(G) S_G v \geq S_G u\})$
        COLORVERTEX(v, &c, &s)
    end while

    return $(c,$ max_color$)$
end function

```
procedure ColorVertex(v, &c, &s)
    color := 1
    while color ∈ s(v) do
        color := color +1
    end while
    c(v) = c(v) ∪ {color}
    colored := colored +1

    if color > max_color then
        max_color := color
    end if

    s(v) = s(v) ∪ {color}
    for all u ∈ N_G(v) do
        s(v) = s(v) ∪ {color}
    end for
end procedure
```

3.3 Zmodyfikowany algorytm CS

Ostatnim algorytmem zaproponowanym przez zespół będzie zmodyfikowany algorytm podobny do algorytmu CS (*ang. connected sequential*). Jego modyfikacja będzie polegała na tym, że w przypadku konieczności użycia koloru, zostanie uruchomiona procedura wymiany, bazująca na algorytmie wyznaczania permutacji Fishera-Yatesa.

Algorytm zakłada, że w przypadku braku dostępnego koloru z puli, będziemy po kolei iść wzdłuż pewnej ścieżki od naszego wierzchołka i próbować wymieniać kolory do momentu, gdy któryś z wierzchołków będzie się dało pokolorować użytymi już kolorami, albo gdy algorytm zakończy permutację. Nowego koloru użyjemy tylko w drugim przypadku. Aby zwiększyć skuteczność, będziemy sprawdzać wymianę wszystkich możliwych kolorów.

3.3.1 Pseudokod

```
Wejście: graf G

colored := 0
max_color := 0

function Color(G)
    c := []
    for all v ∈ V(G) do
        c(v) = ∅
    end for

    for all v ∈ V(G), l ∈ {1, ..., p} do
        for all color ∈ {1, ..., max_color} do
            if color ∉ c[{v} ∪ N_G(v)] then
                c(v) := c(v) ∪ {color}
                break
            end if
        end for
        if ¬ColorInterchange(v, &G, &c) then
            c(v) := c(v) ∪ {color + 1}
            max_color := max_color +1
        end if
    end for
    return (c, max_color)
end function
```

procedure ColorInterchange(v, &G, &c)
  for all $i \in 1, ..., ANNEALING\_CONSTANT$ **do**
    $u := rand(N_G(v))$
    $X = c(u) \cap c[N_G(v) - \{u\}]$
    for all $x \in X$ **do**
      for all color $\in \{1, ...,$ max_color$\}$ **do**
        if color $\notin c\,[\{u\} \cup N_G(u)]$ **then**
          $c(v) := c(v) \cup \{x\}$
          $c(u) := c(u) - \{x\} \cup \{$color$\}$
          return $true$
        end if
      end for
    end for
    $v := u$
  end for
  return $false$
end procedure

4 Założenia techniczne

W wyborze języka, w którym zostanie stworzony program, zespół kierował się przejrzystością implementacji, aby móc skupić się na teoretycznej części problemu. Ostatecznie, program zostanie napisany w języku C# - duży wpływ miał tutaj fakt, że był to jeden z niewielu języków, który jest dobrze znany obu członkom zespołu. Jest to język stosujący paradygmat programowania obiektowego, co pozwoli nam potraktować grafy w bardziej obrazowy sposób podczas implementacji. Dodatkowo, posiada duże wsparcie społeczności oraz bindingi do bibliotek wizualizujących grafy (m.in. Graphviz). Zaletą jest również większa prędkość obliczeń na maszynie wirtualnej .NET w stosunku do języków interpretowanych.

Projekt będzie obejmował:

- interfejs użytkownika (parser wejścia, komunikaty itp.)
- implementację wyżej wymienionych algorytmów
- prezentację wyniku (wyjście tekstowe + wizualizacja za pomocą biblioteki graphviz)
- przykładowe benchmarki testujące skuteczność oraz czas wykonania na wybranych grafach

Literatura

[1] Marek Kubale, *Analiza efektywności algorytmów kolorowania grafów*, PTM 1980 `https://wydawnictwa.ptm.org.pl/index.php/matematyka-stosowana/article/viewFile/1531/1457` *(dostęp: 23 marca 2019)*

[2] Marek Kubale, *Optymalizacja dyskretna. Modele i metody kolorownia grafów*, WNT 2002

[3] Andrew Radin, *Graph coloring heuristics from investigation ofsmallest hard to color graphs*, RIT Scholar Works 2000 `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.867.178&rep=rep1&type=pdf` *(dostęp: 23 marca 2019)*