Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Marek Cygan, Marcin Pilipczuk
Nr albumu: 209457, 214563

Nowe algorytmy rozwiązujące problem szerokości grafu

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dra Łukasza Kowalika
Instytut Informatyki UW

Maj 2008

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

W poniższej pracy przedstawiamy znane wcześniej oraz nowe wyniki dotyczące dokładnego i przybliżonego rozwiązywania NP-zupełnego problemu Bandwidth. W szczególności przedstawiamy nowy algorytm dokładnie rozwiązujący problem Bandwidth w czasie $O^*(5^n)$ i pamięci $O^*(2^n)$ oraz nowe podejście do aproksymacji ze stałym współczynnikiem tego problemu. Pokazujemy również kilka specjalnych przypadków, które da się łatwiej rozwiązać.

Słowa kluczowe

NP-zupełny, dokładny algorytm, aproksymacja, Bandwidth

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

F. Theory of Computation
F.2 Analysis of Algorithms and Problem Complexity
F.2.2 Nonnumerical Algorithms and Problems

Tytuł pracy w języku angielskim

New algorithms solving the Bandwidth problem

Spis treści

Podziękowania

Dziękujemy naszym narzeczonym za wyrozumiałość podczas pisania pracy w okresie przedślubnym. Dziękujemy również promotorowi i recenzentom za pomoc, a także cenne uwagi do naszej pracy.

Rozdział 1

Wprowadzenie

1.1. Algorytmy wykładnicze

Problemy NP-trudne znajdują się w centrum uwagi algorytmiki teoretycznej od bardzo długiego czasu. Pojawiają się naturalnie w wielu zastosowaniach, a przy tym prawdopodobnie nie da się ich rozwiązać w czasie wielomianowym — czyli efektywnie. Dlatego poszukuje się różnych sposobów ominięcia problemu.

Jednym ze starych sposobów jest poszukiwanie *algorytmów aproksymacyjnych* dla optymalizacyjnych wersji problemów NP-trudnych. W klasycznym rozumieniu, poszukujemy algorytmu działającego w czasie wielomianowym, który dla danego problemu optymalizacyjnego zwraca odpowiedź, która nie będzie *dużo gorsza* od optymalnej, np. wielkość optymalizowana będzie mniejsza lub większa o co najwyżej stały czynnik. Dla przykładu, dla problemu Vertex Cover (mając dany nieskierowany graf $G = (V, E)$ znaleźć jak najmniejszy zbiór $A \subset V$ taki, że każda krawędź G ma przynajmniej jeden koniec w A) łatwo jest aproksymować ze stałym współczynnikiem. Wystarczy znaleźć dowolne maksymalne w sensie zawierania skojarzenie M w grafie G i do zbioru A wziąć wszystkie skojarzone wierzchołki. Każda krawędź grafu ma przynajmniej jeden koniec w A, gdyż inaczej moglibyśmy powiększyć M o tą krawędź. Z drugiej strony, optymalny (najmniejszy) zbiór A_{OPT} musi zawierać po przynajmniej jednym końcu każdej krawędzi z M, czyli mieć moc co najmniej równą połowie mocy A. Ten algorytm wobec tego jest wielomianowym algorytmem 2-aproksymacyjnym dla problemu Vertex Cover.

Niestety, wiele problemów nie można skutecznie aproksymować ze stałym współczynnikiem. Przykładowo Johan Håstad (zob. [10]) pokazał, że problem zbioru niezależnego, który będziemy oznaczać przez Independent Set, jest nieaproksymowalny w czasie wielomianowym ze współczynnikiem $n^{1-\epsilon}$ dla żadnego $\epsilon > 0$ przy założeniu, że NP $\neq$ ZPP. Analogiczny rezultat dla problemu kolowania wierzchołkowego otrzymali Uriel Feige i Joe Kilian (zob. [6]). Unger [16] pokazał, że problem omawiany w tej pracy — Bandwidth — nie jest aproksymowalny ze stałym współczynnikiem w czasie wielomianowym przy założeniu, że NP $\neq$ ZPP.

W tej sytuacji, poza poszukiwaniem algorytmów wielomianowych o jak najlepszym współczynniku aproksymacji, obiecującym wydaje się poświęcenie złożoności na rzecz lepszego współczynnika aproksymacji. W problemie Bandwidth, o ile nie można raczej się spodziewać algorytmu wielomianowego aproksymującego ze stałym współczynnikiem, o tyle można rozważać algorytm wykładniczy, aproksymujący ze stałym współczynnikiem, lecz o złożoności istotnie lepszej niż najlepszy znany algorytm dokładny. W rozdziale 4 pokazujemy taki nowy algorytm.

Poza wykładniczymi algorytmami aproksymacyjnymi, można się pokusić o jak najszybsze

lecz *dokładne* (i w związku z tym zapewne wykładnicze) rozwiązanie problemu NP-trudnego. W przypadku, gdy prawdopodobnie dla tych problemów nie istnieją wielomianowe algorytmy dokładne, algorytmy wykładnicze pozostają jedynym wyjściem, jeśli zależy nam na dokładności rozwiązania. Dla niejednego problemu NP-zupełnego (np. problemu komiwojażera), współczesne algorytmy dokładne potrafią rozwiązywać w praktyce problem dla całkiem dużych instancji (np. dla grafów 1000-wierzchołkowych w problemie komiwojażera). Dodatkowo, algorytmy dokładne są często inspiracją dla algorytmów aproksymacyjnych. Poszukiwanie jak najszybszych dokładnych algorytmów dla problemów NP-trudnych ostatnio stało się bardzo popularne w środowisku badaczy, czego przykładem są prace przeglądowe Woegingera [17, 18, 19] czy Fomina, Grandoni i Kratsch [7]. W szczególności, problem rozwiązany w tej pracy, czyli znalezienia szybszego algorytmu dokładnego dla problemu Bandwidth, jest wymieniony jako otwarty problem nr 45 w pracy przeglądowej Woegingera [17].

1.2. Problem szerokości grafu (Bandwidth)

Problem Bandwidth można sformułować następująco: mając dany spójny, nieskierowany graf $G = (V, E)$ znaleźć takie ustawienie wierzchołków z V — bijekcję $\pi : V \to \{1, 2, \ldots, |V|\}$, dla której wartość $\max_{(u,v)\in E} |\pi(u) - \pi(v)|$ jest najmniejsza możliwa. Wartość tę nazywamy *szerokością* (ang. bandwidth) grafu G i oznaczamy $\mathrm{bw}(G)$. Oznaczmy również $n = |V|$. Problem Bandwidth można rozpatrywać również w wersji decyzyjnej: mając daną liczbę b rozstrzygnąć, czy istnieje takie ustawienie π, dla którego $\max_{(u,v)\in E} |\pi(u) - \pi(v)| \leqslant b$. Problem minimalizacyjny i decyzyjny są praktycznie równoważne. W rozdziale 2 definiujemy dokładnie problem Bandwidth i pokazujemy, na czym polega ta równoważność.

Mimo, że dla nielicznych klas grafów problem jest rozwiązywalny w czasie wielomianowym [1, 11], problem Bandwidth jest NP-zupełny nawet przy założeniu, że G jest drzewem [9, 12]. Co więcej, Unger [16] pokazał, że problem Bandwidth nie należy do klasy APX nawet przy założeniu, że graf G jest drzewem specjalnego typu. Najlepszy obecnie znany algorytm aproksymacyjny działający w czasie wielomianowym, opracowany przez Uriela Feige [4], ma współczynnik aproksymacji $O(\log^3 n\sqrt{\log n \log\log n})$.

W celu przedstawienia tła problemu, w rozdziale 3 pokazujemy ciekawy algorytm dynamiczny Saxe'a [15], rozwiązący problem decyzyjny Bandwidth w czasie i pamięci $O^*(n^b 2^{2b})$. Przedstawiamy również najlepszy znany dotychczas wynik — algorytm Feige i Kiliana [5] działający w czasie $O^*(10^n)$ i w pamięci wielomianowej. Przytaczamy również precyzyjnie wyniki wspomniane w poprzednim akapicie.

1.3. Nasze wyniki

W pracy przedstawiamy nowe wyniki dotyczące problemu Bandwidth. W rozdziale 4 pokazujemy klasę algorytmów $(4k-1)$-aproksymacyjnych działających w czasie $O^*(2^{\frac{n}{k}})$ dla dowolnego k całkowitego dodatniego oraz pokazujemy tzw. redukcję, czyli algorytm tworzący dwa razy mniejszą instancję problemu, aproksymując rozwiązanie ze stałą 9. W rozdziale 5 przedstawiamy dwa algorytmy rozwiązujące problem Bandwidth działające w czasie $O^*(5^n)$, z czego jeden potrzebuje $O^*(2^n)$, natomiast drugi $O^*(4^n)$ pamięci. Później pokazujemy, że, używając nietrywialnej metody szacowania złożoności, drugi algorytm działa on w czasie $O(4.829^n)$, lecz niestety złożoność pamięciowa wciąż wynosi $O^*(4^n)$. W rozdziale 6 pokazujemy, że problem decyzyjny Bandwidth można rozwiązać szybciej w przypadkach $b \geqslant \frac{n}{2}$ i $\frac{n}{2} > b \geqslant \frac{n}{3}$. Rozdział 7 jest poświęcony dyskusji na temat ewentualnych dalszych możliwości atakowania problemu Bandwidth i poprawiania zamieszczonych w tej pracy wyników.

Ten rozdział zawiera również informację, który z autorów jest odpowiedzialny za wymyślenie którego z nowych wyników opisanych w tej pracy.

Pierwszy z opisywanych algorytmów dokładnych oraz przypadki szczególne, opisane w rozdziale 6, będą przedstawiane przez nas na konferencji 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2008), która odbędzie się na przełomie czerwca i lipca 2008 [13]. Redukcja problemu bandwidth, opisana w rozdziale 4, pochodzi z pracy [14].

Rozdział 2

Notacja i podstawowe fakty

2.1. Problem minimalizacyjny Bandwidth

Przyjmijmy następującą definicję minimalizacyjnego problemu Bandwidth:

Definicja 2.1. *Problemem minimalizacyjnym* Bandwidth nazwiemy następujący problem: mając dany spójny i nieskierowany graf $G = (V, E)$ znaleźć najmniejszą liczbę naturalną b, dla której istnieje bijekcja $\pi : V \to \{1, 2, \ldots, |V|\}$ taka, że dla każdej krawędzi $uv \in E$ zachodzi $|\pi(u) - \pi(v)| \leqslant b$.

Funkcję π będziemy nazywać *ustawieniem* wierzchołków grafu G. Dla krawędzi uv wartość $|\pi(u) - \pi(v)|$ będziemy nazywać *długością* krawędzi uv w ustawieniu π. Będziemy mówić, że w ustawieniu π wierzchołek u jest na lewo (prawo) od wierzchołka v, jeśli $\pi(u) < \pi(v)$ ($\pi(u) > \pi(v)$). Będziemy mówić, że $\mathrm{bw}(\pi) = \max_{uv \in E} |\pi(u) - \pi(v)|$ jest *szerokością* ustawienia π. *Szerokością* grafu spójnego G będziemy nazywać odpowiedź w problemie Bandwidth, czyli minimalną szerokość ustawień π po wszystkich ustawieniach, i będziemy ją oznaczać $\mathrm{bw}(G)$. Ustawienie π nazwiemy *optymalnym*, jeśli $\mathrm{bw}(\pi) = \mathrm{bw}(G)$. Wartość $\pi(v)$ będziemy nazywali *pozycją* wierzchołka v w ustawieniu π, natomiast przez $\mathbf{ciag}(\pi)$ będziemy oznaczać ciąg $\pi^{-1}(1), \ldots, \pi^{-1}(n)$.

Uwaga 2.2. Zauważmy, że dodanie założenia o spójności grafu nie powoduje utraty ogólności problemu. Jeśli mamy ustawienie π dla grafu niespójnego G, to ustawienie π' w którym wierzchołki tej samej spójnej składowej w G występują w tej samej kolejności co w π, lecz najpierw występują wszystkie wierzchołki pierwszej spójnej składowej, później drugiej, później trzeciej itd., ma szerokość nie większą niż $\mathrm{bw}(\pi)$. W związku z tym w grafie niespójnym można osobno rozważać szerokość każdej spójnej składowej.

Uwaga 2.3. Trywialny algorytm, sprawdzający dla każdego ustawienia (permutacji) π wartość $\mathrm{bw}(\pi)$ i wybierający minimum, jest algorytmem rozwiązującym w czasie $O((|V| + |E|) \cdot |V|!)$ dokładnie problem minimalizacyjny Bandwidth.

Przyjmijmy więc, że wszystkie grafy występujące w dalszej części pracy są spójne.

2.2. Problem decyzyjny Bandwidth

Przyjmijmy następującą definicję decyzyjnego problemu Bandwidth:

Definicja 2.4. *Problemem decyzyjnym* Bandwidth nazwiemy następujący problem: mając dany spójny i nieskierowany graf $G = (V, E)$ oraz liczbę naturalną $1 \leqslant b \leqslant |V| - 1$ zdecydować, czy istnieje ustawienie π o szerokości nie większej niż b.

Fakt 2.5. *Problem decyzyjny* BANDWIDTH *należy do klasy* NP.

Dowód. Niech świadkiem dla odpowiedzi pozytywnej będzie ustawienie π o szerokości nie większej niż b. Ustawienie jest wielkości liniowej od wielkości grafu. Przy tym sprawdzenie, czy jest to rzeczywiście ustawienie (permutacja wierzchołków) oraz czy długość każdej krawędzi w tym ustawieniu nie przekracza b jest prostym wielomianowym algorytmem. □

Zachodzi przy tym następujące twierdzenie, którego tutaj nie udowodnimy:

Twierdzenie 2.6. *Problem decyzyjny* BANDWIDTH *jest* NP*-zupełny.*

Stąd wniosek, że nie powinniśmy spodziewać się znalezienia wielomianowego algorytmu rozwiązującego problem BANDWIDTH. W tej pracy skupimy się, między innymi, na znalezieniu jak najlepszego (pod względem głównie złożoności czasowej) dokładnego algorytmu rozwiązującego ten problem. Oczywiście, będzie on miał złożoność wykładniczą od rozmiaru problemu.

W pracy zajmiemy się również aproksymacją minimalizacyjnego problemu BANDWIDTH.

Definicja 2.7. Algorytm dla minimalizacyjnego problemu BANDWIDTH nazwiemy *f-aproksymacyjnym*, jeśli dla grafu wejściowego G zwraca on odpowiedź (domniemaną szerokość grafu) leżącą w przedziale $[\mathrm{bw}(G), f(G) \cdot \mathrm{bw}(G)]$. Jeśli f jest funkcją stałą, nazwiemy ten algorytm *aproksymacyjnym ze stałym współczynnikiem.*

2.3. Równoważność problemów minimalizacyjnego i decyzyjnego BANDWIDTH

Zauważmy, iż w oczywisty sposób algorytm rozwiązujący problem minimalizacyjny BANDWIDTH rozwiązuje również problem decyzyjny. Zauważmy również, że zachodzi następujący fakt:

Fakt 2.8. *Jeśli dany jest algorytm A rozwiązujący w czasie $O(f(|V|, |E|))$ problem decyzyjny* BANDWIDTH*, to istnieje algorytm A' rozwiązujący w czasie $O(f(|V|, |E|) \log |V|)$ problem minimalizacyjny* BANDWIDTH*. Wystarczy wyszukać binarnie odpowiedź.*

W związku z tym, w świetle wystepujących w tej pracy złożoności, przeważnie wykładniczych, problemy minimalizacyjny i decyzyjny będą dla nas właściwie równoważne. Algorytmy dokładne będą rozwiązywać problem decyzyjny. Algorytmy aproksymacyjne oczywiście mają sens tylko dla problemu minimalizacyjnego.

Zauważmy również następujący fakt:

Fakt 2.9. *Jeśli dany jest algorytm A rozwiązujący w czasie $f(|V|, |E|)$ problem decyzyjny* BANDWIDTH*, to istnieje algorytm B, który w czasie $O(|V|^2 f(|V|, |E|) + |V| \cdot (|E| + |V|))$ rozwiązuje problem minimalizacyjny* BANDWIDTH *oraz znajduje optymalne ustawienie π, tj. ustawienie spełniające $\mathrm{bw}(\pi) = \mathrm{bw}(G)$.*

Dowód. Dany jest egzemplarz problemu $G = (V, E)$. Najpierw w czasie $O(\log n f(|V|, |E|))$ znajdujemy liczbę $\mathrm{bw}(G)$. Zauważmy, iż dodając krawędzie do grafu nie możemy zmniejszyć jego szerokości. Dla każdej pary wierzchołków u i v takich, że $u \neq v$ i $uv \notin E$:

- dodajemy do G krawędź uv, otrzymując graf G',
- za pomocą algorytmu A sprawdzamy, czy $\mathrm{bw}(G') = \mathrm{bw}(G)$, czy też $\mathrm{bw}(G') > \mathrm{bw}(G)$,

- jeśli $\mathrm{bw}(G') = \mathrm{bw}(G)$ zostawiamy w G krawędź uv, w przeciwnym razie usuwamy ją.

Po przejściu wszystkich par u, v, których jest $O(|V|^2)$, otrzymujemy graf $\bar{G}$ zawierający jako podgraf graf G, spełniający $\mathrm{bw}(\bar{G}) = \mathrm{bw}(G)$ oraz, przy dodaniu do niego dowolnej krawędzi, szerokość grafu się zwiększa. Zauważmy, że jeśli π jest optymalne dla $\bar{G}$, to jest również optymalne dla G. Graf $\bar{G}$ ma specyficzną strukturę i nie jest trudno znaleźć dla niego optymalne ustawienie.

Niech π będzie optymalnym ustawieniem dla $\bar{G}$. Oznaczmy $n = |V|$ i niech $v_1, \ldots, v_n$ będą wierzchołkami $\bar{G}$ tak ponumerowanymi, że $\pi(v_i) = i$. Wówczas v_i może sąsiadować jedynie z wierzchołkami (o ile istnieją) $v_{i-b}, v_{i-b+1}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{i+b}$. Co więcej, jako że do $\bar{G}$ nie można dołożyć żadnej krawędzi nie powiększając szerokości grafu, wszystkie te krawędzie muszą istnieć. Innymi słowy, $(v_i, v_j) \in \bar{E}$ wtedy i tylko wtedy gdy $|i - j| \leqslant \mathrm{bw}(G)$ oraz $i \neq j$.

Na mocy tych obserwacji skonstruujemy prosty algorytm znajdujący optymalne ustawienie π dla grafu $\bar{G}$. Jeśli $\mathrm{bw}(G) = |V| - 1$, to dowolne ustawienie π jest optymalne. W przeciwnym wypadku wierzchołki v_1 i v_n są jedynymi wierzchołkami o stopniu b. Przyjmijmy dowolny z nich jako v_1 — zauważmy, że π jest optymalnym ustawieniem wtedy i tylko wtedy gdy π' zdefiniowany przez $\pi'(v) = n + 1 - \pi(v)$ (czyli odwrócone ustawienie π) jest optymalne. Usuwając v_1, zostanie w grafie tylko jeden wierzchołek o stopniu $\mathrm{bw}(G)$, który sąsiadował uprzednio z v_1 — będzie to jedyny kandydat na v_2. Tak możemy kontynuować, aż zostanie nam $b + 1$ wierzchołków, tworzących klikę — wszystkie są o stopniu b. Teraz dla każdego z tych $b + 1$ wierzchołków znajdujemy sąsiada, ustawionego najbardziej na lewo (czyli o najmniejszym indeksie) i sortujemy je względem tego indeksu (w przypadku remisu, ustawiamy wierzchołki w dowolnej kolejności). Łatwo zauważyć, że to jest ustawienie grafu $\bar{G}$ o szerokości b.

Znajdowanie π działa w czasie $O(|V| \cdot (|V| + |E|))$, dla każdego wierzchołka musimy przejrzeć wszystkie pozostałe wierzchołki i krawędzie. To ustawienie jest też optymalnym ustawieniem dla G.

Ostatecznie, w czasie $O(|V|^2 f(|V|, |E|) + |V| \cdot (|E| + |V|))$, czyli dokładając co najwyżej wielomianowy współczynnik do złożoności, potrafimy znaleźć ustawienie optymalne na podstawie algorytmu rozwiązującego decyzjny problem Bandwidth. □

2.4. Pozostała notacja

W pracy będziemy używali notacji złożoności $O^*()$. Jest to notacja $O()$ z pominięciem czynników wielomianowych, które są mniej istotne od czynników wykładniczych.

Rozdział 3

Znane wyniki

W tym rozdziale przedstawimy znane wcześniej wyniki dotyczące problemu Bandwidth. Pokażemy dwa algorytmy rozwiązujące dokładnie problem decyzyjny: jeden działa w czasie $O^*(10^n)$, drugi — $O^*(n^b 2^{2b})$. Zademonstrujemy, jak bazując na narzędziach wypracowanych przez pierwszy z nich stworzyć algorytmy ze stałym współczynnikiem aproksymacji.

Celem tego rozdziału jest pokazanie różnych sposobów podejścia do rozwiązywania problemu Bandwidth oraz wprowadzenie pewnych narzędzi potrzebnych w dalszej pracy.

3.1. Podział na kubełki

Większość algorytmów wykładniczych, zarówno aproksymacyjnych jak i dokładnych, rozwiązujących problem Bandwidth, korzysta z operacji, którą możemy nazwać *podziałem na kubełki*.

Załóżmy, że mamy dany egzemplarz problemu decyzyjnego Bandwidth, czyli graf spójny $G = (V, E)$, $|V| = n$ oraz liczbę b. Oznaczmy przez N zbiór $\{1, 2, \ldots, n\}$.

Definicja 3.1. *Podziałem wierzchołków na kubełki* nazwiemy dowolne przyporządkowanie $p : V \to \{0, 1\}^N$, czyli każdemu wierzchołkowi przyporządkowywujemy jakiś podzbiór N. Ustawienie π jest *zgodne* z podziałem p, jeśli $\pi(v) \in p(v)$ dla każdego wierzchołka v.

Intuicyjnie rzecz ujmując, wiele algorytmów będzie działało zgodnie z następującym schematem: najpierw generujemy pewną liczbę (zazwyczaj wykładniczą) podziałów wierzchołków na kubełki, a następnie dla każdego podziału szukamy ustawienia π zgodnego z tym podziałem, które ma szerokość nie większą niż b. Wartości $p(v)$ zazwyczaj będą przedziałami w zbiorze N.

Zauważmy, że ze zbioru $p(v)$ możemy usunąć pozycje, na których na pewno nie postawimy wierzchołka v. Są to takie pozycje, dla których istnieje krawędź uv powodująca, że dla wierzchołka u nie ma pozycji w $p(u)$ w odległości co najwyżej b.

Definicja 3.2. Podział na kubełki p nazwiemy *przyciętym*, jeśli spełniona jest następująca zależność: jeśli $uv \in E$, to dla każdego $i \in p(v)$ istnieje $j \in p(u)$ taki, że $|i - j| \leqslant b$.

Fakt 3.3. *Jeśli p jest podziałem na kubełki, to podział q zdefiniowany następująco:*

$$q(v) = \{i \in p(v) : \forall_{uv \in E} \exists_{j \in p(u)} |i - j| \leqslant b\}$$

jest przycięty i spełnia następujący warunek: dla każdego ustawienia π o szerokości nie większej niż b zgodnego z p, π jest zgodne z q.

Dowód. Cała trudność w tym fakcie leży w zrozumieniu definicji. Dla dowolnego wierzchołka v, dla każdej krawędzi uv, $\pi(u) \in p(u)$ i $\pi(v) \in p(v)$, bo π jest zgodne z p, oraz $|\pi(u) - \pi(v)| \leqslant b$. Wobec tego warunek z definicji $q(v)$ jest spełniony, czyli $\pi(v) \in q(v)$, czyli π jest zgodne z q. □

Wobec tego, możemy się ograniczyć do rozważania wyłącznie ustawień przyciętych. Zauważmy, iż z dowolnego ustawienia p w czasie wielomianowym jesteśmy w stanie uzyskać ustawienie przycięte q, zgodnie z definicją z faktu 3.3.

Warto zauważyć następujący prosty fakt:

Fakt 3.4. *Istnieje algorytm działający w czasie wielomianowym, który dla danego egzemplarza problemu decyzyjnego lub minimalizacyjnego* Bandwidth *i dla podziału na kubełki p stwierdzi, czy istnieje ustawienie π zgodne z p (ale niekoniecznie spełniające* $\mathrm{bw}(\pi) \leqslant b$ *w przypadku problemu decyzyjnego).*

Dowód. Skonstruujmy następujący graf dwudzielny: $V_1 = V$, $V_2 = \{1, 2, \ldots, n\}$ i (v, i) jest krawędzią wtedy i tylko wtedy gdy $i \in p(v)$. Wówczas ustawienie π odpowiada doskonałemu skojarzeniu w przedstawionym grafie, które możemy znaleźć przy pomocy klasycznego algorytmu. [1] □

Definicja 3.5. *Klasycznym podziałem na kubełki* będziemy nazywać podział na kubełki p spełniający następujące warunki:

1. liczba $\bar{b} = b$ lub $\bar{b} = b + 1$
2. wartościami funkcji p są zbiory postaci $\{i\bar{b}+1, i\bar{b}+2, \ldots, (i+1)\bar{b}\} \cap N$, czyli przedziały zawierające $\bar{b}$ elementów, za wyjątkiem ewentualnie ostatniego;
3. jeśli $uv \in E$, to $p(u)$ i $p(v)$ są tym samym lub sąsiednimi przedziałami; dla $\bar{b} = b$ oznacza to, że podział jest przycięty;
4. dla każdego dopuszczonego przedziału I zachodzi $|p^{-1}(I)| = |I|$.

Dopuszczalne wartości funkcji p w podziale klasycznym nazywać będziemy też *kubełkami klasycznymi*. Gdy nie będzie ustalonego żadnego podziału, przez *kubełek* będziemy rozumieli kubełek klasyczny.

Uwaga 3.6. Zauważmy, że w definicji klasycznego podziału na kubełki dopuszczalne wartości podziału p są rozłączne. Wobec tego ostatni punkt definicji jest warunkiem koniecznym do istnienia ustawienia zgodnego z tym podziałem — w przedziale I musimy ustawić dokładnie $|I|$ wierzchołków.

Twierdzenie 3.7. *Istnieje co najwyżej $n3^n$ klasycznych podziałów na kubełki dla danego egzemplarza problemu decyzyjnego* Bandwidth (G, b). *Co więcej, da się je wygenerować w czasie $O^*(3^n)$ i w wielomianowej pamięci (nie licząc wypisywanego wyniku).*

Dowód. Opiszemy algorytm generujący wszystkie klasyczne podziały. Weźmy dowolny wierzchołek v i umieśćmy go w jednym z kubełków — to możemy zrobić na co najwyżej n sposobów. Dalej, mając już określone p na pewnym niepustym podzbiorze $A \subset V$, wybieramy wierzchołek $v \in V \setminus A$ taki, że istnieje $uv \in E$, gdzie $u \in A$ — innymi słowy wybieramy pewny

[1] Przy założeniu, że wartości funkcji p są przedziałami jesteśmy w stanie skonstuować prosty, bardziej efektywny, zachłanny algorytm, który dla kolejnych pozycji będzie wybierał nieużyty dotychczas wierzchołek v, którego przedział $p(v)$ się najszybciej kończy. Dowód poprawności tego algorytmu pozostawiamy czytelnikowi.

nieprzydzielony wierzchołek v, którego sąsiad u już został przydzielony do jakiegoś kubełka. Zauważmy, że aby spełnić punkt 3 definicji podziału klasycznego, v może być umieszczony jedynie w tym samym kubełku co wierzchołek u lub w którymś z dwóch sąsiednich. W momencie, gdy $A = V$, sprawdzamy, czy warunek 3 i 4 jest spełniony dla wygenerowanego podziału i jeśli tak, wypisujemy ten podział.

Pierwszy wierzchołek możemy umieścić na n sposobów w kubełkach (jest co najwyżej n kubełków), każdy kolejny na 3 sposoby — w sumie wygenerujemy co najwyżej $n3^n$ klasycznych podziałów. Oczywiste jest, że algorytm wygeneruje wszystkie klasyczne podziały. Wybór wierzchołka v w algorytmie oraz sprawdzenie na koniec warunków definicji można zrobić w czasie wielomianowym, więc algorytm działa w czasie $O^*(3^n)$. Podczas działania algorytmu musimy jedynie pamiętać stos rekurencji — dla każdego wierzchołka w A zapisujemy, do którego kubełka jest on przydzielony. Algorytm działa więc w pamięci wielomianowej. □

Warto zdefiniować tu jeszcze jeden, podobny rodzaj podziału na kubełki. Przyda nam się on w dalszej części tego rozdziału.

Definicja 3.8. *Połówkowym podziałem na kubełki* będziemy nazywać podział na kubełki p spełniający następujące warunki:

1. oznaczmy $s = \lfloor \frac{b}{2} \rfloor$; wartościami funkcji p są niepuste zbiory postaci $\{ib+1, ib+2, \ldots, ib+s\} \cap N$ oraz $\{ib+s+1, ib+s+2, \ldots, (i+1)b\} \cap N$, czyli przedziały na przemian długości $\lfloor \frac{b}{2} \rfloor$ i $\lceil \frac{b}{2} \rceil$, za wyjątkiem ewentualnie krótszego ostatniego przedziału;

2. podział jest przycięty; zauważmy, że tutaj oznacza to, że jeśli $uv \in E$, to $p(u)$ i $p(v)$ to albo ten sam kubełek, albo dwa sąsiednie kubełki, albo dwa kubełki mające wspólny kubełek-sąsiada pomiędzy nimi;

3. dla każdego dopuszczonego przedziału I zachodzi $|p^{-1}(I)| = |I|$.

Podobnie jak w przypadku podziału klasycznego, tutaj punkt 3 spełnia rolę prostego warunku koniecznego dla istnienia ustawienia zgodnego z podziałem. Zachodzi podobne twierdzenie jak dla podziału klasycznego:

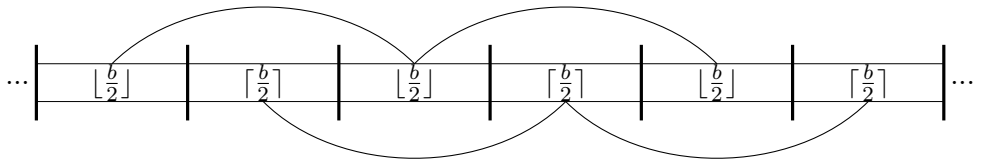
Twierdzenie 3.9. *Istnieje co najwyżej $n5^n$ połówkowych podziałów na kubełki dla danego egzemplarza problemu decyzyjnego* Bandwidth (G, b). *Co więcej, da się je wygenerować w czasie $O^*(5^n)$. Generacja, nie licząc wypisywanego wyniku, potrzebuje tylko wielomianowej pamięci.*

Dowód. Algorytm generujący podziały jest całkowicie analogiczny do przedstawionego przy klasycznych podziałach. Jedyną różnicą jest to, że dla wierzchołka v mamy pięć możliwości przydziału go do kubełka: jeśli u jest przydzielonym już sąsiadem v, to v możemy przydzielić do tego samego kubełka, do sąsiada $p(u)$ lub do sąsiada sąsiada $p(u)$. □

3.2. Algorytm $O^*(10^n)$

Uriel Feige i Joe Killian [5], na bazie połówkowego podziału na kubełki (definicja 3.8) stworzyli algorytm działający w czasie $O^*(10^n)$ i rozwiązujący decyzyjny problem Bandwidth. Algorytm opiera się na następującym spostrzeżeniu:

Uwaga 3.10. Dowolne ustawienie π zgodne z ustawieniem połówkowym p spełnia warunek $|\pi(u) - \pi(v)| \leqslant b$ na każdej krawędzi $uv \in E$ łączącej wierzchołki z tego samego lub sąsiednich



Rysunek 3.1: Pogrubione pionowe kreski pionowe oddzielają kolejne kubełki wielkości $\frac{b}{2}$. Istotne krawędzie łączą jedynie kubełki w odległości dwa.

kubełków podziału. Innymi słowy, przy konstrukcji ustawienia π jedynymi istotnymi krawędziami, są te łączące niesąsiednie kubełki, mające pomiędzy sobą wspólny kubełek-sąsiada. Co więcej, usuwając pozostałe (nieistotne) krawędzie, problem rozpada się na dwa niezależne podgrafy — krawędzie łączą tylko co drugi kubełek (zob. rys. 3.1).

Algorytm Feige i Killiana, dla każdego połówkowego podziału (które zgodnie z twierdzeniem 3.9 możemy generować on-line w czasie $O^*(5^n)$ i których jest nie więcej niż $n5^n$), rozbija problem na dwa podproblemy i je próbuje rozwiązać, tj. skonstruować zgodne z podziałem ustawienie π spełniające $\mathrm{bw}(\pi) \leqslant b$. W każdym podproblemie dzielimy każdy kubełek na dwie połowy i na każdy możliwy sposób przydzielamy wierzchołki do tych dwa razy mniejszych kubełków, tworząc dokładniejsze podziały p'. Zauważmy, że jeśli $uv \in E$ jest istotną krawędzią w podziale p i $p(u)$ jest na lewo od $p(v)$, to: po pierwsze, jednocześnie u nie może być przydzielone do lewej połówki kubełka $p(u)$ i v do prawej połówki $p(v)$ — odległość między tymi przedziałami jest większa niż b; po drugie, jeśli u zostanie przydzielone do prawej połówki kubełka $p(u)$, a v do lewej połówki kubełka $p(v)$, to krawędź uv na pewno nie będzie miała długości większej niż b. Wobec tego, usunąwszy takie nieistotne krawędzie, po podziale każdy podproblem rozbija nam się na dwa niezależne mniejsze problemy: wierzchołki umieszczone w lewych połówkach przedziałów oraz wierzchołki umieszczone w prawych połówkach przedziałów. Jeśli $T(n)$ jest czasem rozwiązania podproblemu wielkości n, to mamy równanie rekurencyjne:

$$T(n) = 2^n \cdot 2 \cdot T\left(\frac{n}{2}\right)$$

co daje rozwiązanie $T(n) = O^*(4^n)$. W naszym algorytmie dla każdego z co najwyżej $n5^n$ połówkowych podziałów rozwiązujemy dwa egzemplarze podproblemu, obie wielkości $\frac{n}{2}$, co daje w sumie złożoność $O^*(5^n \cdot 4^{\frac{n}{2}}) = O^*(10^n)$.

3.3. Algorytm $O^*(n^b 2^{2b})$

Całkowicie odmienne podejście przyjął James Saxe [15] w swoim algorytmie rozwiązującym problem decyzyjny Bandwidth w czasie $O(n^{b+1}2^{2b})$. Algorytm ten jest przykładem programowania dynamicznego, ze stosunkowo sprytną analizą. Ustalmy egzemplarz decyzyjnego problemu Bandwidth $(G = (V, E), b)$ i oznaczmy $n = |V|$.

Definicja 3.11. *Poprawnym podziałem ustawienia* nazwiemy podzbiór $A \subset V$ i ciąg różnych wierzchołków $v_1, v_2, \ldots, v_b$ należących do $V \setminus A$ takie, że nie ma krawędzi w grafie łączącej A z $V \setminus A \setminus \{v_1, v_2, \ldots, v_b\}$. Co więcej, zbiór $\{v_1, \ldots, v_b\}$ ma co najwyżej $2b$ sąsiadów w zbiorze $V \setminus \{v_1, \ldots, v_b\}$.

Definicja 3.12. Powiemy, że poprawny podział $(A, (v_1, \ldots, v_b))$ *sąsiaduje* z poprawnym podziałem $(B, (v'_1, v'_2, \ldots, v'_b))$, jeśli $B = A \cup \{v_1\}$ i $v_i = v'_{i-1}$ dla $i = 2, 3, \ldots b$.

Fakt 3.13. *Jeśli π jest ustawieniem i $\mathrm{bw}(\pi) \leqslant b$, to oznaczając $v_i = \pi^{-1}(i)$, dla każdego $0 \leqslant k \leqslant |V| - b$ para $(\{v_1, \ldots, v_k\}, (v_{k+1}, v_{k+2}, \ldots, v_{k+b}))$ jest poprawnym podziałem ustawienia i kolejne dwie takie pary sąsiadują ze sobą.*

Dowód. Dowolna krawędź łącząca $\{v_1, \ldots v_k\}$ z $\{v_{k+b+1}, \ldots, v_n\}$ byłaby dłuższa niż b w ustawieniu π, więc taka krawędź nie może istnieć. Co więcej sąsiedzi zbioru $\{v_{k+1}, \ldots, v_{k+b}\}$ spoza tego zbioru zawierają się w zbiorze $\{v_{k-b}, \ldots, v_k\} \cup \{v_{k+b+1}, \ldots, v_{k+2b}\}$, który ma moc $2b$, czyli są to poprawne podziały ustawienia. Sąsiedztwo jest oczywiste z konstrukcji. □

Fakt 3.14. *Jeśli mamy ciąg $(A_0 = \emptyset, (v_1, \ldots, v_b))$, $(A_1 = \{v_1\}, (v_2, \ldots, v_{b+1}))$, $\ldots$, $(A_{n-b} = \{v_1, \ldots, v_{n-b}\}, (v_{n-b+1}, \ldots, v_n))$ poprawnych podziałów, z których każde dwa kolejne sąsiadują ze sobą, to $\pi(i) := v_i$ jest ustawieniem spełniającym $\mathrm{bw}(\pi) \leqslant b$.*

Dowód. Załóżmy przeciwnie, czyli istnieją takie v_i i v_j, że $i + b < j$ i $v_i v_j \in E$. Spójrzmy wówczas na element $(A_i, (v_{i+1}, \ldots, v_{i+b}))$ w danym ciągu. Wówczas ta krawędź łączy $v_i \in A_i$ z $v_i \in V \setminus A_i \setminus \{v_{i+1}, \ldots, v_{i+b}\}$, czyli to nie jest poprawny podział, sprzeczność. □

Wniosek 3.15. Na podstawie powyższych faktów możemy skonstruować algorytm przeszukiwania w głąb. Będzie on zaczynał przeszukiwanie od wszystkich ustawień postaci $(\emptyset, (v'_1, \ldots, v'_b))$. W każdym odwiedzonym ustawieniu będziemy próbować je rozszerzyć o każdy nieużyty wierzchołek, sprawdzając czy nowy podział jest poprawny. Algorytm dojdzie do stanu $(A, (v_1, \ldots, v_b))$ spełniającego $|A| = n-b$ wtedy i tylko wtedy, gdy istnieje ustawienie π spełniające $\mathrm{bw}(\pi) \leqslant b$. Wówczas, na podstawie stosu wykonania algorytmu w chwili dojścia do tego podziału, można odtworzyć ustawienie π. Jeśli algorytm zapamiętuje odwiedzone stany (podziały), to działa on w czasie $O^*(S)$ gdzie S to liczba poprawnych podziałów i w pamięci $O(S + |V| + |E|)$.

W prosty sposób możemy oszacować S przez $2^n n^b$: zbiór A możemy wybrać na 2^n sposobów, ciąg $(v_1, \ldots, v_b)$ na mniej niż n^b sposobów. Możemy jednak oszacować to dokładniej, korzystając z założenia o spójności G.

Fakt 3.16. *Istnieje co najwyżej $n^b 2^{2b}$ poprawnych podziałów.*

Dowód. Weźmy dowolny ciąg $(v_1, \ldots, v_b)$ — jest ich mniej niż n^b. Rozważmy wierzchołki które sąsiadują z dowolnym z wierzchołków $v_1, \ldots, v_b$, ale nie należą do zbioru $\{v_1, \ldots, v_b\}$. Jeśli jest ich więcej niż $2b$, to ten ciąg nie może być drugą współrzędną żadnego poprawnego podziału. Jeśli jest ich nie więcej niż $2b$, spróbujmy skonstruować A — pierwszą współrzędną podziału poprawnego. Dla każdego z tych sąsiadów możemy określić, czy leży on w A czy w $V \setminus A \setminus \{v_1, \ldots, v_b\}$ — na 2^{2b} sposobów. Zauważmy, że dla pozostałych wierzchołków jest już jednoznacznie wyznaczone, czy leżą one w A czy w $V \setminus A \setminus \{v_1, \ldots, v_b\}$ — jeśli z grafu usuniemy wierzchołki $v_1, \ldots, v_b$, to każda pozostała spójna składowa może leżeć tylko w całości w A lub w całości poza A (na mocy warunków poprawnego podziału), zaś w każdej takiej spójnej składowej dla przynajmniej jednego wierzchołka (sąsiada usuniętych wierzchołków) określiliśmy, gdzie leży. Tak więc podziałów jest nie więcej niż $n^b 2^{2b}$. □

Wniosek 3.17. Opisany wcześniej algorytm przeszukiwania w głąb, działa w czasie $O((|V| + |E|)n^b 2^{2b})$. Można go poprawić technicznie do $O(n^{b+1} 2^{2b})$, lecz nie będziemy tego opisywać, gdyż nie poprawia to istotnie złożoności.

Uwaga 3.18. Zauważmy, że jesli $b < \frac{n}{3 \log_2 n}$, to $n^{b+1} 2^{2b} < n 2^n$, czyli ten algorytm jest lepszy od wszystkich innych przedstawionych w tej pracy. Problematyczne dla nas będą więc zapytania (egzemplarze problemu decyzyjnego Bandwidth) dla stosunkowo dużych b. W szczególności, w rozdziale 6 zajmiemy się m.in. przypadkiem $b \geqslant \frac{n}{2}$.

3.4. Wyniki dotyczące aproksymacji

3.4.1. Aproksymacja w czasie wielomianowym

Aby zacytować znane wyniki w tej dziedzinie, potrzebujemy następującej definicji.

Definicja 3.19. *Gąsiennicą* (ang. caterpillar tree) nazwiemy graf będący drzewem z wyróżnioną ścieżką $v_1, v_2, \ldots, v_k$ taką, że tylko wierzchołki tej ścieżki mogą mieć stopień większy niż 2.

Innymi słowy, gąsienice to drzewa, w których wszystkie wierzchołki, w których jest rozgałęzienie, układają się na jednej ścieżce. Jeśli ułożymy wierzchołki wyróżnionej ścieżki w poziomie, zaś pozostałe wierzchołki będą „zwisać w dół", rzeczywiście taka wizualizacja tego grafu będzie przypominać gąsiennicę.

Walter Unger [16] udowodnił następujące twierdzenie:

Twierdzenie 3.20 (W. Unger)**.** *Problem minimalizacyjny* Bandwidth*, nawet ograniczony do klasy grafów będących gąsiennicami, nie należy do klasy APX. Oznacza to, że o ile $P \neq NP$, nie można aproksymować ich ze stałym współczynnikiem w czasie wielomianowym.*

Nie powinniśmy się więc spodziewać istnienia wielomianowego algorytmu aproksymacyjnego ze stałym współczynnikiem dla problemu minimalizacyjnego Bandwidth. W tej pracy skupimy się na znalezieniu algorytmu, lub klasy algorytmów, aproksymacyjnych ze stałym współczynnikiem, o jak najlepszej, lecz wciąż wykładniczej, złożoności obliczeniowej.

Dla porządku zacytujemy tu najlepszy znany wynik dotyczący aproksymacji minimalizacyjnego problemu Bandwidth w czasie wielomianowym, należący do Uriela Feige [4]:

Twierdzenie 3.21 (U. Feige)**.** *Istnieje algorytm działający w czasie wielomianowym aproksymujący problem minimalizacyjny* Bandwidth *ze współczynnikiem $O(\log^3 n\sqrt{\log n \log\log n})$, gdzie n to liczba wierzchołków w grafie.*

W tej pracy nie będziemy zajmować się dalszym poprawianiem aproksymacji problemu Bandwidth w czasie wielomianowym.

3.4.2. Aproksymacja ze stałym współczynnikiem

Wysnujemy teraz proste wnioski z podziałów na kubełki.

Weźmy problem decyzyjny Bandwidth $(G = (V, E), b)$. Zwróćmy uwagę na następującą, prostą obserwację:

Definicja 3.22. *Wielkością kubełka $p(v)$* nazwiemy wartość:

$$K(p(v)) = \max_{i,j \in p(v)} |i - j|.$$

Jeśli $p(v)$ jest przedziałem $[a, b]$, to $K(p(v)) = b - a$. *Wielkością podziału p* nazwiemy wartość:

$$K(p) = \max_{v \in V} K(p(v)) = \max_{v \in V} \max_{i,j \in p(v)} |i - j|.$$

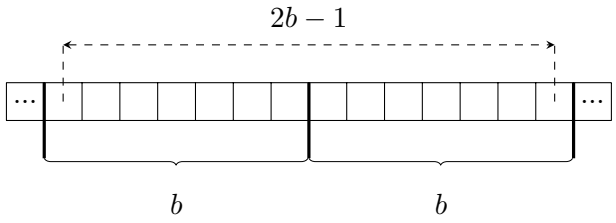
Uwaga 3.23. Wielkością podziału klasycznego jest $b - 1$. Wielkością podziału połówkowego jest $\lceil \frac{b}{2} \rceil - 1$.

Fakt 3.24. *Niech p będzie przyciętym podziałem do kubełków w zadanym problemie i niech $K = K(p)$ będzie jego wielkością. Niech π będzie dowolnym ustawieniem zgodnym z p. Wówczas* $\mathrm{bw}(\pi) \leqslant K + b$.

Dowód. Niech $uv \in E$. Weźmy dowolne ustawienie π zgodne z p. Podział jest przycięty, czyli istnieje $i \in p(u)$ spełniające $|i - \pi(v)| \leqslant b$. Z definicji K, skoro $\pi(u) \in p(u)$, $|i - \pi(u)| \leqslant K$, zatem $|\pi(u) - \pi(v)| \leqslant b + K$. □

Uwaga 3.25. Dla podziału połówkowego lub klasycznego bardzo łatwo znaleźć zgodne ustawienie π: wystarczy jakkolwiek poustawiać wierzchołki w kubełkach — różne kubełki są parami rozłączne. Co więcej, takie ustawienie zawsze istnieje.

Wniosek 3.26. Dowolne ustawienie π zgodne z klasycznym podziałem (dla $\bar{b} = b$) p_k spełnia $\mathrm{bw}(\pi) \leqslant 2b$. Istnieje więc algorytm działający w czasie $O^*(3^n)$ aproksymujący minimalizujący problem Bandwidth z współczynnikiem 2: wyszukujemy binarnie najmniejsze b' dla którego istnieje podział klasyczny (znalezienie podziału może nas kosztować $O^*(3^n)$), i bierzemy dowolne ustawienie π w tym podziale. Wówczas $b' \leqslant \mathrm{bw}(G)$, ale również $\mathrm{bw}(\pi) \leqslant 2b'$ (zob. rys. 3.2), zatem $\mathrm{bw}(\pi) \leqslant 2b' \leqslant 2\mathrm{bw}(G)$.



Rysunek 3.2: W klasycznym podziale maksymalna odległość pomiędzy wierzchołkami w sąsiednich kubełkach wynosi $2b - 1$.

Wniosek 3.27. Analogicznie, dowolne ustawienie π zgodne z połówkowym podziałem p_p spełnia $\mathrm{bw}(\pi) \leqslant \frac{3}{2}b$. Istnieje więc algorytm działający w czasie $O^*(5^n)$ aproksymujący minimalizujący problem Bandwidth ze współczynnikiem $\frac{3}{2}$: wyszukujemy binarnie najmniejsze b' dla którego istnieje podział połówkowy (znalezienie podziału może nas kosztować $O^*(5^n)$), następnie bierzemy dowolne ustawienie π zgodne z tym podziałem. Wówczas $b' \leqslant \mathrm{bw}(G)$, więc $\mathrm{bw}(\pi) \leqslant \frac{3}{2}b' \leqslant \frac{3}{2}\mathrm{bw}(G)$.

Rozdział 4

Wykładnicze algorytmy aproksymacyjne

Jedną z metod postępowania z problemami NP-trudnymi jest aproksymacja w czasie wielomianowym. Zamiast znajdować rozwiązanie optymalne chcemy otrzymać chociaż przybliżoną wartość wyniku. Niestety jednak wiele naturalnych problemów nie daje się aproksymować z zadowalającymi współczynnikami. Przykładowo Johan Håstad (zob. [10]) pokazał, że problem zbioru niezależnego, który będziemy oznaczać przez INDEPENDENT SET, jest nieaproksymowalny w czasie wielomianowym ze współczynnikiem $n^{1-\epsilon}$ dla żadnego $\epsilon > 0$ przy założeniu, że NP $\neq$ ZPP. Analogiczny rezultat dla problemu kolowania wierzchołkowego otrzymali Uriel Feige i Joe Kilian (zob. [6]).

Inne podejście do NP-trudności reprezentują algorytmy o złożoności parametryzowanej (zob. [2]). Wykładniczy czynnik czasu działania takiego algorytmu nie zależy od rozmiaru egzemplarza, lecz od stopnia jej zawiłości, takiej jak na przykład szerokość drzewowa. Stosowanie takiego podejścia pozwala nam znajdować optymalne rozwiązania jedynie dla wąskiej grupy egzemplarzy danego problemu.

Przedstawione wyżej argumenty powodują, iż istnieje potrzeba znajdowania algorytmów wykładniczych o akceptowalnie długim czasie działania. Przykładowo algorytm o złożoności $O(2^{n/r})$ może być użyteczny dla wystarczająco dużego r. Istotnie, algorytm którego czas działania wynosi $O(2^{n/50})$ może z powodzeniem rozwiązywać egzemplarz problemu dla $n \leqslant 1000$. Niestety istniejące algorytmy dokładne w rozważanej dziedzinie mają złożoność daleką od tego rzędu wielkości. Dla najbardziej badanego problemu INDEPENDENT SET najlepszy znany algorytm (zob. [8]) ma złożoność obliczeniową $O(2^{n/3.47})$. Wszystkie te czynniki motywują do znajdowania wykładniczych algorytmów aproksymacyjnych, których czas działania będzie zależał od żądanego współczynnika aproksymacji.

Jak pisaliśmy w rozdziale 3 problem BANDWIDTH nie należy do klasy APX nawet ograniczając sie do grafów będących drzewami, co oznacza, że prawdopodobnie nie da się tego problemu aproksymować ze stałem współczynnikiem w czasie wielomianowym. Ponadto najlepszy znany wielomiany algorytm ma współczynnik aproksymacji równy $O(\log^3 n\sqrt{\log n \log\log n})$.

W tym rozdziale na początku przedstawimy przykład tzw. *redukcji* problemu NP-zupełnego do mniejszego egzemplarza. Będziemy się posługiwać problemem INDEPENDENT SET, gdzie taka redukcja jest stosunkowo prosta i posłuży nam do przedstawiena idei. Następnie zajmiemy się dużo bardziej skomplikowaną redukcją dla problemu BANDWIDTH. Później spróbujemy poeksperymentować z różnymi podziałami na kubełki, otrzymując algorytm $(4k-1)$ aproksymacyjny działający w czasie $O^*(2^{\frac{n}{k}})$.

4.1. Maksymalny zbiór niezależny

Dla danego grafu nieskierowanego $G = (V, E)$ problem Independent Set polega na znalezieniu najliczniejszego podzbioru zbioru wierzchołków V, w którym żadne dwa wierzchołki nie są połączone krawędzią.

Niech $r \in \mathbb{N}$, $r \geqslant 2$. Załóżmy, że interesuje nas znalezienie zbioru niezależnego, który będzie co najwyżej r razy mniejszy od rozwiązania optymalnego, które oznaczymy jako $OPT \subseteq V$. Zauważmy, że jeśli zbiór wierzchołków V podzielimy na r części $V_1, V_2, \ldots, V_r$, takich że $\cup_i V_i = V$, to dla pewnego i zachodzi $|OPT \cap V_i| \geqslant \frac{|OPT|}{r}$, zatem maksymalny zbiór niezależny w grafie indukowanym przez V_i ma rozmiar równy co najmniej $\frac{|OPT|}{r}$.

Nasz algorytm może zatem podzielić zbiór V na r możliwie równych części, za pomocą algorytmu dokładnego znaleźć rozwiązania optymalne w każdej z nich, po czym wybrać najliczniejszy spośród znalezionych zbiorów. Algorytm ten jest r-aproksymacyjny a każdy z r nowoutworzonych egzemplarzy będzie miał rozmiar nie większy niż $\lceil \frac{n}{r} \rceil$. Stosując najlepszy obecnie znany algorytm dokładny otrzymujemy złożoność $O(2^{n/(3.47r)})$.

Opisaną wyżej konstrukcję możemy nazwać *redukcją*. Znalezienie r-aproksymacyjnego rozwiazania problemu dla rozmiaru n sprowadziliśmy do znalezienia rozwiązania dokładnego w wielomianowej liczbie egzemplarzy rozmiaru $\frac{n}{r}$. Można się zastanawiać, czy nie istnieje lepsza redukcja, na przykład taka, która dla pewnych stałych r_1, r_2 daje algorytm r_1-aproksymacyjny redukując problem do co najwyżej $f(r_2)$ egzemplarzy o rozmiarze nie większym niż $\frac{n}{r_2}$, gdzie $r_1 < r_2$ a f jest funkcją wielomianową. Gdyby taka redukcja istniała, to moglibyśmy ją złożyć ze sobą, czyli zastosować dwukrotnie, otrzymując redukcję dającą r_1^2-aproksymację poprzez znajdowanie rozwiązań dla co najwyżej $f^2(r_2)$ egzemplarzy o rozmiarze $\frac{n}{r_2^2}$. Kontynuując takie składanie otrzymamy w pewnym momencie

$$f^{\log_{r_2} n}(r_2) = r_2^{O(1)\log_{r_2} n} = n^{O(1)},$$

czyli wielomianowo wiele egzemplarzy o rozmiarze $O(1)$. Dla każdego z nich możemy zastosować algorytm wykładniczy, otrzymując wielomianowy algorytm aproksymacyjny o współczynniku $r_1^{\log_{r_2} n} = n^{1/\log_{r_1} r_2}$, co jest niemożliwe przy założeniu, że $NP \neq ZPP$.

4.2. Redukcja w problemie Bandwidth

Przedstawimy teraz redukcję dla problemu Bandwidth, niestety jednak będzie ona dużo bardziej skomplikowana oraz co gorsza mniej wydajna niż redukcja dla zbioru niezależnego, gdyż dla 2-redukcji rozmiaru egzemplarzy otrzymamy 9-aproksymację. Nasza redukcja będzie modyfikowała dany graf tworząc dokładnie jeden nowy egzemplarz – nowy graf, którego liczba wierzchołków będzie co najmniej dwa razy mniejsza niż w oryginalnym grafie.

Niech $G = (V, E)$ będzie spójnym grafem nieskierowanym. Niech M będzie najliczniejszym skojarzeniem w grafie G. Łatwo zauważyć, że każdy sąsiad nieskojarzonego wierzchołka musi być skojarzony, gdyż w przeciwnym wypadku moglibyśmy powiększyć skojarzenie M. Zdefiniujemy teraz funkcję $\rho : V \to V$, która posłuży do scalania grup wierzchołków w jeden wierzchołek. Rozważmy wszystkie krawędzie $uv \in E$, takie że u jest nieskojarzony, natomiast v jest wierzchołkiem skojarzonym. Dla każdej takiej krawędzi definiujemy $\rho(v) = v$. Weźmy teraz pod uwagę wszystkie krawędzie $uv \in M$, możliwe są trzy przypadki:

- Zarówno $\rho(v)$ jak i $\rho(u)$ nie zostały jeszcze zdefiniowane, w takim wypadku przypisujemy $\rho(u) = u$ oraz $\rho(v) = u$ (możemy dowolnie wybrać który koniec owej krawędzi to u a który v).

- Zdefiniowaliśmy wartość funkcji ρ dla dokładnie jednego z końców krawędzi, powiedzmy $\rho(u) = u$, wówczas przypisujemy $\rho(v) = u$.
- Przypisaliśmy $\rho(u) = u$ oraz $\rho(v) = v$, oznacza to, iż wierzchołek u ma nieskojarzonego sąsiada x, ponadto wierzchołek v ma nieskojarzonego sąsiada y. Zauważmy, że musi zachodzić $x = y$, gdyż w przeciwnym wypadku $M \setminus \{uv\} \cup \{ux, vy\}$ byłoby liczniejszym skojarzeniem niż M. W tym przypadku **przedefiniujmy** $\rho(u) = v$ (ponownie wybór końca krawędzi jest dowolny).

Rozważywszy wszystkie krawędzie skojarzenia zdefiniowaliśmy funkcję ρ dla wszystkich wierzchołków skojarzonych $V(M)$. Odnotujmy następujące fakty:

Fakt 4.1. *Dla każdej krawędzi $uv \in M$ zachodzi $\rho(u) = \rho(v) = u$ lub $\rho(u) = \rho(v) = v$.*

Fakt 4.2. *Dla każdego nieskojarzonego wierzchołka $x \notin V(M)$, dla każdego jego sąsiada v mamy $\rho(v) = v$, za wyjątkiem sytuacji w której w grafie G istnieje trójkąt uvx, gdzie $uv \in M$. W tej wyjątkowej sytuacji x jest jedynym nieskojarzonym sąsiadem zarówno u jak i v.*

Musimy jeszcze zdefiniować wartości funkcji ρ dla wierzchołków nieskojarzonych. Rozważmy wszystkie takie wierzchołki w dowolnej kolejności. Niech x będzie nieskojarzonym wierzchołkiem, dla którego wartość $\rho(x)$ nie została jeszcze zdefiniowana. Graf G jest spójny, zatem nie ma w nim wierzchołków izolowanych, co powoduje, że x musi mieć pewnego skojarzonego sąsiada. Jeśli wierzchołek x sąsiaduje z dwoma wierzchołkami u_1,u_2, takimi że $u_1u_2 \in M$ oraz $\rho(u_2) = u_1$, to x jest jedynym nieskojarzonym sąsiadem zarówno wierzchołka u_1 jak i u_2, wtedy przypisujemy $\rho(x) = u_1$. W przeciwnym wypadku, gdy x nie należy do trójkąta zawierającego krawędź skojarzenia, przez u oznaczmy dowolnego sąsiada wierzchołka x. Jeśli u ma jeszcze innego nieskojarzonego sąsiada y dla którego wartość funkcji ρ nie została określona, to przypisujemy $\rho(x) = x$ oraz $\rho(y) = x$, w przeciwnym wypadku definiujemy $\rho(x) = u$. Sposób w jaki zdefiniowaliśmy funkcję ρ pozwala nam stwierdzić, iż:

Fakt 4.3. *Dla co najmniej połowy wierzchołków $v \in V$ zachodzi $\rho(v) \neq v$.*

Fakt 4.4. *Dla każdego wierzchołka $v \in V$, zachodzi $|\rho^{-1}(v)| \leqslant 3$ oraz $\rho(\rho(v)) = \rho(v)$.*

Fakt 4.5. *Dla każdego wierzchołka $v \in V$ wiemy, że wierzchołki v i $\rho(v)$ znajdują się w odległości nie większej niż 2 w grafie G.*

Aby otrzymać nowy, mniejszy egzemplarz naszego problemu dokonamy scalenia wierzchołków, dla których wartości funkcji ρ są równe. Oznaczmy otrzymany graf jako $G' = (V', E')$, gdzie $|V'| = |\rho(V)|$. Zbiór V' możemy interpretować jako podzbiór zbioru V, będący zbiorem punktów stałych funkcji ρ, innymi słowy $V' = \{v \in V : \rho(v) = v\}$. Na podstawie faktu 4.3 wnioskujemy, że liczba wierzchołków grafu G' jest co najmniej dwa razy mniejsza niż liczba wierzchołków grafu G.

Zanim opiszemy w jaki sposób z ustawienia wierzchołków grafu G' otrzymać ustawienie wierzchołków grafu G, oszacujmy wartość $\mathrm{bw}(G')$. W tym celu przez $\pi_1 : V \to \{1, \ldots, |V|\}$ oznaczmy pewne ustawienie wierzchołków V o szerokości równej $\mathrm{bw}(G)$. Z faktu 4.5 natychmiast wynika kolejny fakt:

Fakt 4.6. *Dla każdego wierzchołka $u \in V$ zachodzi $|\pi_1(\rho(u)) - \pi_1(u)| \leqslant 2\mathrm{bw}(G)$.*

Możemy teraz udowodnić następujące twierdzenie:

Twierdzenie 4.7. $\mathrm{bw}(G') \leqslant 3\mathrm{bw}(G) - 1$.

Dowód. Niech $\pi_2 : V' \to \{1, \ldots, |V'|\}$ będzie ustawieniem zbioru V' które zachowuje względny porządek pomiędzy elementami V', taki jak w ustawieniu π_1. Innymi słowy $\mathbf{ciag}(\pi_2)$ jest podciągiem ciągu $\mathbf{ciag}(\pi_1)$ złożonym wyłącznie z wierzchołków V'. Pokażemy, że szerokość ustawienia π_2 wynosi nie więcej niż $3\mathrm{bw}(G) - 1$. Z definicji ustawienia π_2 wynika, że dla dowolnej pary wierzchołków $u, v \in V$ mamy:

$$|\pi_2(\rho(u)) - \pi_2(\rho(v))| \leqslant |\pi_1(\rho(u)) - \pi_1(\rho(v))| \tag{4.1}$$

Weźmy dowolną krawędź $uv \in E$. Chcemy oszacować $|\pi_1(\rho(u)) - \pi_1(\rho(v))|$, w tym celu zauważmy nierówność:

$$|\pi_1(\rho(u)) - \pi_1(\rho(v))| \leqslant |\pi_1(\rho(u)) - \pi_1(u)| + |\pi_1(u) - \pi_1(v)| + |\pi_1(v) - \pi_1(\rho(v))| \tag{4.2}$$

Pokażemy, że prawa strona nierówności 4.2 jest nie większa niż $3\mathrm{bw}(G)$. Wiemy, że $|\pi_1(u) - \pi_1(v)| \leqslant \mathrm{bw}(G)$, gdyż wierzchołki u i v są sąsiednie w grafie G. Z faktu iż M jest najliczniejszym skojarzeniem wynika, że co najmniej jeden z wierzchołków u,v jest skojarzony. Rozważmy przypadki:

- Jeśli oba wierzchołki u,v są skojarzone w M, to korzystając z faktu 4.1 wiemy że u oraz $\rho(u)$ są sąsiednie w G, zatem $|\pi_1(\rho(u)) - \pi_1(u)| \leqslant \mathrm{bw}(G)$, analogicznie $|\pi_1(\rho(v)) - \pi_1(v)| \leqslant \mathrm{bw}(G)$.
- Dokładnie jeden z wierzchołków u,v jest skojarzony. Bez straty ogólności załóżmy, że jest to wierzchołek u, przez u' oznaczmy wierzchołek z nim skojarzony. Jeśli zachodzi $\rho(u) = u$, to na podstawie faktu 4.6 wiemy, że $|\pi_1(\rho(u)) - \pi_1(u)| + |\pi_1(u) - \pi_1(v)| + |\pi_1(v) - \pi_1(\rho(v))| \leqslant 3\mathrm{bw}(G)$. Jeśli natomiast $\rho(u) \neq u$, to z faktów 4.1 i 4.2 wynika, że v sąsiaduje z u', gdzie $u' = \rho(u)$, zatem w grafie G istnieje trójkąt zawierający krawędź skojarzenia oraz wierzchołek v, zatem $\rho(v)$ oraz v są sąsiednie, podobnie jak u oraz $\rho(u)$.

W obu przypadkach prawa strona nierówności 4.2 jest nie większa niż $3\mathrm{bw}(G)$. Łącząc ten fakt z nierównością 4.1 otrzymujemy $\mathrm{bw}(G') \leqslant 3\mathrm{bw}(G)$. Aby zmniejszyć prawą stronę ostatniej nierówności o jeden, zauważmy że jeśli nierówność 4.2 jest ostra, to $|\pi_2(\rho(u)) - \pi_2(\rho(v))| \leqslant 3\mathrm{bw}(G) - 1$. Możemy zatem założyć, że nierówność 4.2 ma postać równości. Bez straty ogólności załóżmy, że $\pi_1(\rho(u)) \leqslant \pi_1(\rho(v))$, zatem z postaci nierówności 4.2 wynika, że $\pi_1(\rho(u)) \leqslant \pi_1(u) < \pi_1(v) \leqslant \pi_1(\rho(v))$.

Mamy teraz dwa przypadki:

- jeśli $u = \rho(u)$ oraz $v = \rho(v)$, to

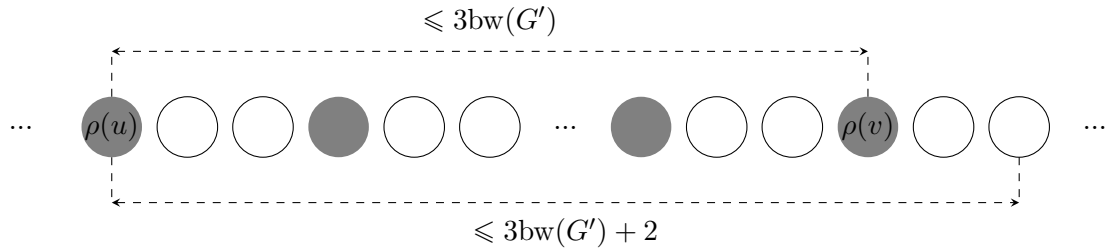
  $$|\pi_2(\rho(u)) - \pi_2(\rho(v))| \leqslant |\pi_1(\rho(u)) - \pi_1(\rho(v))| = |\pi_1(u) - \pi_1(v)| \leqslant \mathrm{bw}(G) \leqslant 3\mathrm{bw}(G) - 1.$$

- w przeciwnym wypadku, mamy $u \neq \rho(u)$ lub $v \neq \rho(v)$, czyli co najmniej jeden z wierzchołków u,v nie należy do zbioru V', zatem

  $$|\pi_2(\rho(u)) - \pi_2(\rho(v))| < |\pi_1(\rho(u)) - \pi_1(\rho(v))| \leqslant 3\mathrm{bw}(G),$$

  gdyż $\pi_1(\rho(u) \leqslant \pi_1(u) < \pi_1(v) \leqslant \pi_1(\rho(v))$. Wówczas $|\pi_2(\rho(u)) - \pi_2(\rho(v))| \leqslant 3\mathrm{bw}(G) - 1$, co kończy dowód twierdzenia 4.7.

□



Rysunek 4.1: Ciąg **ciag**(π_1), zaczernione pola oznaczają wierzchołki będące punktami stałymi funkcji ρ, czyli elementy ciągu **ciag**(π_2).

Teraz pokażemy w jaki sposób na podstawie ustawienia wierzchołków grafu G' możemy skonstruować ustawienie wierzchołków grafu G. Niech $\pi_2 : V' \to \{1, \ldots, |V'|\}$ będzie ustawieniem o szerokości $\mathrm{bw}(G') \leqslant 3\mathrm{bw}(G) - 1$. Zdefiniujmy funkcję $\pi_1 : V \to \{1, \ldots, |V|\}$ poprzez konstrukcję ciągu **ciag**(π_1). Początkowo za nasz ciąg przyjmijmy **ciag**(π_2), następnie do tego ciągu wstawiamy kolejne wierzchołki zbioru $V \setminus V'$ (w dowolnej kolejności), wierzchołek $x \in V \setminus V'$ wstawiamy tuż za wierzchołkiem $\rho(x)$. Ciąg **ciag**(π_1) jednoznacznie wyznacza nam ustawienie π_1. Na podstawie faktu 4.4 wiemy, że każdy wierzchołek ciągu **ciag**(π_2) został zastąpiony przez co najwyżej trzy kolejne wierzchołki. Weźmy teraz dowolną krawędź $uv \in E$, łatwo zauważyć (zob. rysunek 4.1), że

$$|\pi_1(u) - \pi_1(v)| \leqslant 3|\pi_2(\rho(u)) - \pi_2(\rho(v))| + 2 \leqslant 3(3\mathrm{bw}(G) - 1) + 2 = 9\mathrm{bw}(G) - 1$$

Powyższa nierówność prowadzi nas do twierdzenia:

Twierdzenie 4.8. *Przedstawiony schemat postępowania dla problemu* Bandwidth *jest 2-redukcją o współczynniku aproksymacji równym* 9*.*

Jeśli złożymy powyższą redukcję k razy, a następnie użyjemy algorytmu przedstawionego w rozdziale 5, to otrzymamy algorytm 9^k-aproksymacyjny o złożoności $O^*(5^{n/2^k})$. W następnym rozdziale przedstawimy szybszy algorytm aproksymacyjny dla problemu Bandwidth.

4.3. Wariacje na temat podziału na kubełki

W tym rozdziale spróbujemy trochę poeksperymentować z podziałem na kubełki, odejść od podziału klasycznego i połówkowego. Kosztem powiększenia współczynnika aproksymacji (który jednak wciąż będzie stały) otrzymamy szybsze algorytmy. Będzie się to odbywać przy pomocy powiększenia kubełków w procesie podziału na kubełki. Będziemy mieli mniejszy wybór (a więc i mniejszą złożoność, mniej podziałów do przeszukania), lecz wielkość kubełków, a więc i podziału, będzie większa, co powoduje, że aproksymacja będzie gorsza.

W tym rozdziale zakładamy, że kubełki klasyczne mają po b elementów, czyli $\bar{b} = b$ w definicji podziału klasycznego.

4.3.1. Ogólny schemat algorytmu podziału na kubełki

Przyjrzyjmy się algorytmowi generowania wszystkich klasycznych podziałów na kubełki. W jednej fazie, brał on krawędź uv, gdzie u było już przydzielone do kubełka, zaś v jeszcze nie

i przydzielał v do jednego z tych kubełków, które miały sens, biorąc pod uwagę do którego kubełka przydzieliliśmy wcześniej u. Zauważmy, że wybierane krawędzie uv tworzą drzewo rozpinające grafu G i w tym algorytmie jest to dowolne drzewo. Co więcej, możemy najpierw wybrać drzewo rozpinające D, według którego algorytm generowania podziałów na kubełki będzie działał.

Ogólnie, schemat algorytmu podziału wierzchołków na kubełki możemy zarysować następująco:

- algorytm wybiera drzewo rozpinające D, według którego będzie rozdzielał do kubełków; zazwyczaj będzie to dowolne drzewo rozpinające, fakt jego istnienia raczej będzie nam służył do analizy algorytmu niż potrzebny jawnie w samym algorytmie,

- mamy zdefiniowany pewien zbiór dopuszczalnych kubełków $\mathcal{K}$, czyli możliwych wartości konstruowanego podziału p,

- próbujemy umieścić korzeń drzewa D w jednym z kubełków: wybieramy podzbiór $\mathcal{U} \subset \mathcal{K}$ spełniający $N = \bigcup \mathcal{U}$ i wybieramy dla korzenia drzewa D jeden z kubełków — elementów $\mathcal{U}$; innymi słowy, wybieramy dla korzenia drzewa D jeden z kubełków tak, by wszystkie możliwe wybory kubełków pokrywały cały zbiór pozycji N; dla każdego wyboru będziemy rekurencyjnie określać wartości $p(v)$ dla pozostałych wierzchołków,

- przechodzimy przeszukiwaniem w głąb drzewo D: jeśli $p(v)$ jeszcze nie jest określone, zaś u jest ojcem v w D i $p(u)$ jest określone, to wybieramy podzbiór $\mathcal{U}(v) \subset \mathcal{K}$ taki, by

  $$\{i \in \{1, 2 \ldots, n\} : \exists_{j \in p(u)} |i - j| \leqslant b\} \subset \bigcup \mathcal{U}(v),$$

  (czyli by zbiór wszystkich pozycji, gdzie potencjalnie może się znaleźć wierzchołek v, jeśli u znalazło się w $p(u)$, był pokryty przez elementy $\mathcal{U}(v)$) i określamy $p(v) = U(v)$ na $|\mathcal{U}(v)|$ sposobów — dla każdego $U(v) \in \mathcal{U}(v)$; dla każdej wartości $p(v)$ rekurencyjnie określamy wartości p dla dalszych wierzchołków,

- na koniec, przeprowadzamy proces *przycinania* podziału; jeśli któreś $p(v)$ okazało się puste, odrzucamy ten podział na kubełki.

Fakt 4.9. *Jeśli π jest ustawieniem spełniającym* $\mathrm{bw}(\pi) \leqslant b$*, to co najmniej jeden z podziałół wygenerowanych przez powyższy algorytm będzie zgodny z π.*

Dowód. W algorytmie, w momencie wyboru wartości $p(v)$, wybieramy taki $U(v)$, że $\pi(v) \in U(v)$. Zbiory $U(v) \in \mathcal{U}(v)$ pokrywają wszystkie potencjalne pozycje v w ustawieniu π, więc co najmniej jeden wybór $U(v)$ podczas rekurencyjnego definiowania funkcji p będzie właściwy. □

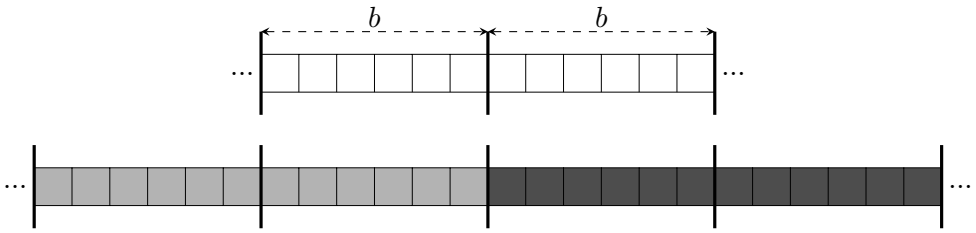
W kolejnych algorytmach będziemy eksperymentować z różnymi wielkościami i układem kubełków, do których przydzielamy wierzchołki.

4.3.2. Podział 2^n

Rozważmy następujący algorytm podziału na kubełki, działający według opisanego schematu, gdzie:

- Liczba $\bar{b} = b$ lub $\bar{b} = b + 1$.

- Dopuszczalnymi kubełkami są wszystkie przedziały zaczynające się od pozycji o numerach $i\bar{b}+1$ dla $i \in \mathbb{Z}$, o długości $2\bar{b}$ (lub krótsze, jeśli taki przedział wystaje poza zbiór $\{1, 2, \ldots, n\}$, wówczas przecinamy go z tym zbiorem).
- Korzeń drzewa rozpinającego przydzielamy do każdego możliwego kubełka, których jest nie więcej niż $\lceil \frac{n}{b} \rceil + 1$.
- W kroku algorytmu, jeśli wierzchołek u jest przydzielony do kubełka zaczynającego się od pozycji $i\bar{b}+1$, to jego syna v przydzielamy do jednego z **dwóch** kubełków zaczynających się na pozycjach $((i-1)\bar{b}+1)$ i $((i+1)\bar{b}+1)$; zauważmy, że te przedziały pokrywają potencjalne pozycje dla v i nie potrzebujemy brać pod uwagę przedziału zaczynającego się od pozycji $(i\bar{b}+1)$ (zob. rys. 4.2).



Rysunek 4.2: Górny fragment rysunku zawiera kubełek wierzchołka u wielkości $2\bar{b}$ dla $\bar{b} = b$. Dolny fragment rysunku prezentuje przedział dozwolonych pozycji dla wierzchołka v, który został pokryty przez dwa kubełki rozmiaru $2\bar{b}$ (jeden jasnoszary, drugi ciemnoszary).

Definicja 4.10. Podziały wygenerowane przez powyższy algorytm będziemy nazywali *podziałami podwójnymi*.

Uwaga 4.11. Korzeń umieszczamy na mniej niż n sposobów w kubełkach, zaś na każdym kroku mamy do wyboru dwa kubełki. Wobec tego istnieje co najwyżej $n2^n$ podziałów wygenerowanych przez ten algorytm. Algorytm działa w czasie $O^*(2^n)$ i potrzebuje wielomianowej pamięci.

Następujące dwa fakty wynikają bezpośrednio z definicji podziału:

Fakt 4.12. *W podziale podwójnym wielkość każdego kubełka jest nie większa niż $2\bar{b}$, czyli wielkość podziału jest nie większa niż $2\bar{b}$.*

Fakt 4.13. *W podziale podwójnym, jeśli uv jest krawędzią drzewa rozpinającego D, po którym odbywa się rozrzucanie do kubełków, to $p(u)$ i $p(v)$ są przedziałami przesuniętymi względem siebie o $\bar{b}$ (ewentualnie obciętymi do przedziału $[1, n]$).*

Zauważmy, iż podobnie jak w przypadku podziałów klasycznych i połówkowych, jesteśmy tu w stanie otrzymać pewien algorytm aproksymacyjny:

Wniosek 4.14. Istnieje algorytm 3-aproksymacyjny dla problemu minimalizacyjnego Bandwidth, działający w czasie $O^*(2^n)$.

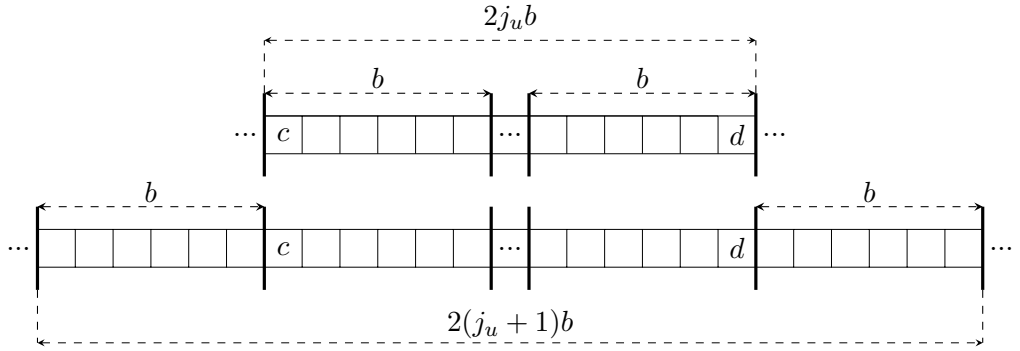
Dowód. Wyszukujemy binarnie minimalne b', dla którego istnieje jakikolwiek podwójny podział na kubełki w przypadku $\bar{b} = b$ i istnieje ustawienie zgodne z tym podziałem (zgodnie z faktem 3.4 sprawdzenie jest wielomianowe). Dowolne takie ustawienie π spełnia $\mathrm{bw}(\pi) \leqslant K(p) + b' \leqslant 2b' + b' \leqslant 3\mathrm{bw}(G)$, czyli jest to 3-aproksymacja. □

4.3.3. Inne podziały, $(4k-1)$-aproksymacja w czasie $O^*(2^{\frac{n}{k}})$

Spróbujmy zastanowić się nad innym sposobem podziału na kubełki, wzorowanym na podwójnym podziale, lecz o większej szerokości, a mniejszej liczbie podziałów. Dzięki temu złożoność ewentualnego algorytmu aproksymacyjnego będzie mniejsza.

Zaproponujemy następujący schemat: przyjmijmy liczbę całkowitą dodatnią k. Niech $n = |V|$ i $N = \{1, 2, 3, \ldots, n\}$.

- W czasie algorytmu definiowane kubełki będą przedziałami liczb całkowitych; dopiero na końcu, dla każdego powstałego podziału, każdy kubełek zostaje przecięty ze zbiorem N oraz przycięty i podział zostaje zaakceptowany, jeśli każdy kubełek pozostał niepusty.
- Dopuszczalnymi kubełkami są przedziały, zaczynające się na pozycjach $ib+1$ dla $i \in \mathbb{Z}$, o długości $2jb$ dla $k \leqslant j \leqslant 2k-1$.
- Dla korzenia wybieramy jedną z liczb $k \leqslant j_0 \leqslant 2k-1$ i przydzielamy korzeń drzewa do jednego z kubełków wielkości $2j_0b$ o niepustym przecięciu z N; tych kubełków jest nie więcej niż n.
- Jeśli v jest wierzchołkiem w drzewie D rozrzucania na kubełki i u jest jego już przydzielonym ojcem i u zostało przydzielone do kubełka $[c, d]$ szerokości $2j_ub$ dla $j_u < 2k-1$, to przydzielamy v do **jednego** kubełka $[c-b, d+b]$, szerokości $2(j_u+1)b$ (zob. rysunek 4.3).

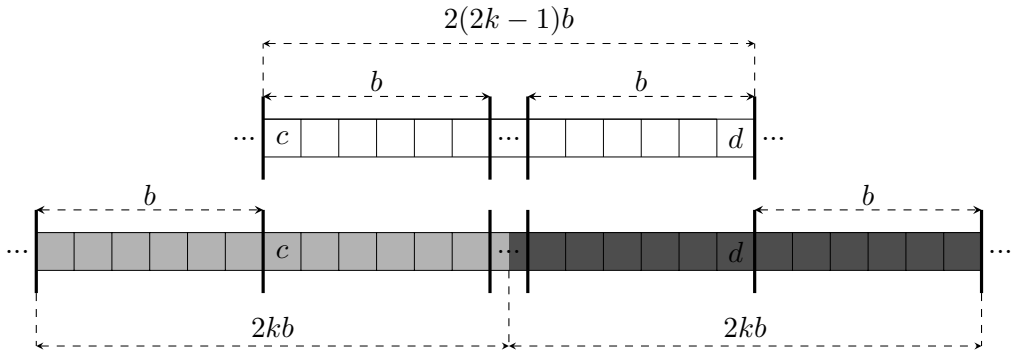


Rysunek 4.3: Górny fragment rysunku zawiera kubełek wierzchołka u wielkości $2j_ub$. Dolny fragment rysunku prezentuje przedział dozwolonych pozycji dla wierzchołka v, który został pokryty przez **jeden** kubełek rozmiaru $2(j_u+1)b$.

- Jeśli u jest ojcem v i u zostało przydzielone do kubełka $[c, d]$ szerokości $2(2k-1)b$, to v przydzielamy do jednego z **dwóch** przedziałów-kubełków $[c-b, \frac{c+d-1}{2}]$ i $[\frac{c+d+1}{2}, d+b]$, oba szerokości $2kb$. Dla każdego z dwóch wyborów rekurencyjnie przydzielany dotychczas nieprzydzielone wierzchołki (zob. rysunek 4.4).
- Na koniec, każdy kubełek przecinamy z N i podział odrzucamy, jeśli któryś kubełek okazał się pusty (rozłączny z N).

Zauważmy, że jeśli oznaczymy, że wierzchołkowi v przyporządkowaliśmy przedział szerokości $2j_vb$, to zachodzi następująca prawidłowość:

- Korzeniowi przyporządkowaliśmy przedział szerokości $2j_0b$.
- Synom korzenia, czyli wierzchołkom w odległości 1 od korzenia — $2(j_0+1)b$.



Rysunek 4.4: Górny fragment rysunku zawiera kubełek wierzchołka u wielkości $2(2k-1)b$. Dolny fragment rysunku prezentuje przedział dozwolonych pozycji dla wierzchołka v, który został pokryty przez **dwa** kubełki rozmiaru $2kb$.

- Wierzchołkom w odległości i od korzenia — przedziały szerokości $2(j_0+i)b$.
- Wierzchołkom w odległości $(2k-1-j_0)$ od korzenia — przedziały szerokości $2(2k-1)b$.
- Wierzchołkom w odległości $(2k-j_0)$ od korzenia — przedziały szerokości $2kb$, przy czym tutaj mamy wybór pomiędzy dwoma przedziałami.
- I tak dalej.

Odległość rozumiemy jako odległość w drzewie rozpinającym D.

Zauważmy, iż takich podziałów jest $n2^S$, gdzie S to liczba wierzchołków, których odległość od korzenia przystaje do $2k-j_0$ modulo k. Podzielmy wierzchołki na k klas V_i: $v \in V_i$ jeśli odległość w D wierzchołka v od korzenia przystaje do i modulo k. Jest to podział rozłączny wierzchołków grafu na k zbiorów, więc istnieje i_0 dla którego $|V_{i_0}| \leqslant \frac{n}{k}$. Możemy dobrać tak j_0, by to właśnie na wierzchołkach z V_{i_0} w algorytmie podziału był wybór pomiędzy kubełkami wielkości $2kb$. Wówczas $S \leqslant \frac{n}{k}$. Zauważmy, iż w takim podziale największy kubełek ma wielkość $2(2k-1)b$, czyli, jeśli po stworzeniu podziału przeprowadzimy proces przycinania podziału, i, podobnie jak w poprzednich algorytmach, będziemy szukać najmniejszego b' dla którego istnieje wyżej opisany podział na kubełki, otrzymamy następującą klasę algorytmów aproksymacyjnych.

Wniosek 4.15. Niech k będzie liczbą całkowitą dodatnią. Istnieje algorytm $(4k-1)$-aproksymacyjny problemu minimalizacyjnego Bandwidth działający w czasie $O^*(2^{\frac{n}{k}})$.

Uwaga 4.16. Dla $k=1$ powyższy podział jest podziałem podwójnym. Innymi słowy, ten podział jest rozszerzeniem schematu podziału podwójnego.

Wniosek 4.17. W szczególności, dla $k=2$, istnieje algorytm 7-aproksymacyjny działający w czasie $O^*(2^{n/2}) = O(1.42^n)$.

Wniosek 4.18. Ten algorytm ma swoją „niszę rynkową” dla pewnej klasy wielkości problemów. Zauważmy, że dla na przykład $n = 1000$ algorytm Feige'a, $O(\log^3 n\sqrt{\log n \log\log n})$-aproksymacyjny jest nieciekawy — jest to więcej niż 1000-aproksymacja. Zaś, przyjmując w tym algorytmie $k=50$ otrzymujemy 201-aproksymację, przy czym $2^{\frac{n}{k}} = 2^{20} \sim 1\,000\,000$, czyli całkiem znośnie.

Rozdział 5

Algorytm dokładny o złożoności $O(4.829^n)$

W tym rozdziale spróbujemy, na bazie klasycznego i podwójnego podziału do kubełków, skonstruować algorytm dokładny, rozwiązujący decyzyjny problem Bandwidth. Zaproponujemy dwa algorytmy, oba będą działały w czasie $O^*(5^n)$, jednakże precyzyjna analiza drugiego algorytmu wykaże, że jego złożoność obliczeniowa wynosi $O(4.829^n)$. Oba algorytmy mają wykładniczą złożoność pamięciową, niemniej jednak w przypadku pierwszego algorytmu wynosi ona $O^*(2^n)$, co sprawia że w praktycznej implementacji zapotrzebowanie na pamięć nie będzie wąskim gardłem. Niestety złożoność pamięciowa drugiego algorytmu wynosi $O^*(4^n)$, co powoduje, że jest to wynik czysto teoretyczny.

W tym rozdziale załóżmy, że mamy dany egzemplarz problemu decyzyjnego Bandwidth $(G = (V, E), b)$ i oznaczmy $n = |V|$. Ponadto, ponumerujmy pozycje na wierzchołki liczbami od 0 do $n-1$, zamiast od 1 do n. Innymi słowy, niech ustawienie π będzie bijekcją prowadzącą z V w zbiór pozycji $N = \{0, 1, \ldots, n-1\}$. W tym rozdziale podział klasyczny na kubełki rozumiemy jako podział na kubełki wielkości $b+1$, czyli przyjmujemy $\bar{b} = b+1$ w definicji podziału klasycznego i podwójnego.

5.1. Algorytm warstwowy

Oznaczmy $k = \lceil \frac{n}{b+1} \rceil$. Do końca rozdzialu 5 przez *kubełek* będziemy rozumieli kubełek klasyczny $\bar{b}$-podziału rozmiaru $\bar{b} = b+1$. Dla porządku przytoczmy tu definicję kubełka:

Definicja 5.1. Kubełkiem o numerze i, dla $0 \leqslant i < k$, będziemy nazywać przedział pozycji $K_i = \{j \in N : i(b+1) \leqslant j < (i+1)(b+1)\}$, inaczej mówiąc $K_i = \{i(b+1), i(b+1)+1, \ldots, (i+1)(b+1)-1\} \cap N$. Ostatni kubełek może być krótszy jeśli $(b+1) \nmid n$. Przez $\mathbf{kub}_\pi(v)$ będziemy oznaczać numer kubełka do którego trafia wierzchołek v w ustawieniu π, czyli $\mathbf{kub}_\pi(v) = \lfloor \frac{\pi(v)}{b+1} \rfloor$.

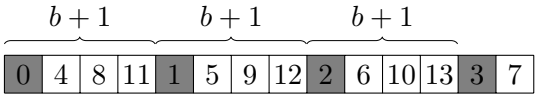
W naszym nowym algorytmie będziemy przydzielać wierzchołki do kolejnych pozycji, jednakże kluczowy okaże się porządek pozycji, w jakim będziemy je wypełniać wierzchołkami:

Definicja 5.2. *Porządkiem warstwowym* zbioru pozycji $N = \{0, 1, \ldots, N-1\}$ nazwiemy permutację $\sigma : N \to N$, taką, że dla $i < j$ zachodzi

$$\Big(\sigma(i) \bmod (b+1), \Big\lfloor \frac{\sigma(i)}{b+1} \Big\rfloor\Big) <_{lex} \Big(\sigma(j) \bmod (b+1), \Big\lfloor \frac{\sigma(j)}{b+1} \Big\rfloor\Big).$$

Innymi słowy (zob. rys 5.1),

$$(\sigma(0), \sigma(1), \ldots) = (0, b+1, 2(b+1), \ldots, 1, 1+(b+1), 1+2(b+1), \ldots).$$



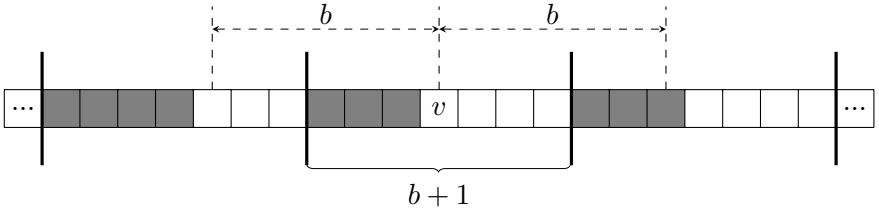
Rysunek 5.1: Przykład porządku warstwowego dla $n = 14$ i $b = 3$, w kolejne pola wpisano liczby w porządku warstwowym dla danych pozycji. Pierwsza pozycja w każdym kubełku jest wypełniona kolorem szarym.

Zauważmy, że przy tak zdefiniowanym porządku pozycji zachodzi następujący fakt:

Fakt 5.3. *Ustawienie π spełnia warunek* $\mathrm{bw}(\pi) \leqslant b$ *wtedy i tylko wtedy gdy dla każdej krawędzi uv jeśli* $\mathbf{kub}_\pi(u) < \mathbf{kub}_\pi(v)$*, to* $\mathbf{kub}_\pi(u) + 1 = \mathbf{kub}_\pi(v)$ *oraz* $\sigma(\pi(u)) > \sigma(\pi(v))$.

Dowód. Formalny dowód można zastąpić rysunkiem 5.2, jednakże dla porządku przedstawimy również dowód rachunkowy. Załóżmy, że ustawienie π spełnia warunek $\mathrm{bw}(\pi) \leqslant b$ i niech krawędź uv spełnia $\mathbf{kub}_\pi(u) < \mathbf{kub}_\pi(v)$, czyli $\lfloor \frac{\pi(u)}{b+1} \rfloor < \lfloor \frac{\pi(v)}{b+1} \rfloor$, czyli $\pi(u) < \pi(v)$. Ale skoro $\mathrm{bw}(\pi) \leqslant b$, to $\pi(u) + b \geqslant \pi(v)$, czyli $\lfloor \frac{\pi(u)}{b+1} \rfloor + 1 = \lfloor \frac{\pi(v)}{b+1} \rfloor$. Wówczas $b \geqslant \pi(v) - \pi(u) = b + 1 + (\pi(v) \bmod (b+1)) - (\pi(u) \bmod (b+1))$, zatem $\pi(u) \bmod (b+1) > \pi(v) \bmod (b+1)$, co daje $\sigma(\pi(u)) > \sigma(\pi(v))$.

W drugą stronę, niech uv będzie krawędzią grafu. Załóżmy bez straty ogólności, że $\pi(v) > \pi(u)$. Chcemy pokazać, że $\pi(v) - \pi(u) \leqslant b$. Jeśli $\mathbf{kub}_\pi(u) = \mathbf{kub}_\pi(v)$, to teza jest oczywista. Wobec tego niech $\mathbf{kub}_\pi(u) < \mathbf{kub}_\pi(v)$, czyli z założenia, $\mathbf{kub}_\pi(u) + 1 = \mathbf{kub}_\pi(v)$ oraz $(\pi(u) \bmod (b+1)) > (\pi(v) \bmod (b+1))$. Wówczas $\pi(v) - \pi(u) = b + 1 + (\pi(v) \bmod (b+1)) - (\pi(u) \bmod (b+1)) \leqslant b$. □



Rysunek 5.2: Szare pola oznaczają pozycje o numerach mniejszych w porządku warstwowym od numeru pozycji na której znajduje się wierzchołek v.

Załóżmy, że mamy dany podział podobny do klasycznego p, tj podział, w którym wartości $p(v)$ są przedziałami postaci $[i(b+1), j(b+1))$. Naszym celem będzie opisanie algorytmu, który sprawdzi czy istnieje ustawienie π zgodne z p i spełnające $\mathrm{bw}(\pi) \leqslant b$. Załóżmy, że wielkość podziału p wynosi $s(b+1)$. Algorytm do którego zmierzamy będzie miał złożoność $O^*((s+1)^n)$.

Od faktu 5.3 niedaleko już do naszego algorytmu. Zdefiniujmy *stany* oraz *przejścia*:

Definicja 5.4. *Stanem* w naszym algorytmie będzie para (A, ϕ), gdzie:

1. $A \subseteq V$ reprezentuje zbiór wierzchołków przydzielonych do pierwszych $|A|$ pozycji w porządku warstwowym,
2. $\phi : A \to \{i : 0 \leqslant i < k\}$ spełnia $K_{\phi(v)} \subset p(v)$ (czyli jest przyporządkowaniem wierzchołków z A do kubełków rozmiaru $b+1$ w sposób zgodny z podziałem p),
3. $|\phi^{-1}(i)| = |\sigma(\{0, 1, \ldots, |A|-1\}) \cap K_i|$, czyli w zbiorze $\phi^{-1}(i)$ jest tyle wierzchołków, ile jest pozycji w kubełku K_i wśród pierwszych $|A|$ pozycji w porządku σ.

Definicja 5.5. *Przejściem* w naszym algorytmie nazywamy parę stanów (A, ϕ) oraz (A', ϕ'), gdzie:

1. $A' = A \cup \{v\}$, $v \notin A$, $\phi'|_A = \phi$;
2. dla każdego sąsiada u wierzchołka v zachodzi dokładnie jeden z warunków:
   - $u \in A$ oraz $\phi(v) \leqslant \phi(u) \leqslant \phi(v) + 1$, czyli u leży w tym samym kubełku co v lub w sąsiednim kubełku na prawo od kubełka v,
   - $u \notin A'$ oraz $(K_{\phi(v)} \cup K_{\phi(v)-1}) \cap p(u) \neq \emptyset$, innymi słowy istnieje pozycja w kubełku do którego trafił wierzchołek v lub w kubełku sąsiednim z lewej strony, która to pozycja należy do zbioru $p(u)$ (zauważmy, że są to jedyne dwa kubełki do których może trafić u aby nie przekroczyć odległości b od wierzchołka v, zob. rys. 5.2).

Oszacujmy liczbę stanów i przejść:

Fakt 5.6. *Stanów algorytmu jest nie więcej niż* $(s+1)^n$.

Dowód. Dla stanu (A, ϕ) każdy wierzchołek może być albo w $V \setminus A$, albo w A i wówczas ϕ przyjmuje co najwyżej s różnych wartości, jako że $K_{\phi(v)} \subset p(v)$, czyli dla każdego wierzchołka mamy co najwyżej $s+1$ możliwości. □

Fakt 5.7. *Z każdego stanu prowadzi co najwyżej n przejść.*

Dowód. Mając dany stan (A, ϕ), by otrzymać przejście do większego stanu, musimy określić ϕ na jednym nowym wierzchołku. W tym celu wybieramy z $V \setminus A$ wierzchołek który przydzielany na pozycję o numerze $|A|$ w porządku warstwowym (nie możemy wybrać innej pozycji ze względu na warunek 3 w definicji 5.4), sprawdzając czy takie przyporządkowanie spełnia warunki definicji 5.4. □

Do udowodnienia poprawności algorytmu, który wkrótce przedstawimy, potrzebne nam będą następujące fakty:

Fakt 5.8. *Jeśli* π *jest ustawieniem spełniającym* $\mathrm{bw}(\pi) \leqslant b$, *to definiując* $\Theta_r = (A_r, \phi_r)$, *gdzie*

$$A_r = \pi^{-1}(\{\sigma(0), \sigma(1), \ldots, \sigma(r-1)\})$$
$$\phi_r(v) = \mathbf{kub}_\pi(v), \text{ dla } v \in A_r$$

dla $0 \leqslant r \leqslant n$ *otrzymujemy ciąg stanów, w którym pomiędzy kolejnymi stanami są przejścia.*

Dowód. Zauważmy, że warunki narzucone na stan i przejście w definicjach 5.4 oraz 5.5 to przeformułowanie warunku równoważnego określonego w fakcie 5.3 □

Fakt 5.9. *Jeśli* $\Theta_r = (A_r, \phi_r)$ *dla* $0 \leqslant r \leqslant n$ *są stanami i istnieją przejścia między* Θ_r *a* Θ_{r+1} *dla każdego* $0 \leqslant r < n$, *to ustawienie* $\pi(A_r \setminus A_{r-1}) = \sigma(r-1)$ *jest zgodne z podziałem* p *oraz spełnia warunek* $\mathrm{bw}(\pi) \leqslant b$.

Dowód. Definicje stanu i przejścia w połączeniu z faktem 5.3 gwarantują, że dla każdego stanu (A_r, ϕ_r) istnieje przyporządkowanie wierzchołków A_r, zgodne z ϕ_r, do pierwszych $|A_r|$ pozycji w porządku warstwowym, w taki sposób, aby odległość pomiędzy sąsiednimi wierzchołkami była nie większa niż b. □

Wniosek 5.10. Algorytm, który zaczyna ze stanu $\Theta_0 = (\emptyset, \emptyset)$ i szuka przeszukiwaniem w głąb ścieżki do stanu $(V, \cdot)$, nie wchodząc wielokrotnie do tego samego stanu, poprawnie sprawdza czy istnieje ustawienie π zgodne z p, spełniające $\mathrm{bw}(\pi) \leqslant b$.

Algorytm ten działa w pamięci $O^*((s+1)^n)$ — dla każdego stanu pamiętamy, czy go już odwiedziliśmy — oraz w czasie $O^*((s+1)^n)$ — z każdego stanu usiłujemy stworzyć przejścia na każdy możliwy sposób; sprawdzenie, czy coś jest przejściem lub stanem jest operacją którą można wykonać w czasie wielomianowym.

5.2. Algorytm dokładny $O^*(5^n)$

W tym rozdziale spróbujemy zastosować do podziału klasycznego wniosek 5.10, czyli algorytm sprawdzający w czasie i pamięci $O^*((s+1)^n)$, czy istnieje ustawienie π spełniające $\mathrm{bw}(\pi) \leqslant b$ i zgodne z podobnym do klasycznego podziałem na kubełki p o wielkości nie większej niż $s(b+1)$.

W tym rozdziale zakładamy, że p jest podziałem klasycznym, czyli $s = 1$. Przypomnijmy, że oznacza to, że jeśli $uv \in E$ to $p(u)$ i $p(v)$ to ten sam lub sąsiednie kubełki.

Twierdzenie 5.11. *Istnieje algorytm działający w czasie $O^*(6^n)$ i pamięci $O^*(2^n)$ rozwiązujący problem decyzyjny* Bandwidth.

Dowód. Wygenerujmy wszystkie podziały klasyczne. Jak już pokazywaliśmy, jest ich $O^*(3^n)$. Dla każdego podziału klasycznego p, algorytmem z wniosku 5.10 spawdźmy, czy istnieje ustawienie π zgodne z podziałem p i spełniające $\mathrm{bw}(\pi) \leqslant b$.

Podziały klasyczne generujemy na bieżąco, co zajmuje pamięć wielomianową. W algorytmie z wniosku 5.10 potrzebujemy $O^*((s+1)^n) = O^*(2^n)$ pamięci. □

Teraz oszacujemy, że algorytm z twierdzenia 5.11 tak naprawdę działa w czasie $O^*(5^n)$.

Fakt 5.12. *Niech (A, ϕ) będzie stanem odwiedzonym przez algorytm z twierdzenia 5.11. Niech uv będzie krawędzią, dla której $u \notin A$ i $v \in A$. Wówczas $p(v) = p(u)$ lub $p(u)$ jest kubełkiem na lewo od $p(v)$.*

Dowód. Oznaczmy $m = |A|$. Skoro stan (A, ϕ) jest odwiedzony w algorytmie z twierdzenia 5.11, to istnieje ciąg stanów i przejść $(\emptyset, \emptyset) = \Theta_0 = (A_0, \phi_0)$, $\Theta_1 = (A_1, \phi_1)$, $\ldots$, $\Theta_m = (A_m, \phi_m) = (A, \phi)$. Niech $v \in A_i \setminus A_{i-1}$. Wówczas, zgodnie z definicją 5.5 dla przejścia od Θ_{i-1} do Θ_i, skoro $uv \in E$ oraz $u \notin A_i$, to $p(u) = K_{\phi_i(v)} = p(v)$ lub $p(u) = K_{\phi_i(v)-1}$, czyli $p(u) = p(v)$ lub $p(u)$ jest kubełkiem na lewo od $p(v)$. □

Fakt 5.13. *W trakcie działania algorytmu z twierdzenia 5.11, w czasie przeglądania wszystkich klasycznych podziałów na kubełki, algorytm ten odwiedza sumarycznie co najwyżej $2n \cdot 5^n$ stanów.*

Dowód. Niech D będzie drzewem rozpinającym G, według którego algorytm rozrzucał wierzchołki na kubełki klasyczne. Spójrzmy na stan (A, ϕ) osiągnięty przy podziale na kubełki p. Korzeń drzewa D przydzielamy na n sposobów do kubełków i korzeń należy do A lub nie należy do A — dwie możliwości.

Niech u będzie wierzchołkiem drzewa D niebędącym korzeniem. Niech u' będzie ojcem u w drzewie D. Wierzchołek u jest przydzielony do jednego z trzech kubełków: do $p(u')$ lub do sąsiada $p(u')$, dodatkowo u należy do A lub nie należy do A. Skorzystajmy teraz z faktu 5.12. Jeśli $u' \in A$, to nie może zajść przypadek: $u \notin A$ i $p(u)$ jest kubełkiem na prawo od $p(u')$. Jeśli zaś $u' \notin A$, to nie może zajść przypadek: $u \in A$ i $p(u)$ jest kubełkiem na lewo od $p(u')$. Wobec tego zawsze dla u jedna z 6 możliwości odpada.

W sumie, wszystkich stanów dla wszystkich podziałów na kubełki klasyczne jest co najwyżej $2n5^n$. □

Wniosek 5.14. Algorytm z twierdzenia 5.11 odwiedza co najwyżej $O^*(5^n)$ stanów i z każdego stanu rozpatruje wielomianowo wiele przejść, czyli algorytm ten działa w czasie $O^*(5^n)$ i pamięci $O^*(2^n)$ [1].

5.3. Algorytm dokładny $O(4.829^n)$

Zastosujmy teraz wniosek 5.10 do innego przydziału na kubełki.

Definicja 5.15. *Podziałem podwójnym bez liści* nazwijmy podział na kubełki podobny do opisanego w definicji 4.10, gdzie kubełki mają rozmiar $2b + 2$, a dla liści nie rozbijamy przedziału długości $4b + 4$ na dwa kawałki, lecz zostawiamy im rozmiar kubełka wielkości $4b + 4$.

Uwaga 5.16. Istnieje co najwyżej $n2^{n-l}$ podziałów podwójnych bez liści, gdzie l to liczba liści w rozważanym drzewie rozpinającym. Możemy wygenerować wszystkie takie podziały zgodnie ze schematem przedstawionym w Rozdziale 4.3.2.

Twierdzenie 5.17. *Algorytm który generuje wszystkie podziały podwójne bez liści i dla każdego takiego podziału p korzystając z wniosku 5.10 sprawdza czy istnieje ustawienie π zgodne z p i spełniające* $\mathrm{bw}(\pi) \leqslant b$ *poprawnie rozwiązuje problem decyzyjny* Bandwidth *w czasie $O^*(5^n)$ i pamięci $O^*(4^n)$.*

Dowód. Oszacujmy najpierw złożoność pamięciową przedstawionego algorytmu. Korzystając z faktu 5.6 wiemy, że dla danego podziału na kubełki stanów algorytmu będzie nie więcej niż 5^n, jednakże potrzebujemy dokładniejszego oszacowania. Jedyne wierzchołki dla których rozmiar kubełka wynosi 4 to liście w drzewie rozpinającym. Rozważmy dowolny wierzchołek v, który jest liściem, niech u będzie ojcem v w drzewie rozpinającym. Zauważmy, że jeśli w danym stanie wierzchołek u został już przydzielony, to wierzchołek v może być nieprzydzielony, bądź przydzielony do jednego z **trzech** kubełków (przedziałów wielkości $b + 1$), gdyż gdyby v trafił do pewnego ze skrajnych kubełków, wierzchołki u i v byłyby zbytnio oddalone od siebie. Podobnie jeśli wierzchołek u nie został jeszcze przydzielony, to v nie może być przydzielony do skrajnie lewego kubełka, gdyż nie byłby wtedy spełniony warunek z faktu 5.3. W obu przypadkach mamy tylko cztery możliwości wyboru dla wierzchołka v, co poprawia nasze oszacowanie na liczbę stanów i pokazuje, że złożoność pamięciowa wynosi $O^*(4^n)$.

Przejdźmy teraz do szacowania złożoności obliczeniowej. Pokażemy, że w trakcie działania przedstawionego algorytmu odwiedzanych jest nie więcej niż $3n5^n$ stanów. Niech D będzie drzewem rozpinającym, według którego pracuje algorytm rozrzucania na kubełki. Niech (A, ϕ) będzie stanem odwiedzonym w naszym algorytmie dla podziału na kubełki p. Wartość

[1] Do spamiętywania odwiedzonych stanów możemy użyc słownikowej struktury danych, jednakże możemy również użyć tablicy rozmiaru 2^n nie pogarszając złożoności obliczeniowej. Takiej tablicy nie musimy inicjalizować dla każdego wygenerowanego podziału, wystarczy że w danej komórce tablicy będziemy zapamiętywać numer podziału dla którego dany stan był ostatnio odwiedzony

podziału p dla korzenia D wybieramy na co najwyżej n sposobów i w każdym z nich korzeń może być nieprzydzielony, przydzielony do lewego kubełka lub przydzielony do prawego kubełka, co daje $3n$ możliwości. Weźmy teraz dowolny wierzchołek v niebędący korzeniem drzewa rozpinającego. Jeśli v jest liściem, to podczas szacowania złożoności pamięciowej pokazaliśmy, że dla v mamy co najwyżej 4 możliwości. Załóżmy teraz, że v nie jest liściem, zatem $p(v)$ to przedział dwóch kubełkow, ponadto przez u oznaczmy ojca wierzchołka v w drzewie rozpinającym. Rozważmy przypadki:

1. Jeśli $u \notin A$, to wtedy jeśli w podziale p wierzchołkowi v przypisano dwa prawe kubełki (z czterech możliwych), to mamy 3 możliwości - v jest nieprzydzielony ($v \notin A$), przydzielony do lewego ($v \in A$ oraz $\phi(v)$ to lewy kubełek) bądź prawego kubełka ($v \in A$ oraz $\phi(v)$ to prawy kubełek). Jeśli natomiast w podziale p wierzchołkowi v przypisano dwa lewe kubełki, to v może być nieprzydzielony ($v \notin A$) lub przydzielony do prawego kubełka, odpada nam możliwość, że v jest przydzielony do lewego kubełka, zatem mamy **pięć** wyborów dla v.

2. Jeśli $u \in A$ oraz u został przydzielony przez funkcję ϕ do lewego kubełka, to mamy tylko **cztery** możliwości dla wierzchołka v, mianowicie jeśli w podziale p dla v przypisano lewą parę kubełków, to v może być nieprzydzielony, przydzielony do lewego bądź prawego kubełka, czyli trzy możliwości. Jeśli natomiast w podziale p przypisano dla v prawą parę kubełków (spośród czterech możliwych), to v może być jedynie przydzielony do lewego kubełka (pozostałe dwie możliwości nie spełniają założen faktu 5.3). Sumarycznie mamy **cztery** możliwe wybory dla wierzchołka v.

3. Jeśli $u \in A$ oraz u został przydzielony do prawego kubełka, to ponownie mamy pięć możliwości, jedyna możliwość która odpada, to jeśli w podziale p dla wierzchołka v wybrano lewą parę kubełków oraz p miałby być przydzielony do lewego z nich, wtedy wierzchołki u i v byłyby za daleko od siebie, co wykluczą tą jedną z sześciu możliwosci i pozstawia nam **pięć** możliwości dla wierzchołka v.

We wszystkich przypadkach mamy co najwyżej pięć wyborów dla każdego wierzchołka, co pokazuje, że łączna liczba odwiedzonych przez algorytm stanów jest nie większa niż $3n5^n$ a złożoność obliczeniowa wynosi $O^*(5^n)$. □

Zauważmy, że w powyższej analizie czasami mamy cztery a czasami pięć możliwych wyborów dla danego wierzchołka. Cztery wybory mamy w przypadku gdy v jest liściem, bądź gdy ojciec v został przydzielony do lewego klasycznego kubełka. Podczas powyższych rozważań w obu przypadkach z pięcioma wyborami pewne z tych pięciu wyborów przypisują v do lewego klasycznego kubełka, co sprawia, że dzieci wierzchoła v, niezależnie od tego czy są liśćmi czy też nie, będą miały co najwyżej cztery wybory! Oznacza to, że nie możliwa jest konstrukcja przykładu, w którym dla każdego wierzchołka będziemy mieli dokładnie pięc możliwości i pozwala przypuszczać że złożoność naszego algorytmu można lepiej oszacować. W tym celu użyjemy techniki nazywanej Measure&Conquer (w skrócie M&C), przedstawionej po raz pierwszy przez Fomina, Grandoniego i Kratscha (zob. [8]) do dokładniejszej analizy złożoności przedstawionego przez nich algorytmu dla problemu maksymalnego zbioru niezależnego. Przy naiwnym szacowaniu złożoności można pokazać, że wynosi ona $O(2^{0.406n})$, natomiast użycie metody M&C pozwala oszacować złożoność obliczeniową przez $O(2^{0.288n})$.

Istotą metody M&C jest zdefiniowanie miary egzemplarza problemu. Podczas wcześniejszych szacowań miarą naszego problemu była liczba wierzchołków grafu, czyli n. Zauważyliśmy jednak, że podczas szacowania rekurencyjnego całkowitej liczby stanów wierzchołki które

mają przydzielonego ojca do lewego klasycznego kubełka są dla nas korzystniejsze od pozostałych, gdyż gwarantują one co najwyżej cztery możliwe wybory dla tego właśnie wierzchołka. Z tego powodu dla każdego wierzchołka v przypiszemy wagę:

- rozważone wierzchołki będą miały wagę 0,
- jeśli v jest korzeniem, bądź ojciec v jest nieprzydzielony, to waga wierzchołka v będzie równa 1,
- jeśli ojciec v jest przydzielony do lewego klasycznego kubełka, to wagę wierzchołka v oznaczymy poprzez α $(0 \leqslant \alpha \leqslant 1)$,
- jeśli natomiast ojciec v jest przydzielony do prawego klasycznego kubełka, to waga v będzie równa β $(0 \leqslant \beta \leqslant 1)$.

Jeśli teraz będziemy szacować całkowitą liczbę stanów dla wszystkich podziałów na kubełki, to w przypadku gdy mamy aż pięc możliwych wyborów, niektóre z nich spowodują, że waga dzieci danego wierzchołka zmniejszy się z wartości 1 do α, co spowoduje że rozmiar nowego egzemplarza ulegnie zmiejszeniu. Z kolei w innych przypadkach rozmiar nowego egzemplarza ulegnie zwiększeniu. Dlatego też należy znaleźć takie wartości α i β, które pozwalają na najlepsze oszacowanie złożoności obliczeniowej naszego algorytmu.

Przeanalizujmy teraz wszystkie przypadki. Niech v będzie rozważanym wierzchołkiem, natomiast u będzie jego ojcem w drzewie rozpinającym. Jeśli v jest liściem, to wierzchołek v generuje cztery wybory, w zależności od tego ile wynosi waga wierzchołka v (1, α czy też β) mamy następujące równania szacujące całkowitą liczbę stanów odwiedzonych w naszym algorytmie, którą oznaczmy jako $T(w)$, gdzie w to miara egzemplarza, czyli suma wag jego wierzchołków:

$$T(w) = 4T(w-1) \tag{5.1}$$

$$T(w) = 4T(w-\alpha) \tag{5.2}$$

$$T(w) = 4T(w-\beta) \tag{5.3}$$

Załóżmy teraz, że v nie jest liściem, w takim wypadku v ma co najmniej jedno dziecko w drzewie rozpinającym i przypisanie v do lewego, bądź prawego kubełka spowoduje zmianę wagi co najmniej jednego wierzchołka z wartości 1 na α lub β. Ponownie rozważmy przypadki przedstawione w dowodzie twierdzenia 5.17:

1. Jeśli wierzchołek u jest nieprzydzielony, to mamy pięć możliwości, spośród których dwa prowadzą do nieprzydzielenia v, w dwóch przypadkach v jest przypisany do prawego kubełka, natomiast w jednym przypadku v jest przypisany do lewego kubełka. Wierzchołek v zostaje przetworzony, zatem jego waga zmniejsza się z wartości 1 do 0, co daje nam następujące równanie:

$$T(w) = 2T(w-1) + 2T(w-1-(1-\beta)) + T(w-1-(1-\alpha)) \tag{5.4}$$

2. Analogicznie rozważamy drugi przypadek, czyli gdy u jest przydzielony do lewego kubełka. W takim wypadku mamy cztery możliwości dla wierzchołka v, z czego w jednym przypadku v jest nieprzydzielony, w jednym przydzielony do prawego kubełka oraz w dwóch przydzielony do lewego kubełka, zatem:

$$T(w) = T(w-\alpha) + T(w-\alpha-(1-\beta)) + 2T(w-\alpha-(1-\alpha)) \tag{5.5}$$

3. Jeśli natomiast u jest przydzielony do prawego kubełka, to ponownie mamy pięć możliwości tworzących równanie:

$$T(w) = 2T(w-\beta) + 2T(w-\beta-(1-\beta)) + T(w-\beta-(1-\alpha)) \tag{5.6}$$

Rozwiązaniem równania $T(w) = c_1T(w-w_1) + c_2T(w-w_2) + c_3T(w-w_3)$, $T(w) = 1$ dla $w \leqslant 1$ jest funkcja rzędu $O(x_0^n)$, gdzie x_0 jest największym miejscem zerowym funkcji $1-c_1x^{-w_1}-c_2x^{-w_2}-c_3x^{-w_3}$. Jako że mamy do wyznaczenia wartości jedynie dwóch zmiennych możemy przeszukać dla obu zmiennych przedział $[0,1]$ próbując wszystkich wartości postaci $\frac{a}{10\,000}$, gdzie $a \in \mathbb{Z}$,$0 \leqslant a \leqslant 10\,000$ (gdyby zmiennych do ustalenia było więcej moglibyśmy użyc programowania kwazi-wypukłego, zob. [3]) i wybrać takie wartości α, β dla których x_0 jest najmniejsze

Najlepsze oszacowanie otrzymaliśmy dla wartości $\alpha = 0.8805$ oraz $\beta = 1$ (zob. tabela 5.1)

równanie	5.1	5.2	5.3	5.4	5.5	5.6
x_0	4.000000	4.828037	4.000000	4.828485	4.386510	4.828485

Tabela 5.1: Tabela zawiera największe miejsca zerowe x_0 funkcji odpowiadających kolejnym równaniom dla $\alpha = 0.8805$ oraz $\beta = 1$.

Dla tak dobranych wag otrzymujemy oszacowanie mówiące, że całkowita liczba odwiedzonych stanów dla wszystkich podziałów na kubełki wynosi nie więcej niż $O(4.829^n)$, co prowadzi nas do następującego twierdzenia:

Twierdzenie 5.18. *Przedstawiony algorytm poprawnie rozwiązuje problem* Bandwidth *w czasie* $O(4.829^n)$ *i pamięci* $O^*(4^n)$.

Rozdział 6

Szczególne przypadki — duża szerokość grafu

W tym rozdziale zajmiemy się rozstrzyganiem problemu decyzyjnego BANDWIDTH dla danych $(G = (V, E), b)$ w szczególnym przypadku: gdy b jest stosunkowo duże, np. $b \geqslant \frac{n}{2}$. Będziemy proponować algorytm podobny do algorytmu 5.11, lecz pokażemy, że w drugiej fazie — sprawdzania, czy istnieje ustawienie π zgodne z klasycznym b-podziałem na kubełki p — możemy specjalnie potraktować pierwszy i ostatni kubełek klasyczny.

Będziemy ponownie zakładać, że $n = |V|$. Niech $N(A) = \{u : \exists_{v \in A} uv \in E\}$ zbiorem sąsiadów wierzchołków ze zbioru $A \subset V$.

6.1. Twierdzenie o skrajnym kubełku

Kluczem do dalszych rozważań będzie następująca obserwacja:

Twierdzenie 6.1. *Niech $A \subset V$ będzie podzbiorem mocy $m \leqslant b + 1$. Niech $\pi' : V \setminus A \to \{1, 2, \ldots, n - m\}$ będzie bijekcją — ustawieniem wierzchołków zbioru $V \setminus A$ na pozycjach $\{1, 2, \ldots, n - m\}$, spełniającym $\mathrm{bw}(\pi') \leqslant b$. Wówczas istnieje ustawienie π wierzchołków grafu G, przedłużające π' (czyli $\pi' = \pi|_{V \setminus A}$) spełniające $\mathrm{bw}(\pi) \leqslant b$ wtedy i tylko wtedy, gdy spełniony jest następujący warunek:*

Dla każdej liczby $1 \leqslant k \leqslant n - m$ zbiór

$$X_k = N(\pi'^{-1}(\{1, \ldots, k\})) \cap A$$

jest mocy co najwyżej $\max\{0, k + b - (n - m)\}$.

Dowód. Załóżmy, że istnieje przedłużenie π spełniające $\mathrm{bw}(\pi) \leqslant b$. Weźmy $1 \leqslant k \leqslant n - m$ i niech $v \in V \setminus A$ spełnia $\pi(v) = \pi'(v) \leqslant k$, czyli v jest ustawiony na pozycji nie dalszej niż k-ta. Załóżmy, że $uv \in E$. Skoro $\mathrm{bw}(\pi) \leqslant b$, to $\pi(u) \leqslant k + b$. Jeśli $u \in A$, to $\pi(u) > n - m$, czyli dla u jest dostępnych tylko $(k + b) - (n - m)$ wartości funkcji π, czyli takich u jest co najwyżej $(k + b) - (n - m)$.

W drugą stronę, załóżmy że warunek jest spełniony i spróbujmy skonstruować ustawienie π — musimy je określić na wierzchołkach ze zbioru A. Wprowadźmy pomocniczną funkcję $\mathbf{sasiad} : A \to \{1, 2, \ldots, n - m, \infty\}$ określoną następująco:

$$\mathbf{sasiad}(u) = \min\{i : \pi'^{-1}(i) \text{ jest sąsiadem } u\},$$

czyli pozycja najdalej ustawionego na lewo sąsiada u w ustawieniu π'. W przypadku braku takiego sąsiada, przyjmujemy $\mathbf{sasiad}(u) = \infty$.

Posortujmy elementy A według wartości funkcji **sasiad** — weźmy dowolne ustawienie π przedłużające π', spełniające warunek, że jeśli $\mathbf{sasiad}(u) < \mathbf{sasiad}(v)$ to $\pi(u) < \pi(v)$. Pokażemy, że to ustawienie spełnia $\mathrm{bw}(\pi) \leqslant b$. Załóżmy przeciwnie — niech uv będzie krawędzią, dla której $\pi(u) - \pi(v) > b$. Skoro $\mathrm{bw}(\pi') \leqslant b$, to przynajmniej jeden z wierzchołków u i v musi należeć do A. Skoro $m = |A| \leqslant b + 1$, to jeśli $u, v \in A$, to $\pi(u) - \pi(v) \leqslant b$. Wobec tego, skoro $\pi(u) > \pi(v)$, to $u \in A$ i $v \in V \setminus A$.

Przyjrzyjmy się zbiorowi $X_{\pi(v)}$. Elementy A posortowaliśmy niemalejąco według funkcji **sasiad**, wobec tego jeśli $u' \in A$ i $\pi(u') < \pi(u)$, to $\mathbf{sasiad}(u') \leqslant \mathbf{sasiad}(u)$, czyli u' ma sąsiada ustawionego na pozycji nie dalszej niż $\pi(v)$, czyli $u' \in X_{\pi(v)}$. Wobec tego

$$\{\pi^{-1}(n-m+1), \pi^{-1}(n-m+2), \ldots, u\} \subset X_{\pi(v)},$$

czyli

$$|X_{\pi(v)}| \geqslant \pi(u) - (n-m) > \pi(v) + b - (n-m)$$

co jest sprzeczne z założeniem. □

Wniosek 6.2. Powyższe twierdzenie oczywiście można zapisać „w drugą stronę", tj. gdzie π' jest ustawieniem elementów zbioru $V \setminus A$ na pozycjach $\{m+1, m+2, \ldots, n\}$, zaś warunek równoważny jest symetryczny.

6.2. Zapytania o szerokość większą niż $\frac{n}{2}$ i $\frac{n}{3}$

Użyjemy twierdzenia 6.1 do pokazania dwóch prostych algorytmów: rozstrzygających problem Bandwidth dla $b \geqslant \frac{n}{2}$ w czasie $O^*(2^{2n-b}) = O^*(2^{\frac{3n}{2}}) = O(2.83^n)$ oraz dla $\frac{n}{2} > b \geqslant \frac{n}{3}$ w czasie $O^*(4^n)$.

Fakt 6.3. *Jeśli $b \geqslant \frac{n}{2}$, to klasycznych podziałów na kubełki jest nie więcej niż 2^n. Po prostu są tylko dwa kubełki.*

Twierdzenie 6.4. *Dla $b \geqslant \frac{n}{2}$ istnieje algorytm działający w czasie $O^*(2^{2n-b}) = O^*(2^{\frac{3n}{2}}) = O(2.83^n)$ i pamięci $O^*(2^{n-b}) = O^*(2^{\frac{n}{2}}) = O(1.42^n)$ rozstrzygający decyzyjny problem* Bandwidth.

Dowód. Sprawdzamy każde możliwe rozstawienie wierzchołków do lekko zmodyfikowanych kubełków klasycznych: kubełkami są zbiory $\{1, 2, \ldots, n-b\}$ oraz $\{n-b+1, \ldots, n\}$. Zauważmy, że drugi zbiór ma b elementów, a pierwszy $n - b \leqslant b$. Podziałów jest $\binom{n}{b} \leqslant 2^n$. Niech A będzie zbiorem wierzchołków przypisanych do prawego kubełka ($|A| = b$), zaś $V \setminus A$ — zbiorem wierzchołków przypisanych do lewego kubełka. Wobec tego w dalszej części algorytmu rozważamy jedynie krawędzie pomiędzy A i $V \setminus A$.

Będziemy konstruować ustawienie $\pi' : V \setminus A \to \{1, 2, \ldots, n-b\}$ spełniające warunek z twierdzenia 6.1. Zauważmy, iż ten warunek dotyczy zbiorów $\pi'^{-1}(\{1, 2, \ldots, k\})$ dla $1 \leqslant k \leqslant n - b$ — dla każdego takiego zbioru ma zachodzić $|X_k| \leqslant (k + b) - (n - b)$ — ten warunek, mając dany zbiór $\pi'^{-1}(\{1, 2, \ldots, k\})$, jest sprawdzalny w czasie wielomianowym.

Pokażemy, że sprawdzenie istnienia π' można dokonać w czasie i pamięci $O^*(2^{n-b})$ korzystając z algorytmu przechodzącego stany (podzbiory $V \setminus A$) i szukającego ścieżki między $\emptyset$ a $V \setminus A$. Dokładniej, *stanem* w naszym algorytmie będziemy nazywali dowolny podzbiór $B \subset V \setminus A$ spełniający warunek z twierdzenia 6.1 dla $k = |B|$, tzn. zbiór $N(B) \cap A$ ma moc co najwyżej $(|B| + b) - (n - b)$. Między dwoma stanami $B, B' \subset V \setminus A$ jest przejście, jeśli $|B'| = |B| + 1$ i $B \subset B'$. Wówczas, jeśli istnieje ustawienie π', to wszystkie zbiory $\pi'^{-1}(\{1, 2, \ldots, k\})$ dla $0 \leqslant k \leqslant n - b$ są stanami i między kolejnymi stanami są przejścia; co

więcej, mając taki ciąg stanów $\emptyset = B_0 \subset B_1 \subset \ldots \subset B_{n-b} = V \setminus A$, kładąc $\pi^{-1}(v) = j$ gdzie $\{v\} = B_j \setminus B_{j-1}$, mamy ustawienie π' spełniające warunek z twierdzenia 6.1. Istnienie takiego ustawienia π', na mocy twierdzenia 6.1, jest równoważne istnieniu ustawienia π zgodnego z podziałem na kubełki i spełniającego $\mathrm{bw}(\pi) \leqslant b$.

Wobec tego, w czasie i pamięci $O^*(2^{n-b})$, możemy sprawdzić powyższym algorytmem, czy takie π istnieje. Sprawdzając dla każdego podziału na kubełki, otrzymujemy algorytm działający w czasie $O^*(2^{2n-b})$ i pamięci $O^*(2^{n-b})$. □

Teraz zajmijmy się przypadkiem $\frac{n}{2} > b \geqslant \frac{n}{3}$. Jest on tak naprawdę bardzo podobny do poprzedniego: dzielimy wierzchołki na trzy kubełki klasyczne (wielkości b, b i $2n - b \leqslant b$) a następnie robimy podobne przeszukiwanie dla środkowego kubełka jak w poprzednim algorytmie, lecz sprawdzając warunki z twierdzenia 6.1 zarówno w lewo, jak i w prawo.

Fakt 6.5. *W przypadku $\frac{n}{2} > b \geqslant \frac{n}{3}$ klasycznych podziałów na kubełki jest nie więcej niż $\binom{n}{b}\binom{n-b}{b} < 2^{2n-b}$. Ze wszystkich wierzchołków wybieramy b do pierwszego kubełka, a następnie z pozostałych wybieramy b do drugiego kubełka.*

Twierdzenie 6.6. *Istnieje algorytm działający w czasie $O^*(4^n)$ i pamięci $O^*(2^b)$ rozstrzygający w przypadku $\frac{n}{2} > b \geqslant \frac{n}{3}$ problem decyzyjny* Bandwidth.

Dowód. Sprawdzamy każde możliwe rozstawienie wierzchołków do kubełków klasycznych: kubełkami są zbiory $\{1, 2, \ldots, b\}$, $\{b+1, b+2, \ldots, 2b\}$ oraz $\{2b+1, 2b+2, \ldots, n\}$, każdy z nich jest nie większy niż b. Wierzchołki przyporządkowane do pierwszego kubełka oznaczamy A', do trzeciego A, zaś do drugiego $V' = V \setminus A \setminus A'$. Celem naszym jest sprawdzenie, czy istnieje ustawienie π zgodne z tym podziałem spełniające $\mathrm{bw}(\pi) \leqslant b$. Wobec tego podział na kubełki klasyczne odrzucamy, jeśli istnieje krawędź między A' i A: wówczas na pewno takie π nie istnieje. Krawędziami wewnątrz kubełków nie powinniśmy się przejmować, gdyż kubełki mają wielkość nie większą niż b. Interesujące dla nas są krawędzie między V' a $A \cup A'$.

Na mocy twierdzenia 6.1 oraz wersji symetrycznej — wniosku 6.2, ustawienie $\pi' : V' \to \{b+1, \ldots, 2b\}$ rozszerza się do ustawienia π wtedy i tylko wtedy, gdy zachodzą warunki z twierdzenia 6.1 dla zbiorów $\pi'^{-1}(\{b+1, \ldots, b+k\})$ oraz warunki symetryczne z wniosku 6.2 dla zbiorów $\pi^{-1}(\{2b, 2b-1, \ldots, 2b-k+1\}$ dla $0 \leqslant k \leqslant b$. Uznamy więc, że zbiór $B \subset V'$ jest *stanem* algorytmu, jeśli zbiór B spełnia warunek z twierdzenia 6.1 dla $k = |B|$, czyli:

$$|N(B) \cap A| \leqslant |B|,$$

oraz zbiór $V' \setminus B$ spełnia warunek odwrotny z wniosku 6.2, czyli:

$$|N(B) \cap A'| \leqslant |V' \setminus B| = b - |B|.$$

Przejście, tak jak poprzednio, będzie parą stanów różniących się o jeden wierzchołek. Analogicznie jak w poprzednim twierdzeniu, istnienie odpowiedniego ustawienia $\pi' : V' \to \{b+1, b+2, \ldots, 2b\}$ odpowiada istnieniu ścieżki stanów, pomiędzy którymi są przejścia, od stanu $\emptyset$ do stanu V'. Tego zaś możemy, spamiętując odwiedzone stany, dokonać w czasie i pamięci $O^*(2^b)$.

W sumie, jeśli każdy podział na kubełki klasyczne sprawdzimy w czasie i pamięci $O^*(2^b)$ (czy istnieje ustawienie π zgodne z tym podziałem), to otrzymamy algorytm działający w pamięci $O^*(2^b)$ i czasie $O^*(2^b 2^{2n-b}) = O^*(4^n)$. □

6.3. Pomijanie skrajnych kubełków

Na koniec tego rozdziału warto zauważyć, iż obserwację z twierdzenia 6.1 można zastosować do ogólnego algorytmu, przedstawionego w twierdzeniu 5.11.

Twierdzenie 6.7. *Dla dowolnego b istnieje algorytm rozstrzygający w czasie $O^*(3^n 2^{n-b})$ i w pamięci $O^*(2^{n-b})$ problem decyzyjny* Bandwidth.

Dowód. Sprawdzamy wszystkie podziały na kubełki klasyczne, których jest $O^*(3^n)$. Dla każdego takiego podziału wykonujemy algorytm opisany w twierdzeniu 5.11, lecz z pominięciem pierwszego i ostatniego kubełka. Dodatkowo o *stanie* $O(A, \phi)$ algorytmu zakładamy, że $K_1 \setminus \phi^{-1}(1)$ (drugi kubełek) spełnia warunek z wniosku 6.2 dla $A = K_0$, zaś $\phi^{-1}(\lceil \frac{n}{b+1} \rceil - 2)$ (przedostatni kubełek) spełnia warunek z twierdzenia 6.1 dla $A = K_{\lceil \frac{n}{b+1} \rceil - 1}$. □

Uwaga 6.8. W tym algorytmie nie potrafimy przeprowadzić tak precyzyjnej analizy, jaka prowadziła do oszacowania złożoności algorytmu z twierdzenia 5.11 do $O^*(5^n)$. Jest tak dlatego, iż nie uwzględniamy wszystkich wierzchołków w czasie przeszukiwania i analiza znacznie się komplikuje.

Uwaga 6.9. Dla $b \geqslant \frac{n}{3}$ ten algorytm sprowadza się do algorytmów opisanych w sekcji 6.2.

Uwaga 6.10. Dla $b < \frac{n}{3}$ ten algorytm jest gorszy od algorytmu $O^*(5^n)$. Dla $b \geqslant \frac{n}{4}$ zachodzi $O^*(3^n 2^{n-b}) = O(5.05^n)$. Ten algorytm ma więc sens tylko dla przypadków opisanych w sekcji 6.2 albo gdy zależy nam na złożoności pamięciowej.

Rozdział 7

Podsumowanie

7.1. Pytania otwarte

W tym rozdziale chcieliśmy się zastanowić, jak dalej można badać problem Bandwidth i jakie pytania pojawiają się w świetle nowych wyników. W tym rozdziale będziemy stawiali pytania — problemy otwarte, na które nie znamy odpowiedzi. Oczywiście, nieśmiertelnym pytaniem pozostaje:

Pytanie 7.1. Zaproponować algorytm, rozwiązujący problem Bandwidth dokładnie, działający w czasie szybszym niż $O^*(4.83^n)$.

Wydaje się, że ewentualne poprawki do złożoności mogą nie być bardzo trudne, zwłaszcza, że wynik 4.83 jest wynikiem pochodzącym z pierwszej analizy Measure & Conquer w problemie Bandwidth. Przypomnijmy również, że w tym algorytmie złożoność pamięciowa wynosi $O^*(4^n)$, która eliminuje ten wynik z wszelkich zastosowań praktycznych. Dlatego następujące pytanie wydaje się naturalne:

Pytanie 7.2. Zaproponować algorytm, rozwiązujący problem Bandwidth dokładnie, działający w pamięci $O^*(2^n)$ i czasie lepszym niż $O^*(5^n)$.

Do tego warto zauważyć, że algorytm Feige i Kiliana, opisany w rozdziale 3, działa w wielomianowej pamięci. Wciąż więc otwarte jest pytanie:

Pytanie 7.3. Zaproponować algorytm, rozwiązujący problem Bandwidth dokładnie, działający w wielomianowej pamięci i czasie lepszym niż $O^*(10^n)$.

Przyglądając się algorytmom z rozdzialu 5 nasuwa się pytanie, czy drugiej fazy algorytmu, czyli rozpoznawania, czy istnieje ustawienie zgodne z danym podziałem klasycznym o szerokości nie większej niż b, nie da się zrobić w czasie szybszym niż $O^*(2^n)$. W szczególności, może istnieje uogólniona wersja twierdzenia 6.1, pozwalająca przeprowadzić drugą fazę algorytmu w czasie np. $O^*(2^{2b})$, $O^*(2^b)$ lub nawet wielomianowym. Ciekawym więc jest pytanie:

Pytanie 7.4. Czy problem stwierdzania, czy istnieje ustawienie o szerokości nie większej niż b, zgodne z danym podziałem klasycznym, jest rozwiązywalny w czasie wielomianowym?

Przejdźmy teraz do algorytmów aproksymacyjnych. Pokazaliśmy w pracy pewną klasę algorytmów aproksymacyjnych; niestety, biorąc dużą stałą aproksymacji w tych algorytmach, okazują się one dużo gorsze od algorytmu aproksymacyjnego Feige, działającego w czasie wielomianowym i aproksymującego z współczynnikiem $O(\log^3 n\sqrt{\log n \log\log n})$. Pytanie jest więc:

Pytanie 7.5. Czy istnieje lepszy schemat aproksymacji ze stałym współczynnikiem, który dla dużej stałej aproksymacji przewyższa algorytm Feige, działając dla takich danych w czasie wielomianowym?

Na koniec, chcieliśmy zadać kilka pytań o redukcję, która — jak na razie — wydaje się bardzo skomplikowana i nieefektywna dla problemu Bandwidth.

Pytanie 7.6. Czy istnieje redukcja o lepszym współczynniku redukcji rozmiaru do straty na optymalnym rozwiązaniu? Czy można wykazać jakieś nietrywialne dolne ograniczenie na ten stosunek?

7.2. Co kto zrobił

W tym rozdziale, w celach formalnych, spróbujemy podzielić, które wyniki są zasługą którego z autorów pracy.

Pierwszy algorytm dokładny, opisany w rozdziale 5, został wymyślony wspólnie w czasie podróży po Rosji. Marek zaproponował podwójny podział na kubełki, który prowadził do drugiego algorytmu dokładnego. Wtedy o obu algorytmach myśleliśmy, że działają w czasie $O^*(6^n)$. Marcin zaproponował lepszą, dość skomplikowaną analizę obu algorytmów, która po chwili dyskusji uprościła się do analizy $O^*(5^n)$. Następnie Marek użył technik Measure & Conquer i oszacował złożoność drugiego algorytmu przez $O^*(4.83^n)$.

Schemat aproksymacyjny $(4k-1)$-aproksymacji został wymyślony przez Marcina, pierwotnie w trochę gorszej wersji, a następnie ulepszony przez Marka do obecnej postaci. Redukcja została wymyślona przez Marka oraz Łukasza Kowalika. Specjalne przypadki w rodziale 6 opracował Marcin.

Bibliografia

[1] S. Assman, G. Peck, M. Syslo, and J. Zak. The bandwidth of caterpillars with hairs of length 1 and 2. *SIAM J. Algebraic Discrete Methods*, 2:387–393, 1981.

[2] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[3] David Eppstein. Quasiconvex analysis of backtracking algorithms. In *Proc. 15th Symp. Discrete Algorithms*, pages 781–790. ACM and SIAM, January 2004.

[4] U. Feige. Approximating the bandwidth via volume respecting embeddings. *J. Comput. Syst. Sci.*, 60(3):510–539, 2000.

[5] U. Feige. Coping with the NP-hardness of the graph bandwidth problem. *SWAT*, pages 10–19, 2000.

[6] Uriel Feige and Joe Kilian. Zero knowledge and the chromatic number. *J. Comput. Syst. Sci.*, 57(2):187–199, 1998.

[7] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS*, 87:47–77.

[8] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In *SODA'06*, pages 18–25, 2006.

[9] M. Garey, R. Graham, D. Johnson, and D. Knuth. Complexity results for bandwidth minimization. *SIAM J. Appl. Math.*, 34:477–495, 1978.

[10] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999.

[11] D. Kleitman and R. Vohra. Computing the bandwidth of interval graphs. *SIAM J. Discrete Math.*, 3:373–375, 1990.

[12] B. Monien. The bandwidth-minimization problem for caterpillars with hairlength 3 is np-complete. *SIAM J. Algebraic Discrete Methods*, 7:505–512, 1986.

[13] Marcin Pilipczuk and Marek Cygan. Faster exact bandwidth. (przyjete na WG 2008).

[14] Marcin Pilipczuk, Lukasz Kowalik, Mateusz Wykurz, and Marek Cygan. Exponential-time approximation via instance reduction. (w przygotowaniu).

[15] J. Saxe. Dynamic programming algorithms for recognizing small-bandwidth graphs in polynomial time. *SIAM Journal on Algebraic Methods*, 1:363–369, 1980.

[16] W. Unger. The complexity of the approximation of the bandwidth problem. *FOCS*, pages 82–91, 1998.

[17] Gerhard Woeginger. Exact algorithms for np-hard problems: A survey. *Combinatorial Optimization — Eureka, you shrink!*, 2570:185–207, 2003.

[18] Gerhard Woeginger. Space and time complexity of exact algorithms: Some open problems. In *Proceedings of the 1st International Workshop on Parametrized and Exact Computation*, pages 281–290, 2004.

[19] Gerhard Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008.