Algorytmy krawędziowego kolorowania acyklicznego wybranych klas grafów

Jakub Dziarkowski

Instytut Informatyki Analitycznej
Wydział Matematyki i Informatyki
Uniwersytet Jagielloński

Opiekun pracy: dr Iwona Cieślik

Abstrakt. Acykliczne kolorowanie krawędziowe grafu jest poprawnym kolorowaniem krawędziowym, w którym nie ma dwukolorowych cykli. Acyklicznym indeksem chromatycznym ($\chi'_a(G)$) nazywamy minimalną liczbę kolorów potrzebną do acyklicznego pokolorowania krawędzi. Istnieje hipoteza [ASZ01], według której $\Delta(G) \leqslant \chi'_a(G) \leqslant \Delta(G) + 2$. Alon i Zaks [AZ02] udowodnili, że problem wyznaczania acyklicznego indeksu chromatycznego jest NP-zupełny. Przedstawili również deterministyczny algorytm działający w czasie wielomianowym, kolorujący acyklicznie grafy o odpowiednio dużej tali (wielkości najmniejszego cyklu) na $\Delta(G) + 2$ kolory. W niniejszej pracy prezentujemy randomizowany algorytm o oczekiwanym wielomianowym czasie działania. Nasz algorytm również koloruje acyklicznie grafy o odpowiednio dużej tali, używając $\Delta(G) + 2$ kolorów, jednak wymaganie co do wielkości tali grafu zostało złagodzone.

Spis treści

ROZDZIAŁ 1

Wstęp

1. Zagadnienia, treść pracy

Acykliczne kolorowanie krawędziowe, podobnie jak standardowe kolorowanie krawędziowe, posiada szeroki zakres zastosowań. Przykłady obejmują problem optymalizacji trasowania i przypisywania długości fali w sieciach komputerowych [NG23], a także rozwiązanie dużych problemów optymalizacyjnych opartych na macierzy Hessego [Geb+09]. Jednym z pierwszych artykułów dotyczących acyklicznego kolorowania krawędziowego jest praca Grünbauma z 1973 roku [Grü73]. W tym artykule autor definiuje kolorowanie acykliczne nieco inaczej niż my, ponieważ nie wymaga, aby kolorowanie acykliczne spełniało warunek poprawnego kolorowania. Mimo, że ogólny problem optymalnego acyklicznego kolorowania jest trudny, udaje się regularnie poprawiać znane wyniki w szczególnych przypadkach, chociażby dla określonych klas grafów lub dopuszczając możliwość użycia większej niż optymalna liczby kolorów.

Niniejsza praca skupia się na algorytmicznym podejściu do problemu acyklicznego kolorowania grafów. W pierwszej części przedstawiamy niezbędne definicje i pojęcia, by następnie zaprezentować dowód twierdzenia o NP-zupełności ogólnego problemu znajdowania optymalnego acyklicznego kolorowania krawędziowego [AZ02]. Centralnym punktem tej pracy jest szczegółowy opis algorytmu acyklicznego kolorowania krawędziowego, który korzysta z co najwyżej dwóch dodatkowych kolorów i działa przy założeniu o ograniczonej wielkości talii grafu. Nasze proponowane rozwiązanie opiera się na randomizowanym algorytmie, który działa w czasie wielomianowym. Ten algorytm jest w pełni inspirowany pracą Noga Alon'a i Ayal Zaks'a [AZ02]. Jedyną różnicą między oboma algorytmami jest to, że ich wersja jest deterministyczna, ale w założeniach o wielkości talii grafu występuje stała rzędu 10^8, którą nam udało się zbić do $4e$ (gdzie e jest liczbą Eulera). To osiągnięcie wiązało się jednak z przekształceniem deterministycznego algorytmu w wersję randomizowaną.

2. Podstawowe definicje i twierdzenia

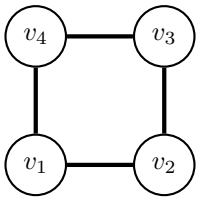
Ważnym punktem tej pracy jest dostarczenie wartości edukacyjnej i uniknięcie jakichkolwiek niejasności dotyczących omawianych zagadnień. W tym celu przedstawimy definicje wszystkich kluczowych obiektów i pojęć, których będziemy używać, rozpoczynając od definicji grafu.

Definicja 1.1. Grafem G będziemy nazywać uporządkowaną parę (V, E), gdzie:

- V jest zbiorem **wierzchołków** (nazywanych również czasem węzłami lub punktami).

- $E \subseteq \{\{x, y\} \mid x, y \in V \ \wedge \ x \neq y\}$ jest zbiorem **krawędzi**. Zawiera on pary nieuporządkowane wierzchołków z V, każdą taką parę w zbiorze E będziemy nazywać krawędzią.

Zbiór wierzchołków grafu G oznaczamy jako $V(G)$, analogicznie zbiór krawędzi grafu G jako $E(G)$.
Oczywiście istnieją też inne definicje, np. takie, które dopuszczają krawędź między wierzchołkiem a samym sobą, wielokrotne krawędzie między wierzchołkami, lub krawędzie skierowane (wtedy E jest zbiorem par uporządkowanych).
Graf spełniający przedstawioną przez nas definicje jest często nazywany **nieskierowanym grafem prostym**. Wszystkie omawiane przez nas grafy będą właśnie nieskierowanymi grafami prostymi, z dodatkowym założeniem o skończoności. Oznacza to, że przyjmujemy $|V(G)| \in \mathbb{N}$.
Użyteczną wizualizacją grafu jest przedstawienie go na płaszczyźnie, w taki sposób, że wierzchołki utożsamiamy z punktami, za to krawędzie z odcinkami łączącymi te punkty. Poniższy rysunek przedstawia graf $(\{v_1, v_2, v_3, v_4\}, \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_1\}\})$.

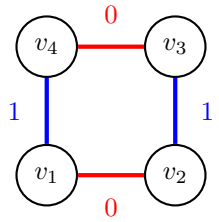


Znając pojęcie grafu, możemy przejść do jego kolorowania, a dokładniej do kolorowania krawędziowego.

Definicja 1.2. Kolorowaniem krawędziowym grafu G określamy funkcję $c : E(G) \to \mathbb{N}$. Kolorowanie krawędziowe nazywamy **poprawnym** jeżeli nie istnieje para sąsiadujących krawędzi pokolorowana na ten sam kolor. Formalnie:

$$c \text{ poprawne } \Leftrightarrow \left(\forall_{e_1 \neq e_2 \in E(G)} : c(e_1) = c(e_2) \Rightarrow e_1 \cap e_2 = \emptyset\right)$$

Wróćmy teraz do naszej wizualizacji grafu. Możemy na niej przedstawić kolorowanie krawędziowe poprzez faktyczne pokolorowanie odcinków reprezentujących krawędzie.



Rysunek przedstawia kolorowanie $c(\{v_1, v_2\}) = 0$, $c(\{v_2, v_3\}) = 1$, $c(\{v_3, v_4\}) = 0$, $c(\{v_4, v_1\}) = 1$. Jest to jednocześnie poprawne kolorowanie krawędziowe tego grafu.

Zauważmy, że samo pokolorowanie krawędzi grafu, a nawet poprawne pokolorowanie krawędzi grafu nie jest trudne. W tym pierwszym przypadku możemy każdej krawędzi przypisać ten sam kolor, a w drugim każda krawędź może dostać inny kolor (uzyskamy w ten sposób kolorowanie grafu G na $E(G)$ kolorów). Nietrywialnym problemem, z jakim często się mierzymy w teorii grafów, jest kolorowanie jak najmniejszą liczbą kolorów.

Definicja 1.3. Indeksem chromatycznym grafu G nazywamy najmniejszą liczbę naturalną k taką, że istnieje poprawne kolorowanie krawędziowe grafu G na k kolorów. Czyli $\exists_c : (c : E(G) \to \{0, ..., k-1\} \wedge c$ jest poprawne). Indeks chromatyczny grafu G oznaczmy przez $\chi'(G)$.

Definicja 1.4. Niech wierzchołek v należy do $V(G)$, wtedy liczbę krawędzi tego grafu, do których należy v (liczbę krawędzi wychodzących z wierzchołka) określamy mianem **stopnia wierzchołka**. Stopień wierzchołka v oznaczamy przez $deg(v)$. Formalnie $deg(v) = |\{e \in E(G) \mid v \in e\}|$.

Definicja 1.5. Stopniem grafu G będziemy nazywać maksymalny stopień wierzchołka w tym grafie. Stopień grafu G oznaczamy przez $\Delta(G)$.
Formalnie $\Delta(G) = max\{deg(v) \ : \ v \in V(G)\}$.

Wprowadzenie pojęcia stopnia grafu stanowi naturalny krok w rozmowie o indeksie chromatycznym. Warto zauważyć, że warunkiem koniecznym, aby kolorowanie krawędziowe było poprawne, jest pokolorowanie różnymi kolorami wszystkich krawędzi wychodzących z wierzchołka v, dla każdego wierzchołka v w grafie. To oznacza, że wartość indeksu chromatycznego możemy ograniczyć od dołu przez stopień grafu: $\Delta(G) \leqslant \chi'(G)$. Przykładem grafu, dla którego zachodzi $\chi'(G) = \Delta(G)$ jest ilustrowany wyżej graf o 4 wierzchołkach, 4 krawędziach i stopniu grafu równym 2. Ograniczenie górne na wartość indeksu chromatycznego znamy z poniższego twierdzenia.

Twierdzenie 1.6. *(**Twierdzenie Vizinga**) [Viz65] Dla każdego nieskierowanego grafu prostego G o stopniu $\Delta(G)$ zachodzi:*

$$\Delta(G) \leqslant \chi'(G) \leqslant \Delta(G) + 1$$

Czyli dla każdego grafu G wiemy, że zachodzi $\chi'(G) = \Delta(G)$ lub $\chi'(G) = \Delta(G)+1$. Okazuje się jednak, że dokładne wyznaczenie indeksu chromatycznego dla dowolnego grafu jest problemem NP-zupełnym.

Twierdzenie 1.7. *[Hol81] Problem wyznaczenia indeksu chromatycznego dla dowolnego grafu jest NP-zupełny. Problem ograniczony do wyznaczania indeksu chromatycznego dowolnego grafu kubicznego (takiego, że stopień każdego wierzchołka w grafie jest równy 3) pozostaje NP-zupełny.*

Definicja 1.8. Ścieżką w grafie G nazywamy taki ciąg wierzchołków $(v_1, ..., v_k)$, że $\forall_{1 \leqslant i \leqslant k} : v_i \in V(G)$ oraz $\forall_{1 \leqslant i < k} : \{v_i, v_{i+1}\} \in E(G)$.

Definicja 1.9. Długość ścieżki $(v_1, ..., v_k)$ wynosi $k - 1$.

Definicja 1.10. Odległość między wierzchołkami u oraz v w grafie G, definujemy jako długość najkrótszej ścieżki postaci $(u, ..., v)$ w grafie G. Odległość między wierzchołkami u, v oznaczamy $d(u, v)$. Jeżeli nie istnieje ścieżka $(u, ..., v)$ w grafie G, to $d(u, v) = \infty$.

Definicja 1.11. Cyklem w grafie G nazywamy taką ścieżkę $(v_1, ..., v_k)$ w grafie G, że $k > 2$ oraz $v_1 = v_k$. **Cykl prosty** to taki cykl, w którym tylko wierzchołek v_1 występuje dokładnie dwa razy, a pozostałe wierzchołki grafu występują na nim co najwyżej raz.

Definicja 1.12. Długość cyklu $(v_1, ..., v_k)$ wynosi $k - 1$.

Zdefiniowaliśmy ścieżkę i cykl jako ciąg wierzchołków, warto jednak mieć na uwadze, że można te pojęcia równoważnie zdefiniować jako ciągi krawędzi. Używając sformułowania krawędzie ścieżki lub krawędzie cyklu, będziemy mówili o krawędziach zawierających kolejne pary wierzchołków (łączących te wierzchołki).
W dalszej części pracy zawsze pisząc cykl będziemy mieli na myśli cykl prosty.
W końcu jesteśmy gotowi do zdefiniowania najważniejszego dla nas pojęcia, czyli acyklicznego kolorowania krawędziowego.

Definicja 1.13. Weźmy dowolny graf G i jego poprawne kolorowanie krawędziowe $c : E(G) \to \mathbb{N}$. Jeżeli dla każdego cyklu występującego w G jego krawędzie są pokolorowane na co najmniej trzy różne kolory to c jest **acyklicznym kolorowaniem krawędziowym** grafu G. Równoważnie mówimy, że graf G z kolorowaniem c nie zawiera dwukolorowych cykli. Warunek acyklicznego kolorowania krawędziowego możemy zapisać formalnie w następujący sposób:

Jeżeli $(v_1, ..., v_k)$ jest cyklem w grafie G, to $|\{c(\{v_i, v_{i+1}\}) : 1 \leqslant i < k\}| > 2$.

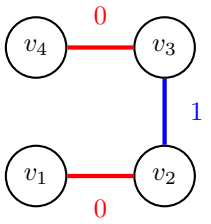
Dla kolorowania krawędziowego definiowaliśmy indeks chromatyczny, analogicznie dla acyklicznego kolorowania krawędziowego definujemy acykliczny indeks chromatyczny.

Definicja 1.14. Acyklicznym indeksem chromatycznym grafu G nazywamy najmniejszą liczbę naturalną k taką, że istnieje acykliczne kolorowanie krawędziowe grafu G na k kolorów. Czyli $\exists_c : (c : E(G) \to \{0, ..., k-1\} \wedge c$ jest acyklicznym kolorowaniem krawędziowym). Acykliczny indeks chromatyczny grafu G oznaczamy przez $\chi'_a(G)$.

Skoro każde acykliczne kolorowanie krawędziowe musi być poprawnym kolorowaniem krawędziowym to możemy ograniczyć od dołu wartość acyklicznego indeksu chromatycznego przez:

$$\Delta(G) \leqslant \chi'(G) \leqslant \chi'_a(G)$$

Przykład grafu, dla którego zachodzi $\Delta(G) = \chi'(G) = \chi'_a(G)$:



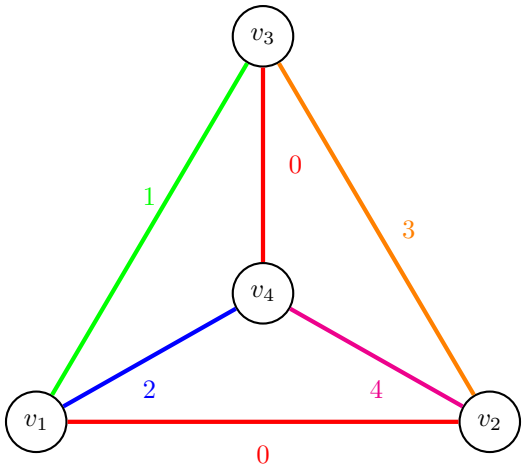
W grafach takich jak ten na rysunku, gdzie w ogóle nie występuje cykl, każde poprawne kolorowanie krawędziowe będzie jednocześnie acyklicznym kolorowaniem

krawędziowym.

Definicja 1.15. Graf, w którym każde dwa wierzchołki są połączone krawędzią nazywamy **kliką**. Klikę o n wierzchołkach oznaczamy jako K_n. Dla kliki zachodzi:

$$\forall_{u,v \in V(K_n)} : \{u, v\} \in E(K_n)$$

Klika z czterama wierzchołkami K_4 jest przykładem grafu, dla którego zachodzi $\chi'_a(K_4) = \Delta(K_4) + 2$. Dokładniej to $\Delta(K_4) = 3$, a potrzebujemy co najmniej 5 kolorów, by uzyskać acykliczne kolorowanie krawędziowe K_4.



Dowód
Załóżmy nie wprost, że $\chi'_a(K_4) < 5$. Bez straty ogólności możemy przyjąć:

- $c(\{v_1, v_2\}) = 0$
- $c(\{v_1, v_3\}) = 1$
- $c(\{v_1, v_4\}) = 2$

Czyli krawędzie wychodzące z v_1 mają takie kolory jak na rysunku.

1° Krawędzie wychodzące z v_3 są kolorów $0, 1, 2$.
Wtedy jedyna opcja jaką mamy by uzyskać poprawne kolorowanie to przypisanie:

- $c(\{v_3, v_2\}) = 2$
- $c(\{v_3, v_4\}) = 0$

Jednak wtedy powstaje cykl dwukolorowy $(v_1, v_4, v_3, v_2, v_1)$ z krawędziami o kolorach kolejno $(2, 0, 2, 0)$.

2° Z v_3 wychodzi krawędź o kolorze spoza zbioru $\{0, 1, 2\}$.
Teraz bez straty ogólności możemy przyjąć, że

- $c(\{v_3, v_2\}) = 3$

Jednak skoro mamy do dyspozycji tylko 4 kolory, a krawędź $\{v_3, v_4\}$ nie może być już w kolorach $1, 2, 3$ aby uzyskać poprawne kolorowanie, otrzymujemy:

- $c(\{v_3, v_4\}) = 0$

Analogicznie krawędź $\{v_2, v_4\}$ nie może być już w kolorach $0, 2, 3$ aby uzyskać poprawne kolorowanie, więc:

- $c(\{v_2, v_4\}) = 1$

ale w ten sposób uzyskaliśmy cykl dwukolorowy $(v_1, v_3, v_4, v_2, v_1)$ z krawędziami o kolorach kolejno $(1, 0, 1, 0)$.
Uzyskana sprzeczność dowodzi, że $\chi'_a(K_4) \geqslant 5$, co w połączeniu z przykładem acyklicznego kolorowania krawędziowego, takiego jak na rysunku, wykorzystującego 5 kolorów daje $\chi'_a(K_4) = 5$.

■

Podejrzewamy, że $\Delta(G) + 2$ jest optymalnym ograniczeniem górnym na wartość acyklicznego indeksu chromatycznego, jednak jak na razie nie udało się tego udowodnić.

Hipoteza 1.16. *(Acyclic Edge Coloring Conjecture - AECC) [ASZ01] Dla każdego nieskierowanego grafu prostego G o stopniu $\Delta(G)$ zachodzi:*

$$\Delta(G) \leqslant \chi'_a(G) \leqslant \Delta(G) + 2$$

Powyższa hipoteza została udowodniona dla wielu szczególnych klas grafów, np.:

- Grafy planarne niezawierające cykli długości 5. [SWW12]
- Grafy spełniające $mad(G) < 4$, gdzie $mad(G)$ jest maksymalnym średnim stopniem grafu G (dokładna definicja znajduje się w cytowanej pracy). [WZ14]
- Grafy planarne, w których każdy cykl długości 3 jest rozłączny krawędziowo z każdym cyklem długości 4. [WSW13]
- Grafy subkubiczne (spełniające $\Delta(G) \leqslant 3$). [AMM12]
- Grafy, dla których zachodzi $g(G) \geqslant 4e(\Delta(G))^3$, gdzie $g(G)$ jest talią grafu (definicja 2.1), a e jest liczbą Eulera. Jest to rezultat osiągnięty przez nas (twierdzenie 2.3) w wyniku poprawy wartości stałej z twierdzenia 2.2.

3. NP-zupełność

Twierdzenie 1.17. *[AZ02] Problem wyznaczenia acyklicznego indeksu chromatycznego dla dowolnego grafu jest NP-zupełny.*

Przedstawiony poniżej dowód pochodzi z tej samej pracy, co twierdzenie.

Lemat 1.18. *Dla grafu F z Rysunku 1. zachodzi:*

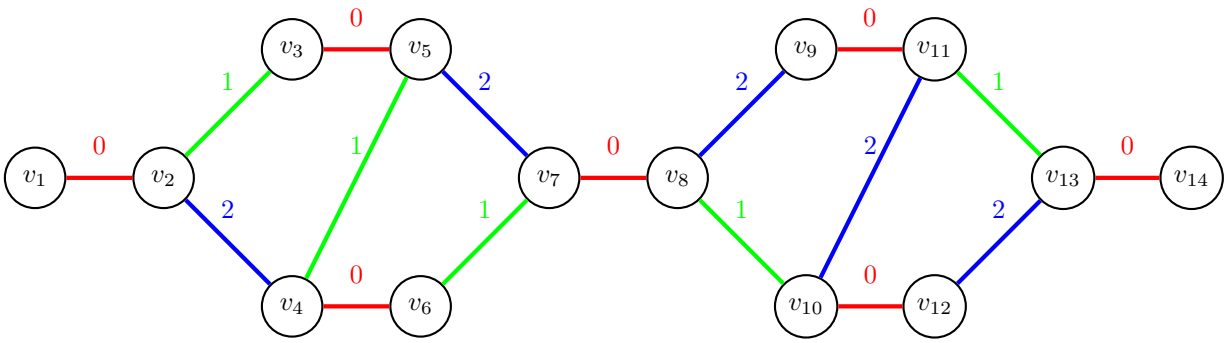
(1) *Każde acykliczne 3-kolorowanie F przypisuje ten sam kolor krawędziom (v_1, v_2) i (v_{13}, v_{14}).*
(2) *Istnieje acykliczne 3-kolorowanie F bez dwukolorowej ścieżki między v_1 i v_{14}.*

Dowód

Definicja 1.19. Podgrafem indukowanym grafu G nazywamy graf powstały przez usunięcie z grafu G pewnej liczby wierzchołków oraz wszystkich wychodzących z nich i wchodzących do nich krawędzi. Weźmy $V' \subseteq V(G)$. Niech $E' = \{\{u, v\} \in E(G) \mid u \in V' \wedge v \in V'\}$. Wtedy $G' = (V', E')$ jest podgrafem indukowanym, grafu G, o wierzchołkach V'.

Przykład kolorowania realizującego punkt (2) możemy zobaczyć na poniższym rysunku 1.
Pokażmy, że dla podgrafu indukowanego F' grafu F o wierzchołkach $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$, każde acykliczne 3-kolorowanie F' przypisuje ten sam kolor krawędziom (v_1, v_2) i (v_7, v_8). Jeżeli c jest funkcją kolorującą to możemy bez straty ogólności założyć, że $c(v_1, v_2) = 0$, $c(v_2, v_3) = 1$, $c(v_2, v_4) = 2$. Takie kolorowanie jednoznacznie wymusza pokolorowanie pozostałych krawędzi dokładnie tak jak na poniższym rysunku. W przeciwnym wypadku powstałby cykl dwukolorowy, np. jeżeli ustalilibyśmy $c(v_3, v_5) = 2$, to powstałby cykl dwukolorowy $(v_2, v_3, v_5, v_4, v_2)$ lub $(v_2, v_3, v_5, v_7, v_6, v_4, v_2)$.
Powtarzając powyższe rozumowanie dla podgrafu indukowanego F'' grafu F o wierzchołkach $\{v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}\}$ otrzymujemy, że każde acykliczne 3-kolorowanie F'' przypisuje ten sam kolor krawędziom (v_7, v_8) i (v_{13}, v_{14}).
Łącząc obie powyżej udowodnione własności dostajemy punkt (1) lematu.

■



Rysunek 1. Graf F z acyklicznym 3-kolorowaniem krawędziowym

Dowód twierdzenia 1.17
Pokażemy, że stwierdzenie czy dla danego grafu G zachodzi $\chi'_a(G) \leqslant 3$ jest problemem NP-zupełnym.
Z twierdzenia 1.7 wiemy, że problem znalezienia indeksu chromatycznego dowolnego grafu kubicznego jest NP-zupełny. Korzystając z tego rezultatu przeprowadzimy redukcję do naszego problemu.
Niech G będzie instancją problemu znalezienia indeksu chromatycznego grafu G przy $\Delta(G) = 3$. Skonstruujmy graf G' zamieniając każdą krawędź $(u, v) \in E(G)$ na kopię grafu F z rysunku 1 w taki sposób, że $u = v_1$ i $v = v_{14}$. Nazwijmy ten graf $F_{u,v}$. Teraz pokażemy, że $\chi'_a(G') = 3 \Leftrightarrow \chi'(G) = 3$.
$\Rightarrow$) $\chi'_a(G') = 3$
W tym przypadku wystarczy krawędź (u, v) w G pokolorować na ten sam kolor co krawędzie (v_1, v_2) i (v_{13}, v_{14}) w grafie $F_{u,v} \subseteq G'$. Z udowodnionego wcześniej lematu wiemy, że jest to ten sam kolor.
$\Leftarrow$) $\chi'(G) = 3$
Postępujemy analogicznie do pierwszego przypadku: graf $F_{u,v}$ 3-kolorujemy acyklicznie tak, żeby kolor (v_1, v_2) i (v_{13}, v_{14}) był taki sam jak kolor (u, v). Korzystając z lematu wiemy, że możemy tak pokolorować.

■

ROZDZIAŁ 2

Algorytm dla grafów o dużej talii

1. Motywacja

Tak jak wspominaliśmy we wstępie, zagadnienie acyklicznego kolorowania krawędziowego jest chętnie podejmowanym tematem badań naukowych. Jest to problem ciekawy zarówno pod kątem teoretycznym jak i algorytmicznym, dlatego też można znaleźć dziesiątki prac dotyczących acyklicznego indeksu chromatycznego, wśród których są zarówno takie zawierające dowody konstruktywne jak i niekonstruktywne.
Głównym obszarem naszych zainteresowań jest podejście algorytmiczne, ze szczególnym uwzględnieniem algorytmów z realnym zastosowaniem, takich które faktycznie można zaimplementować. Nieciężko trafić na dowody konstruktywne, które są takie z formalnego punktu widzenia, jednak mnogość przypadków i wyjątków skutecznie zniechęcają do implementacji takiego algorytmu.
Jak przekonamy się zaraz dowód twierdzenia o acyklicznym indeksie chromatycznym grafów o dużej talii, generuje niezwykle elegancki algorytm takiego kolorowania, dlatego też postanowiliśmy się nim zająć w tej pracy.

2. Twierdzenia

Naszą główną inspiracją była praca Alon'a i Zaks'a, pt. „Algorithmic Aspects of Acyclic Edge Colorings", z której to właśnie pochodzi twierdzenie 2.2.

Definicja 2.1. Dla grafów zawierających przynajmniej jeden cykl definujemy **talię** grafu jako długość najkrótszego cyklu zawartego w grafie. Czasami przyjmuje się, że dla grafu acyklicznego (niezawierającego cykli) talia wynosi nieskończoność.

Twierdzenie 2.2. *[AZ02] Jeżeli* $g(G) > c\,(\Delta(G))^3$*, gdzie* $g(G)$ *jest talią grafu* G*, a* c *jest pewną stałą* ($c \approx 10^8$)*, to* $\chi'_a(G) \leqslant \Delta(G) + 2$ *i istnieje deterministyczny algorytm działający w czasie wielomianowym, który znajduje acykliczne kolorowanie krawędziowe grafu* G *na* $\Delta(G) + 2$ *kolorów.*

Idea dowodu

(1) Znajdujemy poprawne kolorowanie krawędziowe G na $\Delta(G) + 1$ kolorów, algorytmem z [MG92].
(2) Znajdujemy wszystkie cykle dwukolorowe $C_1, ..., C_b$ (autorzy pracy nie podają w jaki sposób).
(3) Znajdujemy zbiory $E_1, ..., E_b$, algorytmem znajdowania maksymalnego przepływu.
(4) Tworzymy grafy H i $\bar{H}$.
(5) Znajdujemy zbiór niezależny w $\bar{H}$, stosując Algorytmiczny Lokalny Lemat Lovász'a w wersji udowodnionej przez Beck'a w 1991 roku [Bec91].

(6) Przekolorowujemy elementy w zbiorze S na nowy kolor, uzyskując acykliczne kolorowanie krawędziowe na $\Delta(G)+2$ kolorów.

Przedstawiony przez nas algorytm różni się tak naprawdę tylko w kroku (5) - też używamy Algorytmicznego Lokalnego Lematu Lovász'a, tylko w innej wersji. W szczególności oznacza to, że dokładne definicje wspomnianych w idei dowodu obiektów, takich jak zbiory $E_1, ..., E_b$ czy grafy H i $\bar{H}$ znajdują się w dalszej części pracy, wraz ze szczegółowymi opisami metod wyznaczania ich.
Dowód twierdzenia 2.2 składa się z kilku kroków, a każdy krok można de facto traktować jako niezależny od reszty algorytm, pełniący pewną szczególną funkcję. Takie podejście całkowitej izolacji kolejnych kroków algorytmu przydaje się zwłaszcza przy implementacji i umożliwia testowania każdej funkcjonalności oddzielnie.
Niestety stosunkowo wysoka stała zawarta w warunkach twierdzenia (rzędu 10^8) sprawia, że ciężko znaleźć zastosowanie (oraz dane wejściowe, dla których działanie programu byłoby szybkie) dla tego algorytmu.
Fragmentem algorytmu, który wymusza tak wysokie dolne ograniczenie na wielkość talli grafu jest zastosowanie Algorytmicznego Lokalnego Lematu Lovász'a w wersji udowodnionej przez Beck'a w 1991 roku [Bec91]. Nie podajemy tutaj dokładnej treści twierdzenia ze względu na istnienie wielu wersji Lokalnego Lematu Lovász'a i Algorytmicznego Lokalnego Lematu Lovász'a. W dalszej części pracy pojawia się to twierdzenie w wersji wykorzystanej przez nas. Od 1991 roku poczyniono wiele postępów związanych z Algorytmicznym Lokalnym Lematem Lovász'a, co pozwoliło nam sformułować poniższe twierdzenie.

Twierdzenie 2.3. *Jeżeli $g(G) \geqslant 4e\,(\Delta(G))^3$, gdzie $g(G)$ jest talią grafu G,a e jest liczbą Eulera, to $\chi'_a(G) \leqslant \Delta(G)+2$ i istnieje randomizowany algorytm o wielomianowym oczekiwanym czasie działania, który znajduje acykliczne kolorowanie krawędziowe grafu G na $\Delta(G)+2$ kolorów.*

3. Dowód twierdzenia 2.3, pseudokod

Dowód przebiega w ten sam sposób, co dowód twierdzenia 2.2 w [AZ02], tak jak wspomnieliśmy wyżej, jedyna różnica opiera się na sposobie zastosowania Algorytmicznego Lokalnego Lematu Lovász'a. W naszej pracy znajduje się jednak dużo więcej szczegółów, zwłaszcza takich, które mają za zadanie ułatwić samodzielną implementację algorytmu.

Dowód

3.1. $\Delta(G)$ i $g(G)$.

Dany jest graf G, o którym wiemy, że:

$$g(G) \geqslant 4e\,(\Delta(G))^3 \tag{1}$$

Chcemy uzyskać acykliczne kolorowanie krawędziowe

$$c_a : E(G) \to \{0, 1, ..., \Delta(G)+1\}.$$

Choć jesteśmy świadomi zachodzenia nierówności (1), to nie dysponujemy informacjami na temat wartości $\Delta(G)$ i $g(G)$. Jako, że wyznaczenie stopnia grafu jest nieco łatwiejsze niż wyznaczenie talii, to zaczniemy właśnie od tego.

Przyjmijmy, że $V(G) = N$ i wierzchołki grafu G są ponumerowanie kolejnymi liczbami naturalnymi, czyli $V(G) = \{0, 1, ..., N-1\}$. Niech $G[i]$ będzie listą sąsiadów wierzchołka i, gdzie $0 \leqslant i < N$. Przez listę sąsiadów rozumiemy listę wierzchołków, które są połączone z i krawędzią w grafie G.
Zauważmy, że $deg(i) = |G[i]|$, a zatem $\Delta(G) = max\{|G[i]| \ : \ 0 \leqslant i < N\}$.

Algorithm 1 Calculate $\Delta(G)$

```
procedure DELTA(G)
    Δ ← 0
    for i ← 0, N − 1 do
        Δ ← max(Δ, G[i].size)
    end for
    return Δ
end procedure
```

W celu wyznaczenia długości najkrótszego cyklu w grafie wykonamy procedurę BFS (Breadth-first search) dla każdego wierzchołka grafu G jako źródła. Algorytm DFS (Depth-first search) okazuje się nie być pomocny w tym problemie. Rozważmy moment w trakcie wykonywania przeglądu BFS od korzenia (źródła) r, w którym przeglądamy sąsiadów wierzchołka v. Niech u będzie dowolnym sąsiadem v, wtedy mamy następujące możliwości:

(1) u jest rodzicem v w drzewie przeglądu BFS.
(2) u po raz pierwszy występuje w przeglądzie BFS, wtedy v zostanie jego rodzicem.
(3) u nie jest rodzicem v i v nie jest rodzicem u w drzewie BFS (czyli widzieliśmy już u podczas przeglądania grafu), $d(r, v) \leqslant d(r, u) \leqslant d(r, v) + 1$ oraz istnieje cykl (może nie być cyklem prostym) przechodzący przez wierzchołki r, u, v o długości $d(r, v) + 1 + d(r, u)$.

Najciekawszy jest dla nas oczywiście punkt trzeci, bo mówi o cyklu o konkretnej długości. To, że znalezione przez nas cykle mogą nie być cyklami prostymi nie jest problemem. Jeżeli r jest korzeniem BFS i znaleźliśmy cykl składający się ze ścieżki z r do v, krawędzi $\{v, u\}$ i ścieżki z r do u, to możemy rozważyć wierzchołek x będący ostatnim wspólnym wierzchołkiem na obu tych ścieżkach. Wtedy cykl złożony ze ścieżki z x do v, krawędzi $\{v, u\}$ i ścieżki z x do u jest cyklem prostym, więc znajdziemy ten lub jeszcze krótszy cykl w trakcie wykonywania BFS od źródła x.
Zauważmy, że gdy najkrótszy cykl prosty przechodzący przez r ma długość k, dla pewnego $k \in \mathbb{N}$, to w przeglądzie BFS od r znajdziemy takie wierzchołki u i v, że $d(r, u) = \lfloor \frac{k-1}{2} \rfloor$, $d(r, v) = \lceil \frac{k-1}{2} \rceil$, $\{u, v\} \in E(G)$ i jedynym wierzchołkiem należącym do najkrótszej ścieżki z r do u i najkrótszej ścieżki z r do v jest r.

Algorithm 2 Find length of shortest cycle with vertex r in G

```
procedure GIRTH_BFS(G, r)
    g ← ∞
    Q ← empty queue
    Q.push(r)
    d(r, r) ← 0
    while Q not empty do
        v ← Q.front
        for all u ∈ G[v] \ parent(v) do
            if d(r, u) = ∞ then
                parent(u) ← v
                d(r, u) ← d(r, v) + 1
                Q.push(u)
            else
                g ← min(g, d(r, u) + d(r, v) + 1)
            end if
        end for
    end while
    return g
end procedure
```

Algorithm 3 Calculate $g(G)$

```
procedure GIRTH(G)
    g ← ∞
    for i ← 0, N − 1 do
        g ← min(g,GIRTH_BFS(G, i))
    end for
    return g
end procedure
```

3.2. $c : E(G) \to \{0, 1, ..., \Delta(G)\}$**.**

Na tym etapie chcemy znaleźć funkcję c będącą poprawnym kolorowaniem krawędziowym grafu G na $\Delta(G) + 1$ kolorów. Wiemy, że takie kolorowanie istnieje dzięki twierdzeniu 1.6. Do znalezienia funkcji kolorującej wykorzystamy algorytm Misra i Gries'a [MG92], który działa w czasie wielomianowym.
Do opisania działania algorytmu potrzebujemy wprowadzić poniższe definicje.

Definicja 2.4. Kolor $c_1 \in \mathbb{N}$ nazywamy **wolnym** na wierzchołku $v \in V(G)$ jeżeli żadna krawędź wychodząca z tego wierzchołka nie jest pokolorowana na c_1.

Definicja 2.5. Wachlarzem wierzchołka u nazywamy ciąg wierzchołków $(v_1, ..., v_k)$ spełniający następujące warunki:

- Ciąg jest niepusty.
- Każdy wierzchołek w ciągu występuje co najwyżej raz i jest sąsiadem u.
- Kolor krawędzi $\{u, v_{i+1}\}$ jest wolny na v_i, dla $1 \leqslant i < k$.

W algorytmie będziemy wykonywać operację rotacji wachlarza, przebiega ona następująco:

(1) $c(v_i, u) \leftarrow c(v_{i+1}, u)$, dla każdego $1 \leqslant i < k$.
(2) Odkoloruj krawędź $c(v_k, u)$.

Z własności wachlarza wiemy, że przed rotacją $c(v_{i+1}, u)$ był wolny na v_i, więc rotacja wachlarza nie psuje poprawności kolorowania.

Definicja 2.6. Niech $u \in V(G)$ będzie wierzchołkiem, a $a, b \in \mathbb{N}$ pewnymi kolorami. Najdłuższą ścieżkę przechodzącą przez u oraz składającą się wyłącznie z krawędzi w kolorach a i b nazywamy ab_u**-ścieżką**.

Operacja odwrócenia ab_u-ścieżki polega na zamianie kolorów krawędzi na tej ścieżce z a na b i z b na a. Dzięki temu, że ab_u-ścieżka jest maksymalną ścieżką w tych kolorach, odwrócenie ab_u-ścieżki również nie psuje poprawności kolorowania.
Używany przez nas algorytm podczas każdej iteracji przypisuje kolor jednej z krawędzi. W trakcie każdego kroku wybierana jest dowolna krawędź, która jeszcze nie została pokolorowana. Następnie znajdujemy maksymalny wachlarz, którego początkową krawędzią jest właśnie ta niepokolorowana. W kolejnym etapie przeprowadzamy jedno odwrócenie ścieżki oraz jedna rotacja wachlarza. Te operacje realizowane są w sposób zapewniający dostępność któregoś z kolorów dla omawianej krawędzi.

Poniżej prezentujemy pseudokod wszystkich funkcji pomocniczych wykorzystywanych w algorytmie. W celu pominięcia tych szczegółów można przejść od razu do lektury pseudokodu głównej części algorytmu Misra i Gries'a - Algorithm 9.

Algorithm 4 Check whether color c_1 is free on v

```
procedure IS_FREE(G, c, v, c_1)
    for all u ∈ G[v] do
        if c(v, u) = c_1 then
            return false
        end if
    end for
    return true
end procedure
```

Algorithm 5 Find smallest free color on v

```
procedure SMALLEST_FREE(G, c, v)
    for i ← 0, 1, ... do
        if IS_FREE(G, c, v, i) then
            return i
        end if
    end for
end procedure
```

Algorithm 6 Generate maximal fan of u with v being it's first vertex

```
procedure MAX_FAN(G, c, u, v)
    fan.push(v)
    do
        for all t ∈ (G[u] \ fan) such {u, t} is colored do
            if IS_FREE(G, c, fan.back, c(u, t)) then
                fan.push(t)
                break
            end if
        end for
    while fan enlarged in this iteration
    return fan
end procedure
```

Algorithm 7 Invert ab_u-path assuming u is at the end of this path

```
procedure INVERT_PATH(G, c, a, b, u)
    if IS_FREE(G, c, u, a) then
        next ← b
        prev ← a
    else
        next ← a
        prev ← b
    end if
    v ← u
    path.push(u)
    do
        for all t ∈ G[v] do
            if c(v, t) = next then
                swap(next, prev)
                path.push(t)
                v ← t
                break
            end if
        end for
    while path grown in this iteration
    swap colors on path
end procedure
```

Algorithm 8 Rotate fan of u

```
procedure ROTATE_FAN(c, u, fan)
    for i ← 0, fan.size − 2 do
        c(u, fan[i]) ← c(u, fan[i + 1])
    end for
    uncolor {u, fan.back}
end procedure
```

Algorithm 9 Find proper edge coloring $c : E(G) \to \{0, 1, ..., \Delta(G)\}$

```
procedure MISRA&GRIES(G)
    for all e ∈ E(G) do
        c(e) ← null
    end for

    Q ← E(G)
    while Q not empty do
        {u, v} ← Q.front                ▷ edge {u, v} will get color in this iteration
        fan ← MAX_FAN(G, c, u, v)

        a ← SMALLEST_FREE(G, c, u)                          ▷ 0 ⩽ a ⩽ Δ(G) − 1
        b ← SMALLEST_FREE(G, c, fan.back)                   ▷ 0 ⩽ b ⩽ Δ(G)

        INVERT_PATH(G, c, a, b, u)          ▷ After inverting b is free on u
        fan' ← empty

        for all x ∈ fan do
            fan'.push(x)
            if IS_FREE(G, c, x, b) then                     ▷ b is free on u and w
                w ← x
                break
            end if
        end for

        ROTATE_FAN(c, u, fan')
        c(u, w) ← b
    end while
    return c
end procedure
```

Najważniejsze obserwacje:

- W każdej iteracji pętli while jedna krawędź zostanie pokolorowana.
- Zawsze używamy kolorów, które są wynikami funkcji smallest_free, ale skoro każdy wierzchołek ma co najwyżej $\Delta(G)$ wychodzących z niego krawędzi, to smallest_free może przybrać $\Delta(G)+1$ różnych wartości (od 0 do $\Delta(G)$).
- Przed odwróceniem ab_u-ścieżki z u nie może wychodzić krawędź w kolorze b prowadząca do wierzchołka spoza wachlarza, bo wtedy moglibyśmy ten wachlarz (będący maksymalnym) powiększyć.
- Jeżeli przed odwróceniem ab_u-ścieżki z u nie wychodzi żadna krawędź o kolorze b, to ab_u-ścieżka jest pusta i odwrócenie jej nic nie zmieni, wtedy w następnym kroku $w \leftarrow fan.back$. Jeżeli przed odwróceniem ab_u-ścieżki z u wychodzi krawędź w kolorze b i prowadzi ona do v_i na wachlarzu, to po odwróceniu b będzie wolne na u i na v_i, a w następnym kroku $w \leftarrow v_i$.

3.3. $C_1, C_2, C_3, ..., C_b$**.**

Przypomnijmy, że według definicji 1.13 acyklicznym kolorowaniem krawędziowym nazywamy poprawne kolorowanie krawędziowe, w którym nie występują dwukolorowe cykle. Skoro w poprzednim kroku udało nam się już znaleźć poprawne kolorowanie krawędziowe, to być może nie musimy nic więcej robić. Aby się przekonać, czy nie wystarczy zwrócić kolorowania c, musimy znaleźć wszystkie cykle dwukolorowe jakie występują w grafie G z kolorowaniem c.
Zbiór wszystkich cykli dwukolorowych w naszym grafie zapiszemy jako $\{C_1, C_2, ..., C_b\}$. Jeżeli po wyznaczeniu tego zbioru okaże się, że $b = 0$ to faktycznie możemy zakończyć działanie algorytmu zwracając kolorowanie $c_a = c$, nie jest to jednak ciekawy przypadek.
Gdy pomyślimy o tym jak znaleźć wszystkie dwukolorowe cykle w grafie, to nasuwają się dwa możliwe rozwiązania:

(1) Znajdź wszystkie cykle w grafie G, a następnie sprawdź, które z nich są dwukolorowe w kolorowaniu c.
(2) Dla każdej pary kolorów (a, b) ze zbioru wartości funkcji c znajdź wszystkie cykle w kolorach (a, b), a następnie weź sumę uzyskanych zbiorów cykli.

Podejście nr 1 jest problematyczne, na pewno nie moglibyśmy go zastosować gdyby G mógł być dowolnym grafem (K_n zawiera co najmniej $\frac{(n-1)!}{2}$ cykli, dla $n > 2$), więc szukanie tych cykli psułoby wielomianowy czas działania naszego algorytmu. Wiemy jednak, że G jest grafem o dużej talii, więc może dałoby się jakoś wykorzystać tę własność i mimo wszystko spróbować najpierw znaleźć wszystkie cykle G. To podejście ma jeszcze jeden ciekawy aspekt, otóż graf G nie jest dowolnym grafem, tylko takim, dla którego z jakiegoś powodu chcemy znać acykliczne kolorowanie krawędziowe. Specyfika problemu może więc sprawić, że mamy więcej informacji o grafie G i może już przed wykonaniem tego algorytmu znamy wszystkie cykle jakie występują w G. W takim przypadku moglibyśmy rozważyć zastosowanie rozwiązania nr 1.
My jednak skłonimy się do podejścia drugiego. Okazuje się bowiem, że bez żadnej dodatkowej wiedzy na temat G jest ono prostsze i szybsze. W takim razie, musimy umieć znajdować wszystkie cykle dwukolorowe w zadanych dwóch kolorach. Znacząco ułatwia ten proces następujący fakt: każde dwa różne cykle dwukolorowe o tych samych dwóch kolorach są rozłączne krawędziowo. Gdyby tak nie było to mielibyśmy sytuację, w której dwa cykle mają pewną wspólną ścieżkę, która następnie się 'rozchodzi się' do różnych wierzchołków. Wtedy z ostatniego wierzchołka na wspólnej ścieżce musiałyby wychodzić dwie różne krawędzie w identycznym kolorze (jedna dla każdego cyklu), co oczywiście nie jest możliwe.
Szukanie cyklu w grafie jest jednym z najpopularniejszych zastosowań algorytmu DFS. Zazwyczaj ogranicza się ono jednak do sprawdzania, czy w grafie w ogóle występuje cykl. Zauważmy jednak, że gdy cykle są rozłączne krawędziowo, wtedy jednym przejściem algorytmu DFS możemy znaleźć wszystkie cykle. Wyobraźmy sobie następującą procedurę:

(1) Znajdź cykl C w grafie G używając DFS i zapisz go do zbioru S (jak nie istnieje to zakończ)
(2) Usuń z $E(G)$ krawędzie należące do cyklu C
(3) Idź do punktu (1)

Taki algorytm jest poprawny, w praktyce jednak nie musimy usuwać krawędzi i uruchamiać DFS'u od nowa, zauważmy jednak, że wykrycie jednego cyklu rozłącznego krawędziowo z pozostałymi w żaden sposób nie wpływa na to, czy wykryjemy kolejny.
W takim razie do znalezienia $C_1, C_2, C_3, ..., C_b$ potrzebujemy tyle razy wykonać DFS, ile jest par kolorów, czyli $\binom{\Delta(G)+1}{2}$ razy.

Algorithm 10 DFS from v on graph G

```
procedure DFS(G, c, cycles, v)
    dfs_color[v] ← grey
    for all u ∈ G[v] \ dfs_parent[v] do
        if dfs_color[u] = white then
            dfs_parent[u] = v
            DFS(G, c, cycles, u)
        else if dfs_color[u] = grey then
            Build cycle C = (u, ..., v, u) using dfs_parent array
            cycles.push(C)
        end if
    end for
end procedure
```

Algorithm 11 Find all bichromatic cycle in G with coloring c

```
procedure BICHROMATIC_CYCLES(G, c)
    cycles ← empty
    for 0 ⩽ i < j ⩽ Δ(G) do
        Reset dfs_color - white for all
        Reset dfs_parent - −1 for all
        G' = (V(G), {e ∈ E(G) | c(e) ∈ {i, j}})
        for v ← 0, N − 1 do
            if dfs_color[v] = white then
                DFS(G', c, cycles, v)
            end if
        end for
    end for
end procedure
```

3.4. $E_1, E_2, E_3, ..., E_b$**.**
Zdefiniujmy zbiory $E_1, E_2, E_3, ..., E_b$, gdzie b oznacza liczbę znalezionych cykli dwukolorowych, jako zbiory spełniające warunki:

- $E_i \subseteq C_i$
- $|E_i| = \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor$
- $\forall_{i,j} : E_i \cap E_j = \emptyset$

Naszym głównym celem jest zmiana obliczonego wcześniej poprawnego kolorowania c na takie, w którym nie ma dwukolorowych cykli. Zbiory $E_1, E_2, E_3, ..., E_b$ posłużą nam do tego w ten sposób, że chcąc sprawić, żeby C_i przestał być cyklem dwukolorowym, wybierzemy krawędź z E_i i zmienimy jej kolor.
Nieoczywisty jest już sam fakt, że takie zbiory istnieją. Chcąc przedstawić dowód istnienia tych zbiorów, znowu musimy wprowadzić kilka definicji.

Definicja 2.7. Grafem dwudzielnym nazywamy trójkę $G = (X, Y, E)$, gdzie X i Y są skończonymi zbiorami wierzchołków, a E jest zbiorem krawędzi takim, że $E \subseteq X \times Y$. Graf dwudzielny jest grafem spełniającym $V(G) = X \cup Y$, $E(G) = E$.

Definicja 2.8. Skojarzeniem w grafie G nazywamy podzbiór krawędzi tego grafu taki, że każdy wierzchołek grafu należy do co najwyżej jednej krawędzi w tym podzbiorze. Skojarzenie zazwyczaj oznaczamy jako M. M jest skojarzeniem w grafie G, jeżeli dla każdego wierzchołka $v \in V(G)$ zachodzi $|\{e \in M \mid v \in e\}| \leqslant 1$.

Definicja 2.9. X-skojarzenie doskonałe w grafie dwudzielnym $G = (X, Y, E)$ jest skojarzeniem, w którym istnieje dokładnie jedna krawędź z wierzchołkiem x dla każdego $x \in X$.

Równoważnie moglibyśmy zdefiniować X-skojarzenie doskonałe w grafie dwudzielnym $G = (X, Y, E)$ jako skojarzenie wielkości $|X|$.

Twierdzenie 2.10. *(**Twierdzenie Halla**) [Hal35] Niech $G = (X, Y, E)$ będzie grafem dwudzielnym. W grafie tym istnieje X-skojarzenie doskonałe wtedy i tylko wtedy, gdy dla każdego podzbioru $W \subseteq X$ zachodzi $|W| \leqslant |N(W)|$, gdzie $N(W)$ oznacza liczbę sąsiadów wierzchołków ze zbioru W w grafie G.*

Twierdzenie Halla można wypowiedzieć równoważnie na wiele różnych sposobów. W tej pracy przedstawiliśmy je w najbardziej użytecznej dla nas formie.
Chcemy teraz zastosować twierdzenie 2.10 do udowodnienia istnienia zbiorów $E_1, E_2, E_3, ..., E_b$ o własnościach wymienionych wyżej.

Dowód
Weźmy $X_H = \left\{C_{i,j} \mid i \in \{1, ..., b\} \wedge j \in \left\{1, ..., \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor \right\}\right\}$.
$C_{i,j}$ możemy zinterpretować jako jedną z $\left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor$ kopii cyklu C_i. Weźmy graf dwudzielny $G_H = (X_H, E(G), E_H)$, gdzie

$$\forall_{C_{i,j} \in X_H, e \in E(G)} : ((C_{i,j}, e) \in E_H \Leftrightarrow \text{krawędź } e \text{ należy do cyklu } C_i)$$

Zanim przejdziemy do sprawdzania warunków twierdzenia Halla dla grafu dwudzielnego G_H, musimy pamiętać o dwóch własnościach:

(1) Długość cyklu C_i wynosi co najmniej $g(G)$.
(2) Każda krawędź z $E(G)$ należy do co najwyżej $\Delta(G)$ cykli dwukolorowych.

Pierwszą z nich otrzymujemy wprost z definicji talii grafu. Druga jest prostym wnioskiem z tego, że „każde dwa różne cykle dwukolorowe o tych samych dwóch kolorach są rozłączne krawędziowo", co pokazywaliśmy wcześniej. Weźmy dowolną krawędź $(u, v) \in E(G)$ należącą do k dwukolorowych cykli. Wtedy z u musi wychodzić co najmniej k krawędzi o kolorach różnych od $c_1(u, v)$, więc $k < k + 1 \leqslant \Delta(G)$.
Weźmy zatem dowolny podzbiór $W \subseteq X_H$ i oszacujmy $N(W)$. Każdy wierzchołek z W jest sąsiadem co najmniej $g(G)$ wierzchołków w G_H, jeśli je wypiszemy

- razem z powtórzeniami - to każdy z elementów z $E(G)$ wystąpi co najwyżej $\left(\Delta(G) \cdot \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor\right)$ razy. Jest to liczba cykli, w których może się zawierać przemnożona przez liczbę kopii jakie ma dany cykl w grafie G_H. Otrzymujemy nierówność:

$$N(W) \geqslant \frac{|W| \cdot g(G)}{\Delta(G) \cdot \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor} \geqslant \frac{|W| \cdot g(G)}{\Delta(G) \cdot \frac{g(G)}{\Delta(G)}} = |W|$$

Pokazaliśmy, że założenia twierdzenia Halla są spełnione dla G_H, czyli korzystając z tego twierdzenia wiemy, że w grafie G_H istnieje X_H-skojarzenie doskonałe.
Ostatnim krokiem dowodu istnienia zbiorów $E_1, E_2, E_3, ..., E_b$ jest ich dokładne zdefiniowanie. Niech E_i będzie zbiorem krawędzi skojarzonych z $C_{i,1}, ..., C_{i,\left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor}$ w X_H-skojarzeniu doskonałym.

■

Jak na razie udało nam się udowodnić, że zbiory (E_i) są poprawnie zdefiniowane. Nie powiedzieliśmy jeszcze jednak nic o tym jak będziemy ich szukać algorytmicznie. Posłużą nam do tego sieci przepływowe.

Definicja 2.11. Siecią przepływową nazywamy piątkę $A = (V, E, c, s, t)$, gdzie

- V jest zbiorem wierzchołków.
- E zbiorem skierowanych krawędzi.
- $c : E \to \mathbb{N}$ nazywamy funkcją przepustowości.
- $s \in V$ jest wierzchołkiem o stopniu wejściowym równym zero (żadna krawędź nie prowadzi do s).
- Analogicznie $t \in V$ jest wierzchołkiem o stopniu wyjściowym równym zero (żadna krawędź nie wychodzi z t).

s nazywamy źródłem, a t ujściem.

Definicja 2.12. $f : E \to \mathbb{N}$ jest **funkcją przepływu** dla sieci przepływowej $A = (V, E, c, s, t)$ pod warunkiem, że:

$$\forall_{e \in E} : f(e) \leqslant c(e)$$

oraz

$$\forall_{v \in V \setminus \{s,t\}} : \sum_{(x,v) \in E} f(x, v) = \sum_{(v,y) \in E} f(v, y)$$

Wartość funkcji przepływu f definiujemy jako:

$$|f| = \sum_{v:(s,v) \in E} f(s, v) = \sum_{v:(v,t) \in E} f(v, t)$$

Intuicja dotycząca funkcji przepływowej jest następująca:

- Nie może przepłynąć więcej niż wynosi przepustowość.
- Tyle ile wpłynęło do wierzchołka, tyle musi wypłynąć.

Typowym problemem dla sieci przepływowej jest problem znalezienia maksymalnej wartości funkcji przepływu.

Definicja 2.13. $(x_1, x_2, ..., x_k)$ nazywamy ścieżką powiększającą w sieci przepływowej $A = (V, E, c, s, t)$, jeśli $x_1 = s$, $x_k = t$ oraz $\forall_{1 \leqslant i < n}$ zachodzi:

$$(((x_i, x_{i+1}) \in E \wedge f(x_i, x_{i+1}) < c(x_i, x_{i+1})) \vee ((x_{i+1}, x_i) \in E \wedge f(x_{i+1}, x_i) > 0))$$

Czyli ścieżka powiększająca zaczyna się w źródle, kończy w ujściu i idzie po krawędziach zorientowanych jak w E i wtedy wymaga by krawędź była jeszcze 'nienasycona' ($f < c$) lub po krawędziach zorientowanych odwrotnie do E, wtedy wymagamy by coś tą krawędzią już 'płynęło' przy pomocy funkcji f.
Po znalezieniu ścieżki powiększającej możemy zwiększyć funkcje przepływu f o co najmniej 1. Dokładnie w ten sposób działa algorytm Ford'a–Fulkerson'a [FF56] znajdowania maksymalnego przepływu, nie precyzuje on jednak sposobu szukania ścieżek powiększających. Modyfikacją powyższego algorytmu jest algorytm Edmondsa-Karpa [EK72]. Jedyna różnica polega na tym, że ścieżki powiększające szukamy algorytmem BFS, co pozwala nam na znajdowanie najkrótszych ścieżek powiększających.
Zastanówmy się teraz jak przedstawić nasz problem wyznaczania zbiorów $E_1, E_2, E_3, ..., E_b$ jako problem wyznaczania maksymalnego przepływu.
Niech $A = (V_A, E_A, p, s, t)$ będzie siecią przepływową taką, że:

- $V_A = \{s, t\} \cup \{1, 2, 3, ..., b\} \cup E(G)$
- We wszystkich poniższych punktach $i \in \{1, 2, 3, ..., b\}$.
- E_A zawiera b krawędzi postaci (s, i), krawędzie (i, e) wtedy i tylko wtedy, gdy $e \in C_i$ oraz $|E(G)|$ krawędzi postaci (e, t).
- $p(s, i) = \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor$
- $p(i, e) = 1$
- $p(e, t) = 1$

Załóżmy na chwilę, że znamy zbiory $E_1, E_2, E_3, ..., E_b$. Możemy w takim razie zdefiniować funkcję $f : E_A \to \mathbb{N}$ taką, że:

- $f(s, i) = \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor$
- $f(i, e) = 1 \Leftrightarrow e \in E_i$
- $f(i, e) = 0 \Leftrightarrow e \notin E_i$
- $f(e, t) = 1 \Leftrightarrow \exists_i : e \in E_i$
- $f(e, t) = 0 \Leftrightarrow \forall_i : e \notin E_i$

Istnienie takiej funkcji przepływu gwarantują nam własności zbiorów (E_i). $|f| = b \cdot \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor$. Jest to jednocześnie maksymalna wartość funkcji przepływu na sieci A, przez nasycenie wszystkich krawędzi wychodzących z s.
Zauważmy, że każda maksymalna funkcja przepływu na A musi spełniać:

- $f(s, i) = \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor$
- $\forall_{i \in \{1,...,b\}} \exists$ dokładnie $\left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor$ różnych $e \in E(G)$ takich, że $f(i, e) = 1$.

Dzięki tym własnościom możemy zdefiniować zbiory E_i znając dowolną maksymalną funkcję przepływu na A. Niech f_M będzie maksymalną funkcją przepływu na A. Weźmy $E_i = \{e \in E(G) \mid f(i, e) = 1\}$. W oczywisty sposób, tak zdefiniowane zbiory E_i spełniają wymagane przez nas własności, podane na początku sekcji 3.4. Do znalezienia dowolnej maksymalnej funkcji przepływu na A używamy algorytmu Edmondsa-Karpa.
Nie będziemy przedstawiać pseudokodu ogólnej wersji algorytmu maksymalnego przepływu, tylko wersję uproszczoną, dostosowaną do naszego problemu i korzystającą z następujących obserwacji:

- Dla każdego $e \in E(G)$ z e wychodzi tylko jedna krawędź w sieci przepływowej A - jest to krawędź do t o przepustowości 1. Dzięki temu zamiast

trzymać wartość funkcji przepływu dla każdej krawędzi w A możemy trzymać tylko:
  - Dla każdego $i \in \{1, ..., b\}$ aktualną wartość $f(s, i)$.
  - Dla każdego $e \in E(G)$ trzymamy $assignment(e)$, które przyjmuje wartość empty jak $f(e, t) = 0$, wartość i jak $f(i, e) = 1$ oraz $f(e, t) = 1$.
- Wszystkie krawędzie wchodzące do t mają przepustowość 1, więc zawsze powiększając wartość przepływu wzdłuż ścieżki powiększającej, zwiększamy f o dokładnie 1.
- Ścieżki powiększające będą składać się z krawędzi $(i, e) \in E_A$ takich, że $assignment(e) \neq i$, oraz (e, i) takich, że $assignment(e) = i$. Mamy na myśli aktualne wartości $assignment$ - w trakcie znajdowania tej ścieżki.

Algorithm 12 Find max_flow in network A defined above, return $\{E_1, E_2, E_3, ..., E_b\}$

procedure MAX_FLOW($G, \{C_1, C_2, C_3, ..., C_b\}$)
  for $i \leftarrow 1, b$ **do**
    $E_i \leftarrow$ empty
    $f(s, i) \leftarrow 0$
  end for
  for all $e \in E(G)$ **do**
    $assignment(e) \leftarrow$ empty
  end for
  $flow\_value \leftarrow b \cdot \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor$
  for $\_ \leftarrow 1, flow\_value$ **do**
    reset BFS_parent ▷ In $BFS\_parent[x]$, we store a vertex from V_A ▷ that enqueued x in the BFS queue.
    $Q \leftarrow$ empty
    Find $i : f(s, i) < \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor$
    $f(s, i) \leftarrow f(s, i) + 1$
    $Q.push(i)$
    while Q not empty **do**
      $x \leftarrow Q.front$
      if $x \in E(G)$ **then**
        if $assignment(x) =$ empty **then** ▷ Found an augmenting path.
          do
            $assignment(x) \leftarrow BFS\_parent(x)$
            $x \leftarrow BFS\_parent(BFS\_parent(x))$
          while $x \neq$ empty
          break
        end if
        if $BFS\_parent(assignment(x)) =$ empty **then**
          $BFS\_parent(assignment(x)) \leftarrow x$
          $Q.push(assignment(x))$
        end if
      else ▷ $x \in \{1, ..., b\}$
        for all $e \in C_x$ **do**
          if $BFS\_parent(e) =$ empty and $assignment(e) \neq x$ **then**
            $BFS\_parent(e) \leftarrow x$
            $Q.push(e)$
          end if
        end for
      end if
    end while
  end for
  for all $e \in E(G)$ **do**
    $i \leftarrow assignment(e)$
    $E_i.push(e)$
  end for
  return $\{E_1, E_2, E_3, ..., E_b\}$
end procedure

3.5. Grafy H i $\bar{H}$.

Tworzymy graf H będący podgrafem G. $H = (V(G), \bigcup E_i)$. Tworzymy graf $\bar{H} = (E(H), \bar{E})$, $(e_1, e_2) \in \bar{E} \Leftrightarrow distance_G(e_1, e_2) \leqslant 1 \wedge e_1 \neq e_2$.
$(distance_G(e_1, e_2) \leqslant 1 \wedge e_1 \neq e_2)$ oznacza dokładnie tyle, że e_1, e_2 to dwie różne krawędzie ze wspólnym wierzchołkiem lub takie, że istnieje krawędź łącząca je.
Naszym celem jest sprawienie, by każdy z cykli $C_1, ..., C_b$ przestał być dwukolorowy. Zauważmy, że w tym celu wystarczy wybrać po jednej krawędzi z każdego cyklu i przekolorować ją na nowy kolor. Musimy to jednak zrobić tak, by kolorowanie pozostało poprawne - nie chcemy przekolorować krawędzi e_1, e_2 takich, które mają wspólny wierzchołek ($distance_G(e_1, e_2) = 0$). Należy także zadbać o to, by nie powstał cykl dwukolorowy zawierający nowy kolor. W tym celu dobierzemy krawędzie do pokolorowania tak, aby nie występowała ścieżka długości 3 z dwoma krawędziami w nowym kolorze, czyli wykluczamy jednoczesne przekolorowanie obu e_1, e_2, jeżeli ($distance_G(e_1, e_2) = 1$).

Definicja 2.14. W grafie $G = (V, E)$ **zbiorem niezależnym** nazywamy zbiór wierzchołków $V' \subseteq V$, pomiędzy którymi nie ma żadnej krawędzi. Czyli podgraf grafu G indukowany na V' nie posiada krawędzi.

Chcemy teraz znaleźć zbiór niezależny w $\bar{H}$ zawierający dokładnie jedną krawędź z każdego zbioru E_i (wierzchołki $\bar{H}$, to krawędzie w G), następnie przekolorujemy wszystkie krawędzie w tym zbiórze niezależnym na nowy kolor. Operacje szukania zbioru niezależnego w $\bar{H}$ zaprezentujemy w następnym podrozdziale.

3.6. S - zbiór niezależny w $\bar{H}$ i zakończenie algorytmu.

Do pokazania, że taki zbiór istnieje, a następnie do znalezienia go będziemy potrzebować Lokalnego Lematu Lovász'a oraz jego wersji algorytmicznej.

Twierdzenie 2.15. *[Spe77] (Lokalny Lemat Lovász'a - wersja symetryczna) Niech $A_1, ..., A_n$ będą zdarzeniami pewnej przestrzeni probabilistycznej takimi, że $(\forall_i : Pr(A_i) \leqslant p)$ i każde zdarzenie jest zależne od co najwyżej d spośród pozostałych zdarzeń. Jeżeli $ep(d+1) \leqslant 1$ (e jest liczbą Eulera), to $Pr\left(\overline{A_1} \wedge \overline{A_2} \wedge ... \wedge \overline{A_n}\right) > 0$.*

Twierdzenie 2.16. *[MT10] (Algorytmiczny Lokalny Lemat Lovász'a - wersja ogólna) Niech $A_1, ..., A_n$ będą zdarzeniami pewnej przestrzeni probabilistycznej, a $N(i)$ zbiorem indeksów takim, że A_i jest zależne od zdarzeń ze zbioru $\{A_j | j \in N(i)\}$. Jeżeli istnieją $x_1, ..., x_n \in [0, 1)$ spełniające*

$$Pr(A_i) \leqslant x_i \prod_{j \in N(i)} (1 - x_j)$$

to algorytm:

Algorithm 13 *Moser–Tardos local lemma algorithm*

procedure LLL
  Zainicjalizuj wszystkie zmienne losowe w danej przestrzeni probabilistycznej.
  while *Zachodzi pewne zdarzenie A_i* ***do***
    Wylosuj nowe wartościowanie wszystkich zmiennych zdarzenia A_i.
  end while
end procedure

gwarantuje, że wartość oczekiwana tego, ile razy musimy wylosować nowe wartościowanie zmiennych zdarzenia A_i wynosi co najwyżej $\frac{x_i}{1-x_i}$.

Czyli oczekiwany czas działania algorytmu Moser'a–Tardos'a będzie równy sumie:

$$Init+\sum_{i=1}^{n}\left(\frac{x_i}{1-x_i}\cdot(\text{czas potrzebny na ponowne wylosowanie zmiennych zdarzenia } A_i)\right),$$

gdzie $Init$ jest czasem potrzebnym na zainicjalizowanie wszystkich zmiennych w danej przestrzeni probabilistycznej.
Będziemy chcieli zastosować twierdzenie 2.16, ale pokażemy, że w naszym przypadku zachodzą założenia twierdzenia 2.15. Możemy tak zrobić, ponieważ jak jest pokazane chociażby w [Spe77] wersja symetryczna Lokalnego Lematu Lovász'a jest szczególnym przypadkiem wersji ogólnej. Dowód tego faktu opiera się na wzięciu $x_1 = ... = x_n = \frac{1}{d+1}$. Używając w ten sposób twierdzenia 2.16 wartość oczekiwana tego, ile razy musimy wylosować nowe wartościowanie zmiennych zdarzenia A_i wynosi:

$$\frac{x_i}{1-x_i} = \frac{1}{d+1}\cdot\frac{d+1}{d} = \frac{1}{d}$$

Dzięki zastosowaniu liniowości wartości oczekiwanej otrzymujemy wartość oczekiwana tego, ile razy musimy wylosować nowe wartościowanie zmiennych dowolnego zdarzenia równą $\frac{n}{d}$.
Stąd, oczekiwany czas działania algorytmu Moser'a–Tardos'a przy założeniach twierdzenia 2.15, będzie równy sumie:

$$Init+\frac{n}{d}\cdot(\text{czas potrzebny na ponowne wylosowanie zmiennych pewnego zdarzenia } A_i),$$

gdzie $Init$ definiujemy tak jak wyżej.
Poniższy lemat, a tak właściwie jego dowód, pokaże nam jak powinniśmy dobrać zdarzenia $A_1, ..., A_n$ oraz wartości p i d konieczne do zastosowania twierdzenia 2.15 w celu znalezienia odpowiedniego zbioru niezależnego w $\bar{H}$.

Lemat 2.17. *[Wan20] Niech $G(V, E)$ będzie grafem, a $V(G) = V_1 \cup V_2 \cup ... \cup V_r$ podziałem zbioru jego wierzchołków takim, że $\forall_i : |V_i| \geqslant 2e\Delta(G)$, gdzie e jest liczbą Eulera. Twierdzimy, że w G istnieje zbiór niezależny wierzchołków, zawierający dokładnie po jednym wierzchołku z każdego V_i.*

Dowód
Załóżmy, że $\forall_i : |V_i| = \lceil 2e\Delta(G)\rceil$. Jeżeli uda nam się przeprowadzić dowód pod tym założeniem to tym bardziej lemat będzie prawdziwy dla $|V_i| \geqslant 2e\Delta(G)$. Wybieramy po jednym wierzchołku z każdego V_i losując jednostajnie i niezależnie. W ten sposób powstaje zbiór W. Dla dowolnego wierzchołka v wiemy, że $Pr(v \in W) = \frac{1}{\lceil 2e\Delta(G)\rceil}$. Weźmy dowolną krawędź $(u, v) = f \in E$ i niech A_f będzie takim zdarzeniem, że oba wierzchołki krawędzi f znalazły się w W. Zatem:

$$Pr(A_f) \leqslant P(u \in W)P(v \in W) = \frac{1}{\lceil 2e\Delta(G)\rceil^2}.$$

Wiemy również, że A_f jest zależne od wszystkich zdarzeń A_e takich, że e ma co najmniej jeden koniec w tym samym zbiorze co u lub v. Liczbę takich zdarzeń A_e możemy oszacować przez $2 \cdot \lceil 2e\Delta(G)\rceil \cdot \Delta(G)$, przy czym w szacowaniu nie może zachodzić równość, ponieważ $e \neq f$.
Stosując notację z twierdzenia 2.15 mamy $p = \frac{1}{\lceil 2e\Delta(G)\rceil^2}$ oraz

$d+1 \leqslant 2 \cdot \lceil 2e\Delta(G) \rceil \cdot \Delta(G)$. Jako, że $ep(d+1) \leqslant 1$ to udało nam się spełnić założenia twierdzenia 2.15, co kończy dowód.

■

Aby zastosować lemat 2.17 potrzebujemy sprawdzić czy zachodzi założenie tego lematu, czyli w naszym przypadku czy $|E_i| \geqslant 2e\Delta(\bar{H})$. Potrzebujemy oszacować $\Delta(\bar{H})$. Weźmy dowolny wierzchołek $f \in V(\bar{H})$, jest on krawędzią w G. Niech $f = (u, v)$. Sąsiedzi f w $\bar{H}$ spełniają $(distance_G(e, f) \leqslant 1$, więc możemy oszacować $\Delta(\bar{H})$ przez liczbę krawędzi wychodzących z sąsiadów u zsumowaną z liczbą krawędzi wychodzących z sąsiadów v. Ostatecznie $\Delta(\bar{H}) < 2(\Delta(G))^2$. Przekształcając założenie dowodzonego przez nas twierdzenia otrzymujemy pożądaną nierówność:

$$g(G) \geqslant 4e\left(\Delta(G)\right)^3 \Rightarrow$$
$$\Rightarrow \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor \geqslant 2e2(\Delta(G))^2 \Rightarrow$$
$$\Rightarrow |E_i| \geqslant 2e\Delta(\bar{H})$$

Podsumowując, idea wyznaczania zbioru S jest następująca: $\bar{H}$ spełnia założenia lematu 2.17, więc wiemy, że S istnieje, ale co ważniejsze dowód lematu 2.17 pokazuje jak dobrać $A_1, ..., A_n$ oraz p i d tak aby spełniały założenia twierdzenia 2.15. Wspomnieliśmy też o tym, że założenia twierdzenia 2.15 są mocniejsze od założeń twierdzenia 2.16, więc będziemy mogli zastosować twierdzenie 2.16 wraz z zawartym tam algorytmem i w ten sposób znajdziemy zbiór S.

Algorithm 14 Our application of Moser–Tardos algorithm

```
procedure MOSER_TARDOS(G, E.c)
    S ← empty
    for all E_i ∈ E do
        S.push(random_edge(E_i))
    end for
    while ∃_{e,f∈S} : (e, f) ∈ E(H̄) do                ▷ e ∈ E_i, f ∈ E_j
        S.remove(e)
        S.remove(f)
        S.push(random_edge(E_i))
        S.push(random_edge(E_j))
    end while
    return S
end procedure
```

Aby móc policzyć złożoność musimy jeszcze policzyć liczbę zdarzeń (n) oraz wartość liczby d, które musimy zdefiniować, by spełnić założenia twierdzenia 2.15. Wartość obu tych liczb możemy poznać poprzez analizę dowodu lematu 2.17. Otrzymujemy

$$n = |\bar{E}| = \frac{\sum_{e \in V(\bar{H})} deg_{\bar{H}}(e)}{2} \leqslant \frac{b\left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor \cdot \Delta(\bar{H})}{2} \leqslant \frac{b\frac{g(G)}{\Delta(G)} \cdot 2\Delta(G)}{2} = bg(G)\Delta(G)$$
$$d < 2 \cdot \lceil 2e\Delta(\bar{H}) \rceil \cdot \Delta(\bar{H}) \leqslant 2 \cdot \lceil 2e2(\Delta(G))^2 \rceil \cdot 2(\Delta(G))^2$$

W naszym przypadku czas inicjalizacji wszystkich zmiennych w przestrzeni probabilistycznej wynosi $b \cdot O(1) = O(b)$, a czas zmiany wartościowania zmiennych związanych z konkretnym zdarzeniem $2 \cdot O(1) = O(1)$. Korzystając z poczynionych

wcześniej wniosków z twierdzenia 2.16 uzyskujemy oczekiwaną złożoność znajdowania zbioru S równą $O(b + \frac{n}{d})$, co jest oczywiście wielomianowe względem rozmiaru grafu równego $|V(G)| + |E(G)|$. Jak już mówiliśmy, ostatnim krokiem jest przekolorowanie krawędzi z S na kolor $\Delta(G) + 1$. Uzyskujemy w ten sposób kolorowanie $c_a : E(G) \to \{0, 1, ..., \Delta(G) + 1\}$ będące acyklicznym kolorowaniem krawędziowym grafu G na $\Delta(G) + 2$ kolorów.

■

4. Dokładna złożoność

W tej sekcji spróbujemy oszacować złożoność obliczeniową podanego algorytmu. Interesuje nas zwłaszcza wyrażenie złożoności każdego kroku algorytmu, jako funkcji od rozmiaru wejściowego grafu G, czyli funkcji od $|V(G)|$ i $|E(G)|$. Oznaczmy $|V(G)| = N$ i $|E(G)| = M$. Wiemy, że zachodzi $M \leqslant N^2$. Pokazano również, że $M \leqslant N^{1+\frac{O(1)}{g(G)}}$ [AHL02]. Warto więc mieć na uwadze, że w przypadku grafów z dużą talią, tak jak u nas, liczba krawędzi jest bliższa liczbie wierzchołków niż kwadratowi liczby wierzchołków.

4.1. $\Delta(G)$ i $g(G)$.
Jeśli przechowujemy graf w postaci list sąsiedztwa, czyli dla każdego wierzchołka trzymamy listę sąsiadujących z nim wierzchołków, to czas obliczania $\Delta(G)$ wynosi $O(N)$ (zakładamy, że jako listy używamy struktury, która zna swoją długość, jak np. std::vector w C++).
Wyznaczając $g(G)$ używamy algorytmu BFS od każdego wierzchołka. Wiadomo, że złożoność jednego algorytmu BFS wynosi $O(N + M)$, więc całkowita złożoność to $O(N^2 + NM)$.
Przy okazji zauważmy, że $\Delta(G) \leqslant N$ i $g(G) \leqslant N$.

4.2. $c : E(G) \to \{0, 1, ..., \Delta(G)\}$.
Wszystkie operacje w głównej pętli algorytmu, takie jak szukanie wolnego koloru, rotacja wachlarza czy odwracanie ścieżki realizujemy w czasie $O(N)$. Biorąc pod uwagę, że główna pętla wykonuje się dokładnie raz dla każdej krawędzi, dostajemy złożoność $O(NM)$.
Istnieje algorytm kolorowania krawędziowego, który działa w $O(M\sqrt{N})$ [Sin21].

4.3. $C_1, C_2, C_3, ..., C_b$.
W opisie znajdowania dwukolorowych cykli pisaliśmy, że potrzebujemy wykonać algorytm DFS $\binom{\Delta(G)+1}{2}$ razy. Szacujemy

$$\binom{\Delta(G)+1}{2} \leqslant (\Delta(G))^2 \leqslant N^2$$

A złożoność jednego algorytmu DFS jest równa $O(N + M)$, więc na znalezienie wszystkich cykli potrzebujemy $O(N^3 + N^2M)$.
Długość każdego z tych cykli to co najmniej $g(G)$, a sumaryczna długość wszystkich cykli nie może przekraczać $M\binom{\Delta(G)+1}{2}$. To pozwala nam oszacować b:

$$b \leqslant \frac{M\binom{\Delta(G)+1}{2}}{g} \leqslant \frac{M(\Delta(G)+1)(\Delta(G)+2)}{8e(\Delta(G))^3} \leqslant M$$

4.4. $E_1, E_2, E_3, ..., E_b$**.**
Policzmy najpierw liczbę wierzchołków i krawędzi w sieci przepływowej skonstruowanej przez nas w celu wyznaczenia $E_1, E_2, E_3, ..., E_b$.

$$|V_A| = 2 + b + M = O(M)$$

$$|E_A| \leqslant b + M + M\binom{\Delta(G)+1}{2} = O(MN^2)$$

Wartość maksymalnej funkcji przepływu szukanej przez nas w tym kroku wynosi

$$|f| = b\left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor \leqslant bg(G) = O(MN)$$

Stosowany przez nas algorytm Edmondsa-Karpa ma złożoność $O(|V_A||E_A|^2) = O(M^3N^4)$. Jednak jako, że algorytm Edmondsa-Karpa jest szczególnym rodzajem algorytmu Forda-Fulkersona to ma on też złożoność $O(|E_A||f|) = O(M^2N^3)$.
Istnieje wiele innych algorytmów rozwiązujących problem szukania maksymalnego przepływu, np. algorytm push-relabel [GT88] - przy jego użyciu uzyskalibyśmy złożoność $O(|V_A|^3) = O(M^3)$, albo algorytm z 2022 roku autorstwa Chen, Kyng, Liu, Peng, Gutenberg i Sachdeva [Che+22], którego złożoność to $O(|E_A|^{1+o(1)} \log U)$, gdzie U jest maksymalną przepustowością krawędzi. Czyli u nas $U = \left\lfloor \frac{g(G)}{\Delta(G)} \right\rfloor = O(N)$, więc stosując ten algorytm złożoność wyniesie $O((MN^2)^{1+o(1)} \log N)$.
Biorąc pod uwagę, że u nas M jest prawdopodobnie bliskie N możliwe, że najlepiej byłoby zastosować algorytm push-relabel, w tym przypadku dodatkową zaletą jest też stosunkowo prosta implementacja.

4.5. Grafy H **i** $\bar{H}$**.**
Dla każdej krawędzi $e \in E(G)$ potrzebujemy znaleźć krawędzie $f \in E(G)$ takie, że $distance_G(e, f) \leqslant 1$. Dla konkretnej krawędzi łatwo to zrealizować w $O(M)$, czyli całkowita złożoność wyniesie $O(M^2)$.

4.6. S **- zbiór niezależny w** $\bar{H}$ **i zakończenie algorytmu.**
Pokazaliśmy już przy okazji omawiania tego etapu, że oczekiwany czas wyznaczania S wyniesie $O(b + \frac{n}{d})$. Udało nam się wtedy oszacować $n = O(bg(G)\Delta(G))$, więc

$$O\left(b + \frac{n}{d}\right) \leqslant O(b + bg(G)\Delta(G)) \leqslant O(M + MN^2)$$

ROZDZIAŁ 3

Podsumowanie

Nasza praca zapewnia solidne podstawy do praktycznego zastosowania acyklicznego kolorowania krawędziowego.
Warto również zwrócić uwagę na prace skupiające się na ograniczeniu acyklicznego indeksu chromatycznego, bez dodatkowych założeń o grafie. Przykładowym rezultatem takich prac jest $\chi'_a(G) \leqslant 3.569(\Delta(G) - 1)$ [FLP19] - prawdopodobnie nie jest to najlepsze aktualnie znane ograniczenie, z uwagi na tempo powstawania prac na ten temat. Przeważnie oparte są one na Lokalnym Lemacie Lovásza, więc także podają randomizowany algorytm działający w czasie wielomianowym.

Bibliografia

[AHL02] Noga Alon, Shlomo Hoory, and Nathan Linial. "The Moore Bound for Irregular Graphs". In: *Graphs and Combinatorics* 18.1 (Mar. 2002), pp. 53–57. ISSN: 1435-5914. DOI: 10.1007/s003730200002. URL: https://doi.org/10.1007/s003730200002.

[AMM12] Lars Døvling Andersen, Edita Máčajová, and Ján Mazák. "Optimal acyclic edge-coloring of cubic graphs". In: *Journal of Graph Theory* 71.4 (2012), pp. 353–364. DOI: https://doi.org/10.1002/jgt.20650. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jgt.20650. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jgt.20650.

[ASZ01] Noga Alon, Benny Sudakov, and Ayal Zaks. "Acyclic edge colorings of graphs". In: *Journal of Graph Theory* 37.3 (2001), pp. 157–167.

[AZ02] Alon and Zaks. "Algorithmic Aspects of Acyclic Edge Colorings". In: *Algorithmica* 32.4 (Apr. 2002), pp. 611–614. ISSN: 1432-0541. DOI: 10.1007/s00453-001-0093-8. URL: https://doi.org/10.1007/s00453-001-0093-8.

[Bec91] József Beck. "An algorithmic approach to the Lovász local lemma. I". In: *Random Structures & Algorithms* 2.4 (1991), pp. 343–365. DOI: https://doi.org/10.1002/rsa.3240020402. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rsa.3240020402. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rsa.3240020402.

[Che+22] Li Chen et al. *Maximum Flow and Minimum-Cost Flow in Almost-Linear Time*. 2022. arXiv: 2203.00671 [cs.DS].

[EK72] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *J. ACM* 19.2 (Apr. 1972), pp. 248–264. ISSN: 0004-5411. DOI: 10.1145/321694.321699. URL: https://doi.org/10.1145/321694.321699.

[FF56] L. R. Ford and D. R. Fulkerson. "Maximal Flow Through a Network". In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404. DOI: 10.4153/CJM-1956-045-5.

[FLP19] Paula M. S. Fialho, Bernardo N. B. de Lima, and Aldo Procacci. *A new bound on the acyclic edge chromatic index*. 2019. arXiv: 1912.04436 [math.CO].

[Geb+09] Assefaw H. Gebremedhin et al. "Efficient Computation of Sparse Hessians Using Coloring and Automatic Differentiation". In: *INFORMS Journal on Computing* 21.2 (2009), pp. 209–223. DOI: 10.1287/ijoc.1080.0286. eprint: https://doi.org/10.1287/ijoc.1080.0286. URL: https://doi.org/10.1287/ijoc.1080.0286.

[Grü73] Branko Grünbaum. "Acyclic colorings of planar graphs". In: *Israel Journal of Mathematics* 14.4 (Dec. 1973), pp. 390–408. ISSN: 1565-8511. DOI: 10.1007/BF02764716. URL: https://doi.org/10.1007/BF02764716.

[GT88] Andrew V. Goldberg and Robert E. Tarjan. "A New Approach to the Maximum-Flow Problem". In: *J. ACM* 35.4 (Oct. 1988), pp. 921–940. ISSN: 0004-5411. DOI: 10.1145/48014.61051. URL: https://doi.org/10.1145/48014.61051.

[Hal35] Peter Hall. "On Representatives of Subsets". In: *Journal of The London Mathematical Society-second Series* (1935), pp. 26–30. URL: https://api.semanticscholar.org/CorpusID:23252557.

[Hol81] Ian Holyer. "The NP-Completeness of Edge-Coloring". In: *SIAM Journal on Computing* 10.4 (1981), pp. 718–720. DOI: 10.1137/0210055. eprint: https://doi.org/10.1137/0210055. URL: https://doi.org/10.1137/0210055.

[MG92] J. Misra and David Gries. "A constructive proof of Vizing's theorem". In: *Information Processing Letters* 41.3 (1992), pp. 131–133. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(92)90041-S. URL: https://www.sciencedirect.com/science/article/pii/002001909290041S.

[MT10] Robin A. Moser and Gábor Tardos. "A Constructive Proof of the General Lovász Local Lemma". In: *J. ACM* 57.2 (Feb. 2010). ISSN: 0004-5411. DOI: 10.1145/1667053.1667060. URL: https://doi.org/10.1145/1667053.1667060.

[NG23] V. Vinitha Navis and A. Berin Greeni. "Application of acyclic coloring in wavelength assignment problem for butterfly and Beneš network". In: *Sādhanā* 48.3 (July 2023), p. 146. ISSN: 0973-7677. DOI: 10.1007/s12046-023-02164-6. URL: https://doi.org/10.1007/s12046-023-02164-6.

[Sin21] Corwin Sinnamon. *Fast and Simple Edge-Coloring Algorithms*. 2021. arXiv: 1907.03201 [cs.DS].

[Spe77] Joel Spencer. "Asymptotic lower bounds for Ramsey functions". In: *Discrete Mathematics* 20 (1977), pp. 69–76. ISSN: 0012-365X. DOI: https://doi.org/10.1016/0012-365X(77)90044-9. URL: https://www.sciencedirect.com/science/article/pii/0012365X77900449.

[SWW12] Qiaojun Shu, Weifan Wang, and Yiqiao Wang. "Acyclic edge coloring of planar graphs without 5-cycles". In: *Discrete Applied Mathematics* 160.7 (2012), pp. 1211–1223. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2011.12.016. URL: https://www.sciencedirect.com/science/article/pii/S0166218X11005051.

[Viz65] Vadim G Vizing. "Critical graphs with given chromatic class (in Russian)". In: *Metody Discret. Analiz.* 5 (1965), pp. 9–17.

[Wan20] Cathy Wang. *The Lovász local lemma, k-coloring, and applications in graph theory*. 2020. URL: https://math.uchicago.edu/~may/REU2020/REUPapers/Wang,Cathy.pdf (visited on 08/17/2023).

[WSW13] Yiqiao Wang, Qiaojun Shu, and Weifan Wang. "The acyclic edge coloring of planar graphs without a 3-cycle adjacent to a 4-cycle". In:

Discrete Applied Mathematics 161.16 (2013), pp. 2687–2694. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2013.04.004. URL: https://www.sciencedirect.com/science/article/pii/S0166218X13001911.

[WZ14] Tao Wang and Yaqiong Zhang. "Acyclic edge coloring of graphs". In: *Discrete Applied Mathematics* 167 (Apr. 2014), pp. 290–303. DOI: 10.1016/j.dam.2013.12.001. URL: https://doi.org/10.1016%2Fj.dam.2013.12.001.