AUTOREFERAT

1. **Imię i Nazwisko**:

Anna Zych-Pawlewicz

2. **Posiadane dyplomy, stopnie naukowe**:

- Magister informatyki, Uniwersytet Jagielloński, 2004
- Magister informatyki, Vrije Universiteit Amsterdam, 2004
- Doktor nauk, dyscyplina: informatyka, ETH Zürich, 2012
  Tytuł rozprawy: *Reoptimization of NP-hard problems*

3. **Informacje o dotychczasowym zatrudnieniu w jednostkach naukowych**:

- 2004–2005: University of Twente, Entschede, Holandia
- 2006–2007: Philips Research Campus, Eindhoven, Holandia
- 2007–2012: Swiss Federal Institute of Technology (ETH), Zürich, Switzerland
- 2012-2013 Uniwersytet Warszawski, postdoc
- 2013-dziś Uniwersytet Warszawski, adiunkt

4. **Omówienie osiągnięć, o których mowa w art. 219 ust. 1 pkt. 2 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2021 r. poz. 478 z późn. zm.).**

a) *Tytuł osiągnięcia naukowego:*

Model ograniczonego budżetu zmian jako pomost pomiędzy algorytmami online a dynamicznymi.

b) *Lista prac wchodzących w skład osiągnięcia naukowego:*

[A1] Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, Anna Zych.
Online bipartite matching in offline time.
55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014), 384–393, IEEE Computer Society, 2014.
`https://doi.org/10.1109/FOCS.2014.48`

[A2] Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, Anna Zych-Pawlewicz.
Shortest Augmenting Paths for Online Matchings on Trees.
Theory of Computing Systems, 62 (2), 337–348, 2018.
`https://doi.org/10.1007/s00224-017-9838-x`

Wersja konferencyjna[1]:
Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, Anna Zych.
Shortest Augmenting Paths for Online Matchings on Trees.

[1]Odniesienia do konkretnych twierdzeń numerowane są według wersji opublikowanej w czasopiśmie.

13th International Workshop on Approximation and Online Algorithms (WAOA 2015), Lecture Notes in Computer Science, 9499, 59–71, Springer, 2015.
https://doi.org/10.1007/978-3-319-28684-6_6

[A3] Bartłomiej Bosek and Dariusz Leniowski and Piotr Sankowski and Anna Zych-Pawlewicz.
A Tight Bound for Shortest Augmenting Paths on Trees.
Theoretical Computer Science, 901, 45–61, 2022.
https://www.sciencedirect.com/science/article/pii/S0304397521007003

Wersja konferencyjna[2]:
Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, Anna Zych-Pawlewicz.
A Tight Bound for Shortest Augmenting Paths on Trees.
13th Latin American Theoretical Informatics Symposium (LATIN 2018),
LATIN 2018: Theoretical Informatics, 10870, 201–216, Springer, 2018.
https://doi.org/10.1007/978-3-319-77404-6_16

[B1] Jakub Łącki, Jakub Oćwieja, Marcin Pilipczuk, Piotr Sankowski, Anna Zych-Pawlewicz.
The Power of Dynamic Distance Oracles: Efficient Dynamic Algorithms for the Steiner Tree.
Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing (STOC) 11–20, 2015.
https://doi.org/10.1145/2746539.2746615

Pełny dowód jest zaprezentowany na repozytorium arXiv[3]:
Jakub Łącki, Jakub Oćwieja, Marcin Pilipczuk, Piotr Sankowski, Anna Zych-Pawlewicz.
The Power of Dynamic Distance Oracles: Efficient Dynamic Algorithms for the Steiner Tree.
Computing Research Repository, abs/1308.3336, 2016.
https://arxiv.org/abs/1308.3336

[C1] Bartłomiej Bosek, Yann Disser, Andreas Emil Feldmann, Jakub Pawlewicz, Anna Zych-Pawlewicz.
Recoloring Interval Graphs with Limited Recourse Budget.
17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020) Leibniz International Proceedings in Informatics (LIPIcs) 162, 17:1 – 17:23, 2020.
https://drops.dagstuhl.de/opus/volltexte/2020/12264

[C2] Bartłomiej Bosek, Anna Zych-Pawlewicz.
Recoloring Unit Interval Graphs with Logarithmic Recourse Budget.
30th Annual European Symposium on Algorithms (ESA), 244, 25:1–25:14, 2022.
https://doi.org/10.4230/LIPIcs.ESA.2022.25

Pełny dowód jest zaprezentowany na repozytorium arXiv[4]:
Bartłomiej Bosek, Anna Zych-Pawlewicz.
Recoloring Unit Interval Graphs with Logarithmic Recourse Budget.
Computing Research Repository, abs/2202.08006, 2022.
https://arxiv.org/abs/2202.08006

[D1] Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartłomiej Wróblewski, Anna Zych-Pawlewicz.
Efficient fully dynamic elimination forests with applications to detecting long paths and

[2]Odniesienia do konkretnych twierdzeń numerowane są według wersji opublikowanej w czasopiśmie.
[3]Odniesienia do konkretnych twierdzeń numerowane są według wersji arXiv.
[4]Odniesienia do konkretnych twierdzeń numerowane są według wersji konferencyjnej.

cycles.
ACM-SIAM Symposium on Discrete Algorithms (SODA), 796-809, 2021.
https://epubs.siam.org/doi/abs/10.1137/1.9781611976465.50

Pełny dowód jest zaprezentowany na repozytorium arXiv[5]:
Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartłomiej Wróblewski, Anna Zych-Pawlewicz.
Efficient fully dynamic elimination forests with applications to detecting long paths and cycles.
Computing Research Repository, abs/2006.00571, 2020.
https://arxiv.org/abs/2006.00571

[D2] Jędrzej Olkowski, Michał Pilipczuk, Mateusz Rychlicki, Karol Węgrzycki, Anna Zych-Pawlewicz.
Dynamic Data Structures for Parameterized String Problems.
40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023), 46:1–46:13, 2023.
https://drops.dagstuhl.de/opus/volltexte/2023/17702

Pełny dowód jest zaprezentowany na repozytorium arXiv[6]:
Jędrzej Olkowski, Michał Pilipczuk, Mateusz Rychlicki, Karol Węgrzycki, Anna Zych-Pawlewicz.
Dynamic Data Structures for Parameterized String Problems.
Computing Research Repository, abs/2205.00441, 2022.
https://arxiv.org/abs/2205.00441

c) *Omówienie celu naukowego ww. prac i osiągniętych wyników wraz z omówieniem ich ewentualnego wykorzystania*

Krótki zarys

Niezwykle ważnym tematem we współczesnej algorytmice jest badanie podstawowych własności dynamicznie zmieniających się sieci, zwanych grafami dynamicznymi. W rzeczywistych sytuacjach połączenia i węzły sieci mogą zanikać lub pojawiać się i ważne jest posiadanie skutecznego sposobu monitorowania podstawowych właściwości takiej sieci. Narzędziami do osiągnięcia tego celu są grafowe algorytmy online oraz dynamiczne.

Tematem przewodnim niniejszego opracowania jest pewien kompromis pomiędzy algorytmami online a algorytmami dynamicznymi, który figuruje w literaturze pod nazwą algorytmów dynamicznych z ograniczonym budżetem na liczbę zmian. Model ten jest interesujący zarówno z punktu widzenia algorytmów online jak i z punktu widzenia algorytmów dynamicznych. Można nawet pokusić się o stwierdzenie, że model ten jest pomostem pomiędzy tymi dwoma dziedzinami.

W grafowych algorytmach online najczęściej przyjmuje się model inkrementalny, gdzie dynamiczny graf zmienia się w czasie poprzez dodawanie do niego kolejnych wierzchołków bądź krawędzi. Dla zadanego problemu obliczeniowego celem jest utrzymywanie rozwiązania po każdej aktualizacji grafu. Co ważne, w konstruowanym rozwiązaniu nie można wycofywać się z uprzednio podjętych decyzji, co oznacza że rozwiązanie każdej kolejnej instancji jest rozszerzeniem rozwiązania dla poprzedniej. Takie ograniczenie algorytmu online podyktowane jest naturą niektórych problemów obliczeniowych. Klasycznym przykładem jest tu szeregowanie zadań, które pojawiają się w czasie, muszą zostać przydzielone natychmiast, zaś wywłaszczanie jest zbyt

[5]Odniesienia do konkretnych twierdzeń numerowane są według wersji konferencyjnej.
[6]Odniesienia do konkretnych twierdzeń numerowane są według wersji konferencyjnej.

kosztowne. Ograniczenie to prowadzi do wielu bardzo pesymistycznych ograniczeń dolnych na jakość rozwiązania możliwego do utrzymywania w modelu online - bardzo często rozwiązanie to musi być dalekie od optymalnego. Naturalnym pytaniem jest czy dolne ograniczenia będą równie pesymistyczne, jeśli algorytm będzie miał możliwość wycofania się z ograniczonej liczby wcześniejszych decyzji. Równie naturalnym pytaniem jest próba uzależnienia jakości rozwiązania od ilości dostępnych zmian decyzji. Model z ograniczonym budżetem zmian narzuca algorytmowi ograniczenie (czyli budżet) na liczbę dostępnych zmian decyzji, i pyta jak dobre może być rozwiązanie przy zadanym budżecie zmian. Jak się okaże, często nawet bardzo niewielki budżet zmian pozwala uzyskać dużo lepsze rozwiązanie. Nawet w przypadku typowych problemów online, jak szeregowanie zadań, kilka zmian w postaci wywłaszczeń może być niewygórowaną ceną za znaczącą poprawę jakości rozwiązania. W obu wspomnianych modelach: online oraz z budżetem zmian, punktem ciężkości jest teoretyczna możliwość utrzymywania rozwiązania o zadanej jakości, nie zaś ewentualna złożoność czasowa algorytmu.

Złożoność czasowa jest zaś kluczowym kryterium w grafowych algorytmach dynamicznych. Tu również wejściem dla algorytmu jest dynamiczny graf, zaś algorytm powinien utrzymywać informację na temat rozwiązania zadanego problemu obliczeniowego. Informacja utrzymywana przez algorytm powinna umożliwiać efektywne zadawanie zapytań o rozwiązanie, powinna ona również być efektywnie aktualizowana po każdej zmianie dynamicznego grafu. Poza jakością utrzymywanego rozwiązania, głównym celem jest tu efektywność aktualizacji danych oraz efektywność zapytań o rozwiązanie. Warto zaznaczyć, że nie wymagamy tu aby rozwiązanie zaktualizowanej instancji było w jakiś sposób podobne do rozwiązania przed aktualizacją. Nie ma tu ograniczeń na ilość zmian wprowadzanych do rozwiązania: często zdarza się że rozwiązanie utrzymywane jest niejawnie, i przy użyciu odpowiednich struktur danych za pomocą pojedynczej operacji można zmienić w zasadzie wszystkie przeszłe decyzje algorytmu. Dlatego pesymistyczne ograniczenia dolne dla algorytmów online nie mają tu zastosowania, zaś utrzymywane rozwiązanie jest optymalne lub bardzo bliskie optymalnemu. Również rozważany wachlarz aktualizacji grafu jest szerszy w modelu dynamicznym niż w modelu online. Najbardziej pożądane są tu tzw. algorytmy *w pełni dynamiczne*, w których w ramach aktualizacji grafu można zarówno dodawać jak i usuwać krawędzie lub wierzchołki. Algorytmy *inkrementalne* pozwalają wyłącznie na dodawanie krawędzi lub wierzchołków, zaś algorytmy *dekrementalne* pozwalają wyłącznie na usuwanie. Algorytmy z ograniczonym budżetem zmian są szczególnym rodzajem algorytmów dynamicznych, gdzie za pomocą budżetu zmian mierzymy podobieństwo rozwiązań przed i po aktualizacji. Są one też bardzo użytecznym narzędziem przy projektowaniu algorytmów dynamicznych. Mając dany algorytm z ograniczonym budżetem zmian, aby uzyskać efektywny algorytm dynamiczny wystarczy zaimplementować efektywne znajdowanie tych zmian.

Prawie wszystkie prace będące częścią cyklu przykładają model ograniczonego budżetu zmian do konkretnego problemu obliczeniowego, oraz pokazują wpływ tego modelu na algorytmy online i dynamiczne. W badanych problemach ograniczony budżet zmian jest również pożądany z praktycznego punktu widzenia. W pracach [A1, A2, A3] poruszany jest problem skojarzenia w grafach odpowiednio dwudzielnych [A1] oraz będących drzewami [A2, A3], w pracy [B1] jest to problem drzewa Steinera w grafach ogólnych oraz planarnych, w pracach [C1, C2] jest to problem kolorowania grafu w grafach przedziałowych [C1] oraz równo-przedziałowych [C2], zaś w pracy [D1] są to problemy głębokości drzewiastej, k-ścieżki oraz k-cyklu. Praca [D2] porusza dynamiczne parametryzowane problemy tekstowe i jest jedyną w cyklu, gdzie budżet zmian nie gra kluczowej roli. Wszystkie rozważane problemy są flagowymi problemami w algorytmice, mającymi szerokie zastosowania. Wyniki wspomnianych prac pokazują moc narzędzia jakim jest ograniczony budżet zmian przy badaniu algorytmów dynamicznych: w prawie każdym przypadku zaproponowana została efektywna implementacja algorytmu z ograniczonym budżetem zmian, która daje najszybszy znany algorytm dynamiczny dla zadanego problemu.

Artykuły [A1], [A2] i [A3] poruszają problem utrzymywania maksymalnych skojarzeń we wzrastających grafach dwudzielnych. W problemie tym graf dwudzielny pomiędzy n klientami i n serwerami prezentowany jest algorytmowi w sposób inkrementalny. Klienci pojawiają się w dowolnym porządku i żądają połączenia z jednym z preferowanych przez siebie serwerów. Wszystkie zaś serwery dostępne są od początku do końca działania algorytmu. Celem jest utrzymywanie optymalnego skojarzenia, co oznacza że nowo przybyły klient musi zostać skojarzony jeśli istnieje taka możliwość. Innymi słowy, jeśli po przybyciu nowego klienta istnieje w grafie ścieżka powiększająca skojarzenie, to algorytm musi to skojarzenie powiększyć. Budżetem zmian towarzyszącym tej operacji będzie liczba klientów którzy zmuszeni są zmienić serwer. Drugim celem jest zminimalizowanie łącznego budżetu zmian. Jest to dobry przykład problemu, gdzie ograniczenie budżetu zmian jest pożądane również z praktycznego punktu widzenia. Problem ten ma szeroki wachlarz zastosowań: od Google Ads poprzez szeregowanie zadań aż po haszowanie. We wszystkich tych zastosowaniach można wyróżnić zbiór klientów (w szeregowaniu zadań klientami będą zadania, w haszowaniu obiekty do zahaszowania), którzy pojawiają się online i żądają przydziału do pewnego podzbioru serwerów. We wszystkich tych przypadkach nie chcemy odmówić klientowi serwisu, a zarazem chcemy zmienić przydzielony już serwer jak najmniejszej liczbie klientów.

W obliczu tych praktycznych zastosowań problem został szeroko zbadany w klasycznym modelu online. W modelu tym najprostszy algorytm zachłanny osiąga współczynnik aproksymacji $1/2$ i żaden deterministyczny algorytm nie jest w stanie osiągnąć lepszego współczynnika (co dość łatwo zauważyć). Sytuacja jest ciekawsza jeśli dopuścimy algorytm randomizowany. W przełomowej pracy [98] Karp *i inni* przedstawili randomizowany algorytm online (badany także w [56, 24]), który gwarantuje współczynnik aproksymacji $(1 - 1/e)$. Autorzy pokazują również, że tego współczynnika nie da się poprawić.

Dość naturalnym pomysłem pozwalającym przezwyciężyć ograniczenia modelu online jest zbadanie problemu pod kątem ograniczonego budżetu zmian. W szczególności uwaga badaczy skupiła się na pytaniu ile zmian potrzebne jest, aby utrzymywać skojarzenie optymalne. Niemniej jednak do momentu powstania pracy [A1] najmniejszy znany łączny budżet zmian dla tego problemu to oczywiste $\mathcal{O}(n^2)$. Dolnym oszacowaniem jest równie oczywiste $\Omega(n \log n)$. Chaudhuri *i inni* w pracy [48] pytają czy $\mathcal{O}(n \log n)$ jest także górnym oszacowaniem na łączny budżet zmian. W artykule [A1] z Boskiem, Leniowskim i Sankowskim prezentujemy algorytm, który zmienia skojarzenia każdego serwera co najwyżej $\mathcal{O}(\sqrt{n})$ razy. Kluczem jest tu wprowadzenie pojęcia zużycia serwera, które odzwierciedla ile razy dany serwer zmienił skojarzenie. W uproszczeniu algorytm powiększa skojarzenie wzdłuż ścieżek które wybierają jak najmniej zużyte serwery. W efekcie uzyskujemy całkowity budżet zmian rzędu $\mathcal{O}(n^{3/2})$. Wynik ten stanowi pierwszy nietrywialny postęp w kierunku rozwiązania hipotezy Chaudhuri *i innych*. Co ciekawe, nasze techniki pozwalają pokazać, że nawet liniowy łączny budżet zmian rzędu $\mathcal{O}(\epsilon^{-1}n)$ (czyli średnio stały budżet $\mathcal{O}(\epsilon^{-1})$ na każde dodanie wierzchołka) pozwala utrzymywać skojarzenie prawie optymalne, a mianowicie $(1 - \epsilon)$-aproksymację. Dodatni parametr ϵ można tu wybrać dowolnie.

Wyniki te są też sukcesem z punktu widzenia algorytmów dynamicznych. Prezentujemy algorytm, który utrzymuje optymalne skojarzenie, znajdując oraz wykonując wszystkie potrzebne zmiany w łącznym czasie $\mathcal{O}(\sqrt{n}m)$, gdzie m to łączna liczba krawędzi grafu. Daje to algorytm równie szybki jak wówczas [7] najlepszy znany statyczny algorytm Hopcrofta-Karpa [91], ale z tą różnicą, że nasz dodatkowo działa w modelu przyrostowym. Prezentujemy również algorytm który utrzymuje $(1 - \epsilon)$-aproksymację w łącznym czasie $\mathcal{O}(\epsilon^{-1}m)$.

Innym naturalnym pomysłem aby zminimalizować budżet zmian dla tego problemu jest poprawianie skojarzenia po najkrótszych ścieżkach powiększających. Pomysł ten pojawia się już w statycznym algorytmie Edmondsa i Karpa [62] i pozwala pokazać pierwszy silnie wielomianowy

[7] dopiero bardzo niedawno poprawiono czas HK [50] przy użyciu randomizacji oraz bardzo zaawansowanych narzędzi programowania liniowego, uzyskując czas działania rzędu $\mathcal{O}(m^{1+o(1)})$

algorytm dla problemu maksymalnego przepływu. Podejście to wciąż jest dalekie od pełnego zrozumienia, pomimo że badane jest już od wielu lat. Wspomniana wcześniej hipoteza Chaudhuri *i innych* [48] mówi w szczególności, że całkowita długość wszystkich najkrótszych ścieżek powiększających skojarzenie wynosi $\mathcal{O}(n \log n)$. Prawdziwość tej hipotezy oznaczałaby, że algorytm najkrótszych ścieżek powiększających jest optymalny z dokładnością do stałej. W momencie stawiania powyższej hipotezy, nawet dla drzew, jedynym znanym górnym ograniczeniem na całkowitą długość najkrótszych ścieżek powiększających było oczywiste $\mathcal{O}(n^2)$. W [A2] z Boskiem, Leniowskim i Sankowskim, poprawiliśmy górne oszacowanie do $\mathcal{O}(n \log^2 n)$, niestety tylko dla drzew. Następnie Bernstein *i inni* [20] udowodnili górne ograniczenie $\mathcal{O}(n \log^2 n)$ dla klasy wszystkich grafów dwudzielnych. Z kolei w [A3] z Boskiem, Leniowskim i Sankowskim przedstawiliśmy dwudziestostronicowy argument dowodzący w przypadku drzew górnego ograniczenia rzędu $\mathcal{O}(n \log n)$, zamykając w ten sposób powyższy problem w tej klasie grafów. Wydaje się, że zastosowana przez nas metoda może być pomocna w rozszerzeniu ograniczenia $\mathcal{O}(n \log n)$ na klasę wszystkich grafów dwudzielnych.

Artykuł [B1] poświęcony jest problemowi dynamicznego drzewa Steinera. Problem drzewa Steinera jest jednym z najbardziej fundamentalnych problemów optymalizacji kombinatorycznej. Rozważany był z punktu widzenia wielu modeli, zaczynając od klasycznej aproksymacji [19, 126, 10, 136, 37, 43], przez modele online [94, 125, 80, 82] i stochastyczne [83, 73], aż po modele teoriogrowe [9, 23] czy re-optymalizację [22, 77, G5, G3, G6]. W obliczu takiego zainteresowania problemem, zaskakujące jest, że do momentu powstania tej pracy nie były znane dla niego żadne podliniowe algorytmy dynamiczne.

W dynamicznym problemie drzewa Steinera mamy zadaną sieć wraz ze zbiorem aktywnych użytkowników tej sieci, którzy chcą utrzymywać jak najtańsze drzewo łączące ich w trybie multicast na czas transmisji telekonferencyjnej. Użytkownicy mogą dołączać do telekonferencji oraz ją opuszczać, niemniej jednak topologia sieci pozostaje tu niezmienna. Z praktycznych względów wskazane jest, aby drzewo zmieniało się jak najmniej. Innymi słowy, mając dany ważony graf, celem jest utrzymywanie jak najtańszego drzewa łączącego dynamicznie zmieniający się zbiór aktywnych wierzchołków nazywanych terminalami. Na zbiorze terminali dostępne są operacje usunięcia pojedynczego terminala oraz dodania pojedynczego terminala.

Problem ten jako pierwsi szeroko badali Imase i Waxman [94]. W klasycznym modelu online, nawet gdy rozważamy jedynie dodawanie terminali, Imase i Waxman [94] pokazują ograniczenie dolne na współczynnik aproksymacji rzędu logarytmu z liczby terminali. Co ciekawe, z pracy Megow *i innych* [125] wynika, że stały amortyzowany budżet zmian $\mathcal{O}(\epsilon^{-1})$ na operację pozwala na $2(1+\epsilon)$-aproksymację, gdzie ϵ to dodatni parametr który można wybrać dowolnie. Co więcej, Gu *i inni* [80] pokazują, że nawet pesymistyczny budżet jednej zmiany na operację umożliwia stałą aproksymację. Niemniej jednak, wszystkie te rozważania ograniczone są do modelu inkrementalnego.

W pełni dynamicznym modelu, który jest naszym głównym przedmiotem rozważań, Imase i Waxman pokazują algorytm który dla ciągu t aktualizacji osiąga łączny budżet zmian $\mathcal{O}(t^{3/2})$ i utrzymuje 4-aproksymację optymalnego drzewa. Nasz algorytm poprawia ten wynik uzyskując amortyzowany budżet zmian rzędu $\mathcal{O}(\epsilon^{-1} \log D)$ na operację oraz utrzymując $(2+\epsilon)$-aproksymację, gdzie D to stosunek maksymalnej wagi do minimalnej, zaś ϵ to dodatni parametr który można wybrać dowolnie. Gupta *i inni* [82] w nieco innym modelu poprawiają budżet Imase i Waxmana do stałego na operacje, utrzymując w dalszym ciągu 4-aproksymację optymalnego drzewa. Ich techniki nie wydają się jednak pozwalać na uzyskanie współczynnika aproksymacji poniżej 4, gdyż pracują oni w modelu gdzie graf jest modyfikowany poprzez usuwanie i dodawanie wierzchołków. Taki model narzuca też inne dodatkowe ograniczenia: przy dodaniu wierzchołka algorytm musi przejrzeć wszystkie incydentne krawędzie, co nieodzownie generuje liniowy koszt przy implementacji takiej aktualizacji grafu.

Przyjęty przez nas model niezmiennego grafu pozwala efektywnie zaimplementować nasz algorytm z małym budżetem zmian. W szczególności w [B1] proponujemy implementację która utrzymuje $(6+\epsilon)$-aproksymację w amortyzowanym czasie na operację $\widetilde{\mathcal{O}}(\sqrt{n}\log D)$ w grafach ogólnych oraz implementację która utrzymuje $(2+\epsilon)$-aproksymację w amortyzowanym czasie na operację $\widetilde{\mathcal{O}}(\sqrt{n}\log D)$ w grafach planarnych. Są to pierwsze podliniowe algorytmy dynamiczne dla problemu drzewa Steinera, jako że wspomniane prace [94, 82] pomijają aspekt efektywnej implementacji algorytmu z małym budżetem zmian. Warto nadmienić, że implementacja jest w tym przypadku dalece nietrywialna oraz wprowadza wiele narzędzi algorytmicznych które mają wartość również w oderwaniu od problemu drzew Steinera. Przykładem są tutaj proponowane przez nas warianty struktury danych nazywanej wyrocznią odległości. Wyrocznie odległości mają szerokie zastosowanie w aplikacjach drogowych i planowaniu tras i jako takie cieszą się dużym zainteresowaniem badaczy [145, 144, 89, 49].

Artykuły [C1] i [C2] poświęcone są problemowi dynamicznego kolorowania grafu. Kolorowanie grafu jest centralnym problemem teorii grafów, cieszy się zainteresowaniem w wielu wariantach, ma mnóstwo zastosowań oraz głębokich konsekwencji w wielu dziedzinach informatyki teoretycznej. W problemie tym mamy daną sieć stacji nadawczych, z których każdej należy przyporządkować odpowiednią częstotliwość nadawania. Niektóre stacje jednak nie mogą nadawać na tej samej częstotliwości. Stacje mogą wyłączać się z nadawania albo ponownie się włączać. Celem jest dynamiczne utrzymywanie przydziału częstotliwości tak, aby każde dwa skonfliktowane nadajniki miały inne częstotliwości. Liczba dostępnych częstotliwości jest ograniczona, pożądane zaś jest ograniczenie liczby zmian częstotliwości nadającym już nadajnikom. Innymi słowy, mamy dany graf na którym utrzymywać chcemy kolorowanie zadaną liczbą kolorów, tak, aby dwa sąsiednie wierzchołki zawsze miały różne kolory. Dostępne modyfikacje grafu to dodanie bądź usunięcie wierzchołka.

W klasycznym modelu online, gdzie zawężamy się do dodawania wierzchołków, problem kolorowania badany był na szeroką skalę. W klasie grafów ogólnych które dają się pokolorować k kolorami, najlepsze co można uzyskać, nawet przy dopuszczeniu algorytmów randomizowanych, to algorytm utrzymujący $(\frac{n}{\log^2 n}\cdot k)$-kolorowanie, gdzie n to ostateczna liczba wierzchołków w grafie [87]. Nawet w klasie drzew dolne ograniczenie na liczbę kolorów dla algorytmu online jest rzędu $\Omega(\log n)$ [16]. Sytuacja jest lepsza jeśli zawęzimy się do k-kolorowalnych grafów przedziałowych: tam $3k-2$ kolorów jest niezbędne ale też wystarcza [102]. W przypadku grafów równoprzedziałowych, ograniczeniem górnym jest $2k-1$ kolorów natomiast ograniczeniem dolnym jest $3k/2$ [38, 65].

W porównaniu z ogromnym zainteresowaniem kolorowaniem online, zagadnienie kolorowania z ograniczonym budżetem zmian wydaje się być wyspą dziewiczą. Barba *i inni* [13] zbadali ten problem w ogólnej klasie grafów. Zaproponowali dwa algorytmy z ograniczonym budżetem zmian. Dla dowolnego dodatniego parametru d, pierwszy algorytm na k-kolorowalnym grafie utrzymuje kolorowanie $k(d+1)$-kolorami i osiąga budżet zmian $\mathcal{O}((d+1)n^{1/d})$ na operację, drugi zaś utrzymuje $k(d+1)n^{1/d}$-kolorowanie i osiąga budżet rzędu $\mathcal{O}(d)$. Drugi algorytm zostaje później poprawiony przez Solomona *i innych* [140]. Barba *i inni* [13] zaś pokazują, że nawet jak zawężymy rozważaną klasę grafów do lasów, to pierwszego algorytmu nie da się poprawić. Konsekwencją tego jest fakt, że jeśli zależy nam na stałej liczbie kolorów, to musimy pogodzić się z budżetem zmian wielomianowym względem wielkości grafu (przynajmniej w modelu w pełni dynamicznym). Dlatego w pracach [C1] i [C2] badamy omawiany problem w klasie grafów przedziałowych, która to klasa nie zawiera klasy lasów.

W pracy [C1] pokazujemy, że dla k-kolorowalnych grafów przedziałowych, ograniczenie $(3k-2)$ kolorów z algorytmu online da się znacznie zmniejszyć wprowadzając umiarkowany budżet zmian. Dokładniej, w modelu inkrementalnym, amortyzowany budżet zmian rzędu $\mathcal{O}(\log n)$ pozwala na utrzymywanie $2k$-kolorowania. Przy amortyzowanym budżecie $\mathcal{O}((k+1)!\sqrt{n})$ można

nawet utrzymywać optymalne k-kolorowanie. Dla grafów równoprzedziałowych prezentujemy zaś algorytm w pełni dynamiczny, który utrzymuje $(k+1)$-kolorowanie i osiąga pesymistyczny budżet zmian $\mathcal{O}(k^2)$ na operację. Oznacza to, że w tej klasie grafów dając algorytmowi jeden nadmiarowy kolor jesteśmy w stanie dostać bez żadnej amortyzacji budżet zmian ograniczony wyłącznie funkcją liczby chromatycznej.

Praca [C2] jest kontynuacją pracy [C1], która jako problem otwarty pozostawia kwestię budżetu zmian dla dokładnego kolorowania grafów równoprzedziałowych. Pokazujemy, że w modelu w pełni dynamicznym, optymalny algorytm zmuszony jest zużyć liniowy budżet na operację nawet w sensie zamortyzowanym. W modelu inkrementalnym zaś pokazujemy algorytm z amortyzowanym budżetem zmian rzędu $\mathcal{O}(k^7 \log n)$ na operację. W kontekście ograniczenia dolnego rzędu $\Omega(\log n)$ wynik ten zamyka problem z dokładnością do czynników wielomianowych względem liczby chromatycznej k. Jest to także wykładnicza poprawa względem algorytmu z pracy [C1], zarówno ze względu na n jak i ze względu na k. Wynik ten pokazuje również kolosalną różnicę w trudności pomiędzy utrzymywaniem optymalnego kolorowania i kolorowania z jednym nadmiarowym kolorem. Algorytmy z prac [C1] i [C2] dają się w większości efektywnie zaimplementować. Efektywna implementacja wynika tu na tyle bezpośrednio ze znanych narzędzi struktur danych oraz wyników dostępnych w literaturze, iż nie poświęcamy jej w tych pracach wiele uwagi.

Ostatnie dwa z omawianych artykułów [D1, D2] podążają w niedawno ugruntowanym kierunku badań nad dynamicznymi algorytmami parametryzowanymi. Algorytm dynamiczny ma tu znów za zadanie utrzymywanie informacji dotyczących rozwiązania pewnego problemu optymalizacyjnego, jednakże jego efektywność mierzona jest względem rozmiaru instancji oraz zadanego parametru (podobnie jak w przypadku statycznych algorytmów parametryzowanych). Najbardziej pożądane są tu algorytmy, które wykonują aktualizacje oraz zapytania w czasie zależnym wyłącznie od parametru nie zaś od rozmiaru instancji. W pracy [6], która otwiera ten kierunek badań, uzyskano taką dla złożoność dla wielu flagowych problemów FPT na grafach. Najbardziej znanym wśród nich jest problem pokrycia wierzchołkowego, w pozostałych zaś głównie zadaniem jest pokrywanie krawędzi grafu rozmaitymi strukturami. Dla innych problemów poruszanych w pracy [6] uzyskano algorytmy z czasem aktualizacji $\mathcal{O}(f(k)\log^{\mathcal{O}(1)} n)$, gdzie $f(k)$ jest funkcją zależną od parametru k, zaś n jest rozmiarem instancji. Taki czas aktualizacji także uznawany jest za efektywny. Uzyskano go w szczególności dla problemu k-ścieżki, który poruszamy w pracy [D1]. Rozważane w pracy [6] aktualizacje to dodanie bądź usunięcie krawędzi, co oznacza że zaproponowane tam algorytmy są w pełni dynamiczne. Praca ta rozbudziła zainteresowanie badaczy dynamicznymi algorytmami parametryzowanymi, co znajduje odzwierciedlenie w artykułach rozwijających tę tematykę [60, 117, 106, D1, D2].

W pracy [D1] główny rozważany przez nas problem to problem dynamicznego utrzymywania płytkiej dekompozycji drzewiastej. Płytka dekompozycja drzewiasta grafu jest ukorzenionym lasem na wierzchołkach tego grafu, takim że każda grafowa krawędź łączy w tym lesie wierzchołek z jego przodkiem. Głębokość drzewiasta grafu jest to parametr, który dla danego grafu określa minimalną głębokość takiego lasu. W złożoności parametryzowanej znanych jest też wiele innych dekompozycji grafowych, których prostotę określa zadany parametr. Najbardziej znana i celebrowana jest dekompozycja drzewiasta o małej szerokości i związany z nią parametr szerokość drzewiasta. W pracy [D1] skupiamy się jednak na płytkiej dekompozycji która jest koncepcyjnie prostsza i bardziej przystępna. Jawny dostęp do płytkiej dekompozycji (podobnie jak dostęp do dekompozycji o małej szerokości) w szczególności umożliwia testowanie prawdziwości formuł logicznych typu MSO_2w czasie FPT przy parametryzacji głębokością (czy też szerokością) oraz długością formuły. Jest to dość silne narzędzie, jako że formuły MSO_2są dość ogólne i potrafią wyrazić wiele grafowych problemów algorytmicznych.

Nie dziwi zatem zainteresowanie badaczy dynamicznymi dekompozycjami drzewiastymi. Z punktu widzenia testowania formuł MSO_2, jak również z punktu widzenia innych algorytmów korzystających z dekompozycji, jest tu niezwykle ważne, aby jawnie tę dekompozycję utrzymywać. Pracę nad tym zagadnieniem zapoczątkowali Dvořák *i inni* [60]. Pracują oni w modelu, gdzie graf zmienia się dynamicznie poprzez dodanie bądź usunięcie krawędzi, zbiór wierzchołków zaś pozostaje niezmienny. Celem jest utrzymywanie optymalnej płytkiej dekompozycji, ale tylko dopóki głębokość drzewiasta grafu ograniczona jest przez zadany z góry parametr d. Dvořák *i inni* zaproponowali w tym modelu algorytm dynamiczny, który aktualizuje dekompozycję w czasie $\mathcal{O}(f(d))$ dla pewnej nieelementarnej funkcji f. Algorytm ten opiera się na obserwacji, że po każdej aktualizacji wystarczy zmodyfikować fragment dekompozycji wielkości $\mathcal{O}(f(d))$. Fragment ten nazywamy rdzeniem dekompozycji. Jest to zatem klasyczny przykład algorytmu z ograniczonym (pesymistycznym) budżetem zmian, który potem udaje się efektywnie zaimplementować.

W pracy [D1] poprawiamy ograniczenie na wielkość rdzenia do $d^{\mathcal{O}(d)}$, co daje znacznie lepsze ograniczenie na budżet zmian. Naszym wkładem jest również lepsze zrozumienie kombinatoryki problemu, co owocuje bardziej zwartym algorytmem dynamicznym oraz przejrzystą jego implementacją. Nasza implementacja daje pesymistyczny czas na aktualizację rzędu $2^{\mathcal{O}(d^2)}$. Ponadto, pokazujemy jak nasz algorytm połączyć ze znaną techniką kolejkowania [64]. Technika ta umożliwia nam utrzymywanie dynamicznego grafu o dowolnej głębokości drzewiastej oraz odpowiadania na zapytania czy głębokość jest poniżej parametru d. Jeśli odpowiedź jest twierdząca, nasz algorytm udostępnia optymalną dekompozycję aktualnego grafu. To z kolei stanowi podwaliny dla proponowanych przez nas dalej w pełni dynamicznych algorytmów dla problemu k-ścieżki oraz k-cyklu. W pierwszym z tych problemów pytamy, czy w grafie istnieje ścieżka długości k, w drugim zaś czy istnieje cykl o długości przynajmniej k. Nasz algorytm dla k-ścieżki aktualizuje graf w amortyzowanym czasie $2^{\mathcal{O}(k^2)}$, co daje jakościową poprawę w stosunku do pracy Almana *i innych*. Nasz algorytm dla k-cyklu aktualizuje graf w amortyzowanym czasie $2^{\mathcal{O}(k^4)} + \mathcal{O}(k \log n)$.[8]

W pracy [D2] rozważamy dynamiczne problemy FPT dla słów. We wszystkich rozważanych tu problemach na wejściu dana jest lista słów. Aktualizacją instancji jest zmiana pojedynczej litery w pojedynczym słowie. W tak zdefiniowanym modelu dynamicznym jako pierwszy rozważamy problem *najbliższego słowa*. W dziedzinie złożoności parametryzowanej jest to znany i szeroko przebadany problem [78, 115, 114, 55]. W problemie tym dana jest lista słów równej długości nad zadanym alfabetem Σ oraz parametr d. Pytamy czy istnieje słowo nad alfabetem Σ, dla którego odległość Hamminga do każdego ze słów na wejściu jest ograniczona przez parametr d. W przypadku tak postawionego dynamicznego problemu najbliższego słowa, w odróżnieniu od problemów wcześniej rozważanych, pesymistyczny budżet zmian w rozwiązaniu, o ile ono istnieje, można łatwo ograniczyć przez $\mathcal{O}(d)$ na aktualizację. Taki budżet zmian wynika bowiem wprost ze statycznego algorytmu FPT dla tego problemu [78]. Dostajemy zatem problem typowo strukturodanowy. Poszukujemy struktury danych, która efektywnie (najlepiej w czasie zależnym wyłącznie od $|\Sigma|$ oraz d) przeliczy rozwiązanie po każdej aktualizacji. Dla problemu najbliższego słowa mamy do zaoferowania dwie randomizowane struktury danych, które aktualizują instancję oraz przeliczają rozwiązanie w pesymistycznym czasie odpowiednio $d^{\mathcal{O}(d)}$ oraz $|\Sigma|^{\mathcal{O}(d)}$. Według naszej wiedzy są to pierwsze dynamiczne algorytmy dla problemu najbliższego słowa. Dzięki ogólnym technikom z dziedziny złożoności parametryzowanej, obie te struktury po aktualizacji wyznaczają rozwiązanie niejako od zera, zatem pojęcie budżetu zmian traci tutaj nieco rację bytu. Pokazuje to, że pytanie o budżet zmian nie dla każdego problemu dynamicznego jest ciekawe.

W poszukiwaniu bardziej ogólnych wyników rozważyliśmy w modelu dynamicznym dwa kolejne problemy parametryzowane na słowach, mianowicie problem *rozłącznych podsłów* (ang. *disjoint factors*) oraz problem *odległości edycyjnej*. W problemie rozłącznych podsłów, na wejściu

[8] Oba te wyniki podane są przy założeniu, że słownik przchowujący krawędzie grafu umożliwia operacje dostępu i aktualizacji w czasie stałym. Alman *i inni* [6] również podają swoje wyniki bazując na tym założeniu.

dany jest parametr k oraz słowo w nad alfabetem $\Sigma = \{1, \ldots, k\}$. Pytamy o to, czy istnieje k rozłącznych podsłów słowa w takich, że dla każdej z k liter alfabetu jedno z podsłów zaczyna się i kończy tą literą. W problemie odległości edycyjnej na wejściu dostajemy parametr k oraz dwa słowa u, w (nad alfabetem Σ) o łącznej długości n, i chcemy zdecydować czy odległość edycyjna pomiędzy u i w nie przekracza k. Innymi słowy pytamy czy można przekształcić słowo u w słowo w za pomocą co najwyżej k edycji w postaci usunięcia litery, dodania litery, lub zmiany litery na inną. Problem rozłącznych podsłów był badany w dziedzinie złożoności parametryzowanej [30], a wybraliśmy go do naszych rozważań ze względu na jego prostą strukturę kombinatoryczną, w nadziei że ogólne techniki będą w jego przypadku wyraźnie widoczne. Z kolei problem odległości edycyjnej jest ważnym problemem z wieloma zastosowaniami (opisanymi na przykład w [128]). Co ciekawe, oba te problemy mają efektywny algorytm dynamiczny wynikający z dostępnych w literaturze narzędzi. W pracy [D2] sformułowaliśmy meta-twierdzenie, które daje dynamiczny algorytm z czasem aktualizacji $\mathcal{O}(f(k) \log \log n)$ dla szerokiej gamy dynamicznych problemów na słowach, jednak wynika ono dość bezpośrednio z narzędzi logiki [70, 124, 138]. W szczególności da się ono zastosować do wymienionych wyżej problemów. Dla obu tych problemów pokazujemy też ograniczenia dolne, które implikują że pozbycie się czynnika $\log \log n$ z czasu aktualizacji wydaje się mało prawdopodobne. Dodatkowo, pokazujemy dla każdego z tych problemów dobraną specjalnie dla niego strukturę danych, która znacząco poprawia czynnik $f(k)$ wynikający z ogólnego twierdzenia.

A. Dynamiczne skojarzenia w grafach dwudzielnych

A.1. **Problem skojarzenia w grafach dwudzielnych z ograniczonym budżetem zmian.** Problem skojarzeń w grafach dwudzielnych ma długą historię. W 1973 roku Hopcroft i Karp zaprezentowali elegancki algorytm, który oblicza maksymalne skojarzenie w czasie $\mathcal{O}(m\sqrt{n})$. Algorytm ten okazał się kamieniem milowym, który trudno było przeskoczyć. Został on poprawiony dla grafów odpowiednio gęstych [127] oraz odpowiednio rzadkich [116], niemniej jednak w przypadku ogólnym problem skojarzeń dwudzielnych bardzo długo czekał na przełom. W niedawnej pracy [50] problem ten doczekał się algorytmu działającego w czasie $\mathcal{O}(m^{1+o(1)})$, co stanowi milowy postęp w algorytmice. Algorytm ten jest jednak bardzo skomplikowany, korzysta z randomizacji oraz zaawansowanych narzędzi programowania liniowego, a także rozwiązuje dużo bardziej złożony problem przepływu. Co ciekawe, powszechnie znana jest też bardzo prosta heurystyka *turbo matching*, która w praktyce jest znacznie szybsza niż algorytm Hopcrofta i Karpa. Po dziś dzień nie jest znana instancja problemu, dla której algorytm Hopcrofta i Karpa byłby lepszy niż *turbo matching*. Wydaje się zatem że dzieli nas jeszcze daleka droga od pełnego zrozumienia problemu skojarzeń w grafach dwudzielnych.

Nasze rozumienie tego problemu pogłębia bogata literatura, która oferuje szeroki wachlarz algorytmów dynamicznych w modelu, gdzie graf aktualizowany jest poprzez dodawanie bądź usuwanie krawędzi. Wariant ten jest szeroko zbadany zarówno w modelu dokładnym [137, 3, 105, 88] jak i aproksymacyjnym [139, 21, 84]. Znacznie mniej wyników literatura oferuje dla aktualizacji w postaci dodawania lub usuwania wierzchołków [79, 48, A1, 20], zaś w modelu online głównym kierunkiem badań jest wspomniany we wstępie algorytm rankingowy [98, 24, 56]. W niniejszym opracowaniu skupimy się na aktualizacjach grafu w postaci dodawania wierzchołków wraz z incydentnymi krawędziami, które odpowiadają klasycznemu modelowi online.

Najbardziej ogólnym scenariuszem inkrementalnym zarówno w modelu online jak i z ograniczonym budżetem zmian jest scenariusz następujący, zwany *dwustronnym*. Algorytm inkrementalny otrzymuje na wejściu graf dwudzielny $G = \langle S, C, E \rangle$ w sposób online, tzn. w *kolejności prezentacji* $v_1 \ll v_2 \ll \ldots \ll v_{2n}$, gdzie $\{v_1, v_2, \ldots, v_{2n}\} = S \cup C$. Po zaprezentowaniu początkowego

fragmentu zbioru wierzchołków $V_t = \{v_1, v_2, \ldots, v_t\}$, algorytm inkrementalny konstruuje skojarzenie M_t w grafie $G[V_t]$,[9] bazując wyłącznie na $G[V_t]$, $\ll$ zawężonej do V_t, oraz $M_1, \ldots, M_{t-1}$. Celem algorytmu z ograniczonym budżetem zmian jest zminimalizowanie liczby zmian w każdej turze t, tzn zminimalizowanie $|M_t \oplus M_{t-1}|$. Można tu rozważać model pesymistyczny (ang. worst case), gdzie minimalizujemy wartość $\mathtt{B_{wc}}$, która dla każdej tury t spełnia $|M_t \oplus M_{t-1}| \leqslant \mathtt{B_{wc}}$. Algorytm online jest szczególnym przypadkiem algorytmu z pesymistycznym budżetem zmian $\mathtt{B_{wc}} = 1$. My jednak skupimy się w tym problemie na ograniczeniu łącznego budżetu zmian $\mathtt{B_{tot}}$, który spełnia $|M_1| + |M_1 \oplus M_2| + \ldots + |M_{2n-1} \oplus M_{2n}| \leqslant \mathtt{B_{tot}}$.

Zawężenie tego scenariusza w taki sposób, że na starcie zaprezentowane są już wszystkie wierzchołki ze zbioru S, zaś wierzchołki C prezentowane są w sposób online, w kolejności prezentacji $c_1 \ll \ldots \ll c_n$, nazywamy modelem *jednostronnym*. Liczba tur wynosi wtedy n, w każdej turze $t \in \{1, \ldots, n\}$ prezentowany jest klient c_t i wyliczane jest skojarzenie M_t. Budżet zmian definiowany jest wtedy analogicznie na bazie n skojarzeń $M_1 \ldots M_n$.

Scenariusz dwustronny okazuje się być dużo trudniejszym niż scenariusz jednostronny. W klasycznym modelu online w obu scenariuszach deterministycznie najlepsze co można osiągnąć to współczynnik aproksymacji $1/2$. Sytuacja ta zmienia się jednak jeśli pozwolimy na algorytm randomizowany: dla modelu jednostronnego Karp *i inni* pokazali randomizowany algorytm $(1 - 1/e)$-aproksymacyjny [98], podczas gdy dla modelu dwustronnego wiadomo, że nie można uzyskać współczynnika ponad 0.5914 nawet przy użyciu randomizacji [42]. Ponadto od lat trwają badania, aby, używając randomizacji, uzyskać w modelu dwustronnym współczynnik lepszy niż $1/2$ [148, 42, 51, 72], co udało się dopiero w 2019 roku z nieznaczną tylko poprawą w stosunku do $1/2$.

Gdy zamiast aproksymacji pozwolimy deterministycznemu algorytmowi dokonywać zmian w utrzymywanym rozwiązaniu, wówczas sytuacja w modelu dwustronnym również będzie niekorzystna. Łatwo zauważyć, że naiwny algorytm z budżetem zmian $\mathtt{B_{tot}} \in \mathcal{O}(n^2)$ jest asymptotycznie najlepszy. Dla przykładu, ścieżka rozszerzona na przemian z każdej strony, wymusza na każdym algorytmie inkrementalnym łącznie $\Omega(n^2)$ zmian. Ograniczenie to działa również jeśli będziemy budować graf dodając krawędzie nie zaś wierzchołki. Obserwacja ta powoduje, że ograniczony budżet zmian dla problemu optymalnego skojarzenia w grafach dwudzielnych bada się w scenariuszu jednostronnym, i tylko taki będziemy rozważać w dalszej części niniejszego opracowania. Pytaniem fundamentalnym jest tutaj jaki budżet zmian jest na to potrzebny. Pytanie to zostało postawione w 1995 przez Grove'a *i innych* [79]. W tej samej pracy autorzy pokazali górne ograniczenie w postaci $\mathtt{B_{tot}} \in \mathcal{O}(n \log n)$ w przypadku, gdy każdy klient łączy się z co najwyżej dwoma serwerami. W przypadku ogólnym do momentu pojawienia się pracy [A1] znane było jedynie trywialne ograniczenie górne $\mathtt{B_{tot}} \in \mathcal{O}(n^2)$, pomimo że Chaudhuri *i inni* postawili hipotezę że właściwą odpowiedzią jest $\mathtt{B_{tot}} \in \mathcal{O}(n \log n)$ [48].

W [A1] z Boskiem, Leniowskim i Sankowskim przedstawiliśmy następujący wynik, który stanowi krok w kierunku rozwiązania tej hipotezy.

Twierdzenie 1 (Wniosek V.9 z [A1] połączony z Algorytmem 1 z [A1])**.** *Istnieje algorytm z łącznym budżetem zmian* $\mathtt{B_{tot}} \in \mathcal{O}(n\sqrt{n})$*, konstruujący maksymalne skojarzenie w grafach dwudzielnych, który można zaimplementować w łącznym czasie* $\mathcal{O}(m\sqrt{n})$*.*

Kluczem do uzyskania powyższego wyniku jest algorytm, którego istotą jest generowanie jak najmniejszego budżetu zmian. Głównym pomysłem jest tutaj rozłożenie dotychczas wykorzystanego budżetu zmian pomiędzy wierzchołki z S zwane serwerami. W tym celu dla każdego serwera $s \in S$ algorytm utrzymuje atrybut $\mathrm{rank}_t(s)$ nazywany rangą serwera, który określa ile razy dotychczas serwer s zmieniał skojarzenie: indeks t oznacza czas w którym zdefiniowany jest atrybut. Mówiąc inaczej, ranga $\mathrm{rank}_t(s)$ określa ile razy serwer s zmieniał skojarzenie do

[9] Przez $G[X] = \langle U \cap X, W \cap X, E[X] \rangle$ oznaczamy podgraf G indukowany przez $X \subseteq U \cup W$, gdzie $E[X] = \{e \in E : e \subseteq X\}$.

momentu pojawienia się t-tego klienta $c_t = v_{n+t} \in C$. W momencie pojawienia się klienta c_t, algorytm znajduje ścieżkę π_t powiększającą skojarzenie M_{t-1}, atrybuty zaś prowadzą algorytm przy znajdowaniu tej ścieżki. Pomysł polega na tym, aby ścieżka π_t używała wierzchołków o jak najniższych rangach, tak aby podzielić dotychczasowy budżet zmian równomiernie pomiędzy serwery. Nasuwa się tu pomysł aby zdefiniować rangę ścieżki jako maksymalną rangę położonego na niej serwera $\mathrm{rank}_t(\pi) = \max_{s \in \pi}\{\mathrm{rank}_t(s)\}$, i wybrać ścieżkę π_t jako ścieżkę powiększającą π o minimalnej randze. Taki wybór ścieżki ma tę wadę, że jeśli ścieżka π_t musi odwiedzić pewną wysoką rangę r, to algorytm może dowolnie zwiększać wszystkie rangi nie większe niż r. Wymagamy więc dodatkowo, aby każdy sufiks ścieżki π_t miał minimalną rangę. Pomocne w kontrolowaniu rang sufiksów ścieżki powiększającej są dodatkowe atrybuty $\mathrm{tier}_t(c)$ przypisane do klientów $c \in C$. Atrybut $\mathrm{tier}_t(c)$ dla $c \in C$, nazywany też poziomem klienta c w czasie t, zdefiniowany jest jako minimalna ranga ścieżki alternującej łączącej klienta c z serwerem nieskojarzonym w chwili t. Algorytm szuka poziomowanej ścieżki π_t, czyli takiej, wzdłuż której poziomy klientów nie wzrastają. To gwarantuje że każdy sufiks ma minimalną rangę.

Dla tak zdefiniowanego algorytmu jesteśmy w stanie udowodnić, że maksymalna ranga serwera nie przekroczy $\mathcal{O}(\sqrt{n})$, co dowodzi również że łączny budżet zmian wynosi $\mathcal{O}(n\sqrt{n})$.

Dodatkowo jesteśmy w stanie znaleźć i zaaplikować wszystkie poziomowane ścieżki powiększające w łącznym czasie $\mathcal{O}(m\sqrt{n})$. Kluczem do efektywnej implementacji jest leniwe utrzymywanie atrybutów $\mathrm{tier}_t(c)$, czyli poziomów klientów $c \in C$. Jak pokazujemy w pracy [A1], poziom klienta $\mathrm{tier}_t(c)$ nie maleje z czasem, może się tylko zwiększać, a jego finalna wartość jest rzędu $\mathcal{O}(\sqrt{n})$. Algorytm dla każdego klienta $c \in C$ utrzymuje atrybut, który ogranicza z dołu wartość $\mathrm{tier}_t(c)$ (algorytm nie zawsze zna prawdziwą wartość $\mathrm{tier}_t(c)$). W każdym momencie t w poszukiwaniu poziomowanej ścieżki π_t algorytm przeszukuje graf rekurencyjnie w głąb kierując się najpierw w stronę wierzchołków o jak najmniejszym atrybucie. W momencie gdy algorytm nie znajduje ścieżki i musi wycofać się z rekurencji, podbijane są atrybuty wierzchołkom, które zostały rekurencyjnie odwiedzone. Atrybuty te można podbić, gdyż brak ścieżki oznacza że atrybuty są istotnie niższe niż prawdziwe poziomy wierzchołków. Podbijanie atrybutów klientom umożliwia amortyzację procesu przeszukiwania grafu. Warto zwrócić tu uwagę na prostotę samego algorytmu, który w każdej turze sprowadza się do przeszukania grafu rekurencyjnie w głąb przy jednoczesnym podbijaniu atrybutów wycofując się z rekurencji.

Algorytm nasz stanowi ciekawą alternatywę dla znanego od lat algorytmu offline Hopcrofta i Karpa, który również znajduje maksymalne skojarzenie w grafie dwudzielnym w łącznym czasie $\mathcal{O}(m\sqrt{n})$. Algorytm nasz ma tę przewagę, że robi nawet więcej, ponieważ znajduje n maksymalnych skojarzeń, podczas gdy algorytm Hopcrofta-Karpa znajduje tylko jedno. Na uwagę zasługuje również, że w przeciwieństwie do algorytmu Hopcrofta i Karpa, algorytm nasz znajduje ścieżki powiększające nie kierując się globalną wiedzą na temat całego grafu, a posługując się tylko i wyłącznie informacją lokalną przypisaną do wierzchołków. Jest to zupełnie nowe podejście do problemu znajdowania skojarzeń w grafach dwudzielnych.

Techniki, które zaproponowaliśmy w pracy [A1] do uzyskania głównego wyniku okazują się również pomocne, gdy zrezygnujemy z wymagania, aby utrzymywane skojarzenie było optymalne. Wtedy nawet liniowy względem n łączny budżet zmian (czyli średnio stały na dodanie wierzchołka) jest wystarczający, o czym mówi poniższe twierdzenie.

Twierdzenie 2 (Lemat VI.2. w [A1])**.** *Dla każdego $\varepsilon > 0$ istnieje $(1-\varepsilon)$-aproksymacyjny algorytm inkrementalny dla problemu maksymalnego skojarzenia w grafach dwudzielnych, który dokonuje sumarycznie $\mathcal{O}(\varepsilon^{-1}n)$ zmian w łącznym czasie $\mathcal{O}(\varepsilon^{-1}m)$.*

Dla porównania, $(1 - 1/e)$-aproksymacyjny algorytm [98] zastosowany do tego modelu daje całkowitą liczbę $\mathcal{O}(n)$ zmian i całkowitą złożoność czasową $\mathcal{O}(m)$, i przed Twierdzeniem 2 nie

było znane nic lepszego. Co więcej, rozwiązanie z Twierdzenia 2 daje ten sam czas działania co najlepsze znane rozwiązanie offline.[10]

Ponadto pokazujemy że nasze techniki dają to samo ograniczenie w przypadku dekrementalnym. W tym modelu algorytm na starcie otrzymuje graf dwudzielny $\langle S, C, E \rangle$. Celem algorytmu jest utrzymywanie maksymalnego skojarzenia podczas, gdy wierzchołki ze zbioru C sukcesywnie są usuwane.

Wniosek 3 (Wniosek VI.4. w [A1])**.** *Istnieje algorytm dekrementalny utrzymujący maksymalne skojarzenie w grafach dwudzielnych dokonując przy tym sumarycznie $\mathcal{O}(n\sqrt{n})$ zmian w łącznym czasie $\mathcal{O}(m\sqrt{n})$.*

Wniosek 4 (Wniosek VI.6. w [A1])**.** *Dla każdego $\varepsilon > 0$ istnieje $(1-\varepsilon)$-aproksymacyjny algorytm dekrementalny dla problemu maksymalnego skojarzenia w grafach dwudzielnych, który dokonuje sumarycznie $\mathcal{O}(\varepsilon^{-1}n)$ zmian w łącznym czasie $\mathcal{O}(\varepsilon^{-1}m)$.*

Na koniec rozszerzamy nasz algorytm na grafy z całkowitoliczbowymi wagami na krawędziach ograniczonymi przez W, uzyskując następujące wyniki.

Wniosek 5 (Wniosek VI.8. w [A1])**.** *Istnieją algorytmy inkrementalny i dekrementalny utrzymujące sumaryczną wagę maksymalnego skojarzenia w całkowitym czasie $\mathcal{O}(W^{3/2}m\sqrt{n})$.*

Wniosek 6 (Wniosek VI.9. w [A1])**.** *Dla każdego $\varepsilon > 0$ istnieją $(1-\varepsilon)$-aproksymacyjne algorytmy inkrementalny i dekrementalny dla problemu maksymalnego ważonego skojarzenia działające w całkowitym czasie $\mathcal{O}(W\varepsilon^{-1}m)$.*

A.2. **Budżet zmian dla algorytmu najkrótszych ścieżek.** Algorytm najkrótszych ścieżek powiększających, nazywany w literaturze algorytmem Sap (ang. Shortest Augmenting Path), jest naturalnym alternatywnym podejściem do problemu dwudzielnych skojarzeń z ograniczonym budżetem zmian. Rozważamy go w modelu jednostronnym wprowadzonym wcześniej. Algorytm ten zawsze gdy pojawi się kolejny klient powiększa skojarzenie po najkrótszej z dostępnych ścieżek powiększających. Intuicyjnie podejście to wydaje się bardzo sensowne: w każdej turze algorytm zachłannie aplikuje najmniejszą możliwą liczbę zmian. Pomysł ten nie jest nowy. Już w 1972 roku posłużyli się nim Edmonds i Karp, aby uzyskać pierwszy silnie wielomianowy algorytm dla maksymalnego przepływu [62]. Chaudhuri *i inni* [48] wysnuli hipotezę, że łączny budżet zmian algorytmu Sap jest rzędu $\mathcal{O}(n \log n)$. Prawdziwość tej hipotezy niosłaby ciekawe konsekwencje nawet dla problemu maksymalnego przepływu. Pokazałaby ona, że łączna długość ścieżek powiększających w algorytmie Edmondsa-Karpa dla sieci jednoprzepustowych[11] o m krawędziach i n wierzchołkach wynosi $\mathcal{O}(m \log n)$. Związek taki można otrzymać konstruując dla danej sieci przepływowej dwudzielny graf krawędziowy: konstrukcja ta jest znaną techniką pozwalającą zredukować problem przepływu do problemu maksymalnego skojarzenia [97]. Otrzymany w ten sposób graf dwudzielny posiada $2m$ wierzchołków.

Ze względu na jego prostotę i szerokie zastosowania, algorytm Sap przykuł uwagę wielu badaczy. Niestety, pomimo wysiłków, przez długi czas nie udawało się dobrze zrozumieć tej fundamentalnej techniki. Grove *i inni* [79] badali szczególny przypadek, gdy każdy klient może połączyć się z co najwyżej dwoma serwerami. Przy takim założeniu udowodnili prawdziwość hipotezy Chaudhuri *i innych*. Udowodnili też ograniczenie dolne rzędu $\Omega(n \log n)$ na łączną długość ścieżek w algorytmie Sap . Prawdziwość swojej hipotezy pokazali również Chaudhuri *i inni*, ale pod warunkiem że klienci pojawiają się w losowym porządku: udowodnili, że wtedy oczekiwana łączna długość ścieżek powiększających w algorytmie Sap jest rzędu $\mathcal{O}(n \log n)$.

[10] Wykonanie $\mathcal{O}(\varepsilon^{-1})$ faz algorytmu Hopcrofta-Karpa daje $(1-\varepsilon)$-aproksymacyjny algorytm dla problemu maksymalnego skojarzenia, który działa w czasie $\mathcal{O}(\varepsilon^{-1}m)$ (patrz [91]).

[11] W sieci jednoprzepustowej wszystkie przepustowości równe są 0 bądź 1.

Jednak w dużo ciekawszym modelu, gdzie porządek klientów jest dowolny, nawet dla drzew, do momentu pojawienia się pracy [A2], najlepszym znanym górnym oszacowaniem na budżet zmian algorytmu Sap było oczywiste $\mathcal{O}(n^2)$.

Kłopoty z analizą algorytmu Sap wynikają z faktu, że najkrótsze ścieżki powiększające nie mają wyraźnej struktury, która pozwoliłaby na rygorystyczne dowody. Łatwiejsze w analizie są inne metody wyboru ścieżki powiększającej, jak metoda przedstawiona w poprzednim podrozdziale, czy też algorytm Chaudhuriego *i innych* dla drzew, w którym wybór ścieżki jest mocno związany ze strukturą drzewa. Niestety, w przypadku najkrótszych ścieżek, ciężko jest uchwycić jakąkolwiek strukturę skojarzenia M_{t-1} która pomoże ograniczyć długość ścieżki w chwili t. Dlatego też prace [79, 48, 20, A2, A3] zajmujące się analizą algorytmu Sap odchodzą od analizy skojarzenia generowanego w każdej turze przez algorytm, a w zamian przyjmują, że w każdej turze algorytm zastaje najgorsze możliwe skojarzenie. Prowadzi to do nieco bardziej ogólnego modelu, który ostatecznie pozwala znacznie uprościć analizę. W modelu tym, tak jak poprzednio, graf $G = \langle S, C, E \rangle$ prezentowany jest po jednym kliencie na turę, gdzie klienci prezentowani są w porządku $c_1 \ll c_2 \ll \ldots \ll c_n$ wraz z incydentnymi krawędziami. Różnica polega na tym, że skojarzenie M_{t-1} które utworzył algorytm w turze $t-1$ dla grafu $G[V_{t-1}]$, podmieniane jest na dowolne skojarzenie M'_{t-1}. W turze t algorytm operuje na grafie $G[V_t]$ oraz na skojarzeniu M'_{t-1}, a jego zadaniem jest rozszerzenie skojarzenia M'_{t-1} o klienta c_t. Budżet zmian w tym modelu definiujemy analogicznie jak poprzednio, ale liczba zmian liczona jest względem skojarzenia M'_{t-1}. Uściślając, łączny budżet zmian jest wartością $\mathtt{B_{tot}}$ która spełnia $|M_1| + |M'_1 \oplus M_2| + \ldots + |M'_{n-1} \oplus M_n| \leqslant \mathtt{B_{tot}}$. Wydawać by się mogło iż model ten jest zbyt pesymistyczny w porównaniu do oryginalnego modelu jednostronnego. Zaletą jest jednak to, że pesymistyczne skojarzenie M'_t nie zależy od wcześniejszych kroków algorytmu, a jedynie od struktury grafu $G[V_t]$, co znacznie upraszcza analizę. Co więcej, wszystkie dotychczasowe wyniki ograniczające łączną długość ścieżek powiększających w algorytmie Sap uzyskane są w tym modelu. Model ten nazywać będziemy modelem z *niepomyślnym skojarzeniem*. Pomimo, że pozytywne wyniki uzyskane w pracach wcześniejszych [79, 48] niejawnie uzyskane są w modelu z niepomyślnym skojarzeniem, model ten został po raz pierwszy zauważony i jawnie wprowadzony w pracy [A2]. W tej samej pracy udało nam się zrobić pierwszy nietrywialny postęp w rozwiązaniu hipotezy Chaudhuri *i innych* dla przypadku drzew.

Twierdzenie 7 (Twierdzenie 1 w [A2])**.** *Całkowita długość ścieżek powiększających w algorytmie Sap na drzewach jest rzędu* $\mathcal{O}(n \log^2 n)$.

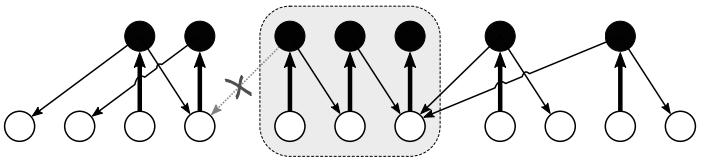
Wynik ten został potem przykryty wynikiem Bersteina *i innych* [20], którzy pokazali że całkowita długość ścieżek w algorytmie Sap jest rzędu $\mathcal{O}(n \log^2 n)$ w ogólnej klasie grafów dwudzielnych. Z kolei w przypadku drzew udało się nam całkowicie rozwiązać problem kilka lat później w pracy [A3].

Twierdzenie 8 (Twierdzenie 1 w [A3])**.** *Całkowita długość ścieżek powiększających w algorytmie Sap na drzewach jest rzędu* $\mathcal{O}(n \log n)$.

Oba te wyniki bazują na wspólnych podwalinach, które przedstawimy najpierw. W szczególności oba wyniki uogólniają prosty lemat (Lemat 9), który rozwiązuje problem długości najkrótszych ścieżek powiększających, gdy każdy klient ma stopień przynajmniej dwa. Kolosalna różnica pojawia się dopiero w uogólnianiu lematu, gdyż praca [A3] bazuje w tym celu na diametralnie innym pomyśle niż praca [A2]. Uogólnienie tego lematu do przypadku gdy klienci mogą mieć stopień jeden jest w obu przypadkach nietrywialne i w obu przypadkach jest głównym wynikiem pracy.

Pierwszą dość istotną wspólną koncepcją przy analizie algorytmu Sap jest wyróżnienie wierzchołków *żywych* i *martwych*. Rozróżnienie to pojawia się już u Grove'a *i innych*. Intuicyjnie, *martwe* wierzchołki to takie, których nie może już nigdy więcej odwiedzić ścieżka powiększająca skojarzenie, ponieważ nie da się z nich ścieżką alternującą osiągnąć wolnego (nieskojarzonego)

serwera. Pozostałe wierzchołki są *żywe*. Martwota wierzchołków jest własnością monotoniczną, tzn. żywy wierzchołek może w turze t zmienić się w wierzchołek martwy, ale wtedy pozostaje martwy we wszystkich turach $t' > t$. Ciekawą obserwacją jest że status (martwota) wierzchołka w turze t jest taki sam dla każdego maksymalnego skojarzenia M'_t. Martwe wierzchołki można alternatywnie zdefiniować jako wierzchołki zbioru $C' \cup N(C')$ [12], gdzie zachodzi $|C'| \geqslant |N(C')|$ oraz $C' \subseteq C$ (jak na Rysunku 1).



RYSUNEK 1. Podział wierzchołków na żywe i martwe. Krawędzie skojarzenia M_t zorientowane są od serwerów (białe wierzchołki) do klientów (czarne wierzchołki), podczas gdy pozostałe krawędzie zorientowane są w kierunku odwrotnym. Martwe wierzchołki zaznaczone są szarym regionem. Skreślona wykropkowana krawędź nie jest częścią grafu, a jedynie obrazuje jakie krawędzie nie mogą wychodzić poza martwy region.

Punktem wyjściowym zarówno w pracy [A2] jak i w pracy [A3] jest następujący prosty lemat.

Lemat 9 (Lemat 1 w [A2], Lemat 12 w [A3])**.** *Niech $C_t = \{c_1, \ldots, c_t\}$ będzie prefiksem klientów zaprezentowanych w turach $\{1, \ldots, t\}$. Jeśli G jest drzewem oraz każdy klient z C_t ma przynajmniej dwóch sąsiadów, to w turze t wszystkie wierzchołki w $G[V_t]$ są żywe, a łączna długość ścieżek powiększających aplikowanych przez* SAP *do tury t jest rzędu $\mathcal{O}(n \log n)$.*

Dowód powyższego lematu zasadza się na tym, że każdy klient $c_{t'} \in C_t$ łączy przynajmniej dwa drzewa z grafu $G[V_{t'-1}]$. W związku z tym, że wszystkie wierzchołki są żywe, $c_{t'}$ ma do wyboru przynajmniej dwie alternatywne ścieżki powiększające, z których krótsza ma długość ograniczoną rozmiarem mniejszego drzewa. Stąd długość najkrótszej ścieżki opłacić mogą wierzchołki mniejszego drzewa, jeśli każdy zapłaci 1. Łatwo zauważyć, że każdy wierzchołek zapłaci łącznie co najwyżej $\log n$, gdyż tyle razy może być wierzchołkiem drzewa które podwaja swój rozmiar.

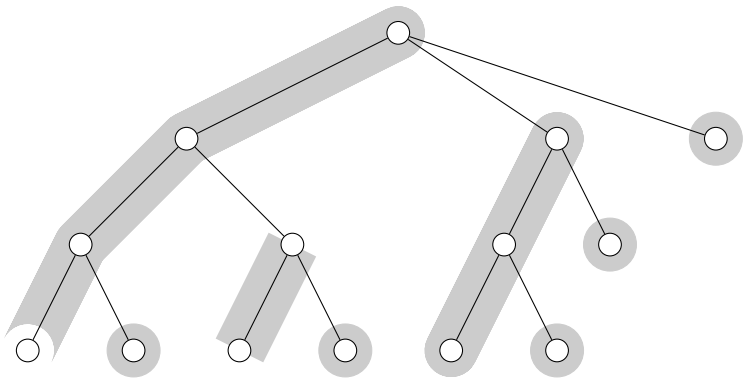
Ten prosty lemat pokazuje, że całą trudność w dowodzie Twierdzenia 7 stanowią klienci o stopniu 1. Załóżmy, że w pewnej turze t zaprezentowany zostaje klient c_t o stopniu 1 i niech π_t będzie ścieżką alternującą użytą przez algorytm SAP aby skojarzyć c_t. W obu pracach [A2] i [A3] istotnym elementem jest wyróżnienie pierwszego wierzchołka na ścieżce π_t z którego odchodzą dwie alternatywne ścieżki alternujące. Taki wierzchołek (jeśli istnieje) jest zawsze klientem, jest unikalny dla każdej ścieżki π_t i nazywamy go *wierzchołkiem rozdzielającym* dla ścieżki π_t (jeśli nie istnieje to przyjmujemy że wierzchołkiem rozdzielającym jest ostatni wierzchołek na ścieżce (czyli serwer)). Można zauważyć, że wierzchołki na ścieżce π_t do wierzchołka rozdzielającego umierają w turze t, zatem wystarczy oszacować długość sufiksu π_t począwszy od wierzchołka rozdzielającego. Oznaczmy wierzchołek rozdzielający dla ścieżki π_t jako $\mathrm{disp}(\pi_t)$, zaś sufiks ścieżki π_t począwszy od $\mathrm{disp}(\pi_t)$ do wolnego serwera oznaczmy jako μ_t. Na podstawie powyższych rozważań wystarczy oszacować sumę długości sufiksów $\sum_{t=1}^{n} |\mu_t|$.

W tym momencie następuje rozłam koncepcyjny pomiędzy pracami [A2] i [A3]. W pierwszej kolejności opiszemy pomysły na to szacowanie zaproponowane w pracy [A2].

Ważnym krokiem w tej pracy jest podział wszystkich sufiksów μ_t, $t \in \{1, \ldots, n\}$, na sufiksy *ostateczne* i *nieostateczne*. Patrząc z punktu widzenia konkretnego klienta c_t który pojawia się w

[12] $N(X)$ oznacza sąsiedztwo zbioru X

turze t, może on być wierzchołkiem rozdzielającym dla wielu ścieżek $\pi_{t'}$, takich że $t' \geqslant t$. Niech $\mathrm{tms}(c_t)$ będzie zbiorem numerów tur $t' \in \{t, \ldots, n\}$ takich że $c_t = \mathrm{disp}(\pi_{t'})$ i niech $\mathrm{last}(c_t) = \max(\mathrm{tms}(c_t))$. Sufiksy $\mu_{t'}$ takie że $t' \in \mathrm{tms}(c_t)$ oraz $t' < \mathrm{last}(c_t)$ nazwiemy *nieostatecznymi*, zaś sufiks $\mu_{t'}$ taki że $t' \in \mathrm{tms}(c_t)$ oraz $t' = \mathrm{last}(c_t)$ nazwiemy *ostatecznym*. Praca [A2] najpierw szacuje łączną długość sufiksów nieostatecznych, a potem, nieco inaczej, szacuje łączną długość sufiksów ostatecznych. Co ciekawe, łączna długość sufiksów nieostatecznych jest rzędu $\mathcal{O}(n \log n)$. Znacznie trudniej oszacować łączną długość sufiksów ostatecznych, i tu pojawia się rząd wielkości $\mathcal{O}(n \log^2 n)$.



Rysunek 2. Lekko-ciężka dekompozycja drzewa.

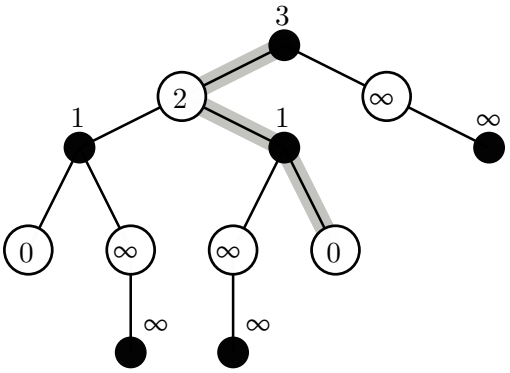
W obu szacowaniach kluczowym narzędziem jest lekko-ciężka dekompozycja ostatecznego drzewa $T_n = G[V_n]$. W dekompozycji tej drzewo jest ukorzenione. Dla każdego wierzchołka który nie jest liściem zdefiniowana jest ciężka krawędź, która prowadzi do syna o najliczniejszym poddrzewie. Ciężkie krawędzie tworzą ścieżki, które z kolei stanowią podział wierzchołków drzewa. Przykład pokazany jest na Rysunku 2.

Przy szacowaniu długości sufiksów nieostatecznych istotną obserwacją jest, że mamy odpowiednio dużo alternatywnych ścieżek powiększających łączących wierzchołek rozdzielający z wolnym serwerem. Co prawda wierzchołki rozdzielające nie powodują łączenia drzew, nie można zatem bezpośrednio użyć argumentu z Lematu 9. Niemniej jednak, używając lekko-ciężkiej dekompozycji, jesteśmy w stanie wyodrębnić zbiór $\mathcal{S}$ poddrzew T_n, których rozmiary sumują się do $\mathcal{O}(n \log n)$. Przy założeniu że wierzchołek rozdzielający ma do wyboru wystarczająco dużo poddrzew z $\mathcal{S}$ w których znajdziemy ścieżkę z niego do wolnego serwera, jesteśmy w stanie opłacić długość najkrótszej ścieżki przypisując ją do jednego z poddrzew w $\mathcal{S}$, podobnie jak w dowodzie Lematu 9.

Szacowanie długości sufiksów ostatecznych wymaga dużo głębszej analizy. Rozważamy tam wierzchołki rozdzielające na konkretnej ciężkiej ścieżce w lekko-ciężkiej dekompozycji T_n i dla każdego takiego wierzchołka c rozważamy czas $\mathrm{last}(c)$. Dokładna analiza tych czasów pozwala ograniczyć długość przecięcia odpowiedniego sufiksu ostatecznego z ciężkimi ścieżkami. Lekkich zaś krawędzi w każdym sufiksie jest niewiele, co wynika z własności dekompozycji lekko-ciężkiej.

W dalszej części niniejszego opracowania opiszemy podejście zastosowane w pracy [A3], aby uzyskać Twierdzenie 8. Twierdzenie to ostatecznie zamyka badany problem w klasie drzew. Jest to zupełnie inne podejście niż zastosowanie dekompozycji lekko-ciężkiej do uzyskania podobnego Twierdzenia 7 w pracy [A2]. W szczególności, w pracy [A3] model z niepomyślnym skojarzeniem odgrywa kluczową rolę. Pierwszy pomysł polega bowiem na tym, aby zastanowić się jak takie niepomyślne skojarzenie może wyglądać.

W tym celu rozważamy grę, gdzie algorytm wybiera przebieg ścieżki powiększającej skojarzenie, zaś przeciwnik wybiera skojarzenie w ten sposób, żeby ścieżka konstruowana przez algorytm była jak najdłuższa. W każdej turze t, kiedy pojawia się klient c_t, gra ta zaczyna się w wierzchołku

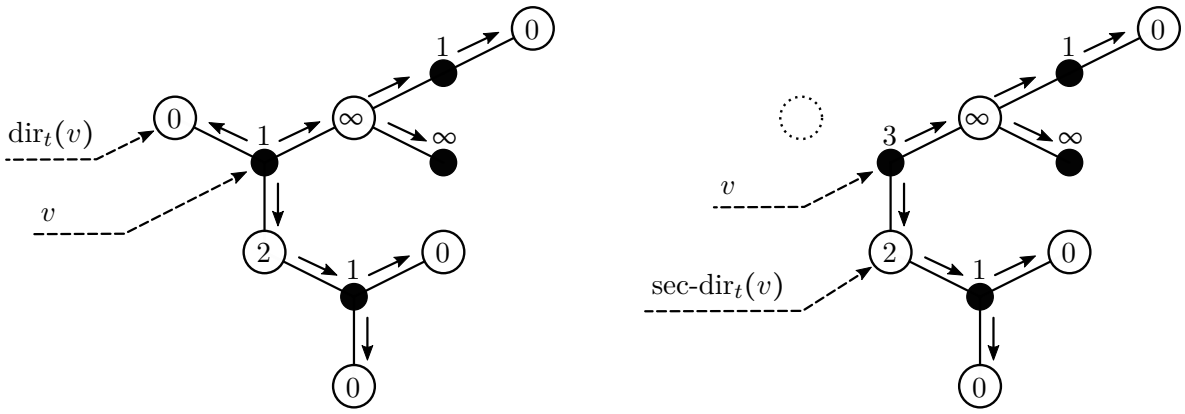


Rysunek 3. Mini-maksowe wartości dla ukorzenionego drzewa z zaznaczoną ścieżką mini-maksową.

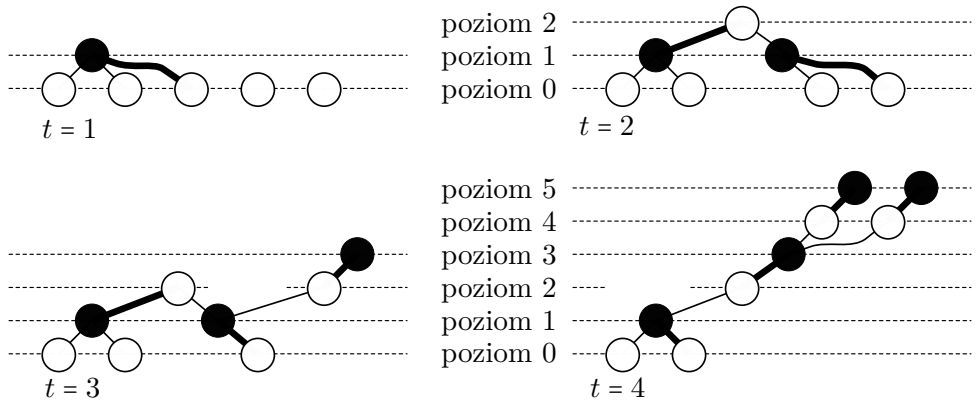
c_t. Pierwszy ruch należy do algorytmu, który wybiera jedną z krawędzi incydentnych do c_t jako pierwszą krawędź konstruowanej ścieżki. Klient c_t nie jest jeszcze skojarzony, zatem algorytm wybiera krawędź nieskojarzoną. Krawędź ta prowadzi do pewnego serwera, gdzie przeciwnik może zdecydować z którym z dotychczas zaprezentowanych klientów w $G[V_t]$ skojarzony jest tenże serwer. Z faktu, że algorytm konstruuje ścieżkę alternującą, musi on teraz podążyć skojarzoną krawędzią wybraną przez przeciwnika. Następną krawędź ścieżki znów wybiera algorytm, i tak dalej aż do momentu, gdy ścieżka zakończy się w liściu drzewa $G[V_t]$. Jeśli liściem tym jest klient, oznacza to, że algorytm nie znalazł ścieżki. Nietrudno zauważyć, że algorytm w każdym kroku chce zminimalizować długość konstruowanej ścieżki, zaś przeciwnik stara się ją zmaksymalizować. Długość takiej mini-maksowej ścieżki dla wierzchołka c_t łatwo wyliczyć, ukorzeniając drzewo w c_t i propagując mini-maksowe wartości ścieżek od liści do korzenia, gdzie dla liścia będącego serwerem przypisujemy wartość 0, zaś dla liścia będącego klientem przypisujemy wartość nieskończoność. Każdy wierzchołek wewnętrzny wylicza swoją wartość na bazie wartości w dzieciach. Przykład ilustrujący wartości mini-maksowe dla konkretnego drzewa ukorzenionego pokazany jest na Rysunku 3.

Niestety, wartości obliczone w ten sposób odzwierciedlają prawdziwą wartość gry tylko w korzeniu. Pozostałe wierzchołki wewnętrzne drzewa nie uwzględniają możliwości pójścia ścieżką alternującą do góry. Idealnie byłoby przechowywać w drzewie wartości które umożliwią odtworzenie ścieżki mini-maksowej z dowolnego wierzchołka. Aby to osiągnąć, wystarczy dla każdego wierzchołka przechowywać zarówno następnika na mini-maksowej ścieżce, jak i następnika na drugiej najlepszej mini-maksowej ścieżce. Niech dla wierzchołka v jego następnikiem na najlepszej mini-maksowej ścieżce od v do liścia w chwili t będzie $\mathrm{dir}_t(v)$. Następnikiem na drugiej najlepszej mini-maksowej ścieżce z wierzchołka v w chwili t niech będzie $\text{sec-dir}_t(v)$ (patrz Rysunek 4). Wówczas, startując z dowolnego wierzchołka u, jesteśmy w stanie odtworzyć mini-maksową ścieżkę z u w następujący sposób. Pierwszym po u wierzchołkiem na ścieżce jest oczywiście $\mathrm{dir}_t(u)$. Załóżmy teraz że znamy już prefiks mini-maksowej ścieżki od wierzchołka u do wierzchołka u' i chcemy wyznaczyć kolejny wierzchołek. Załóżmy, że poprzednikiem wierzchołka u' na wyznaczonym prefiksie ścieżki jest wierzchołek w. Wtedy, jeśli $\mathrm{dir}_t(u') \neq w$, kolejnym wierzchołkiem ścieżki jest $\mathrm{dir}_t(u')$. W przeciwnym wypadku, czyli jeśli ścieżka dociera do u' z wierzchołka realizującego najlepszą mini-maksową ścieżkę dla u', musimy podążyć dalej w stronę wierzchołka który realizuje drugi najlepszy mini-maksowy dystans. Wtedy następnikiem u' jest wierzchołek $\text{sec-dir}_t(u')$.

Dla dowolnego wierzchołka v, wartość najlepszej mini-maksowej ścieżki nazywana jest pierwszym dystansem wierzchołka v, zaś wartość drugiej najlepszej mini-maksowej ścieżki nazywana jest drugim dystansem wierzchołka v. Dla przykładu na Rysunku 4 pierwszym dystansem wierzchołka v jest 1, zaś drugim dystansem wierzchołka v jest 3.



Rysunek 4. Ilustracja pierwszego i drugiego dystansu wierzchołka v.



Rysunek 5. Wierzchołki i ich poziomy dla czterech przykładowych tur $t \in \{1, 2, 3, 4\}$.

Najistotniejszym pojęciem dowodu jest pojęcie poziomu wierzchołka, zdefiniowane jako pierwszy dystans w przypadku klientów i drugi dystans w przypadku serwerów (zobacz Rysunek 5). Poziomy okazują się mieć szereg przydatnych wartości. Są one monotoniczne w czasie. Dodatkowo, dla dowolnego klienta oraz wierzchołków na najlepszej mini-maksowej ścieżce z tego klienta, poziomy wierzchołków są odległościami wierzchołków od końca ścieżki (patrz Rysunek 5). W celu uogólnienia Lematu 9 rozważamy, dla każdej liczby naturalnej k, spójne składowe podgrafu $G[V_t]$ indukowanego na wierzchołkach o poziomie przynajmniej k. Składowe te, dzięki monotoniczności poziomów, łączą się w czasie, co ostatecznie pozwala na amortyzację. Dużą przeszkodą w analizie jest fakt, że fragmenty spójnych składowych umierają w czasie, co powoduje rozpad składowych na żywe fragmenty. Są to dwa przeciwstawne procesy, stąd rozłam dowodu na dwa przypadki ujęte kolejno w Lemacie 23 oraz Lemacie 24 w [A3].

B. Dynamiczne drzewa Steinera i wyrocznie odległości

Problem drzewa Steinera jest jednym z najbardziej fundamentalnych problemów optymalizacji kombinatorycznej. W problemie tym mamy na wejściu ważony graf $G = \langle V, E, d \rangle$ (gdzie $d : E \mapsto \mathbb{R}_+$ jest funkcją przypisującą krawędziom dodatnie wagi zwane również dystansami) oraz zbiór wyróżnionych wierzchołków $S \subseteq V$ nazywanych terminalami. Podgrafem Steinera dla grafu G nazwiemy dowolny podgraf grafu G, w którym każde dwa terminale połączone są ścieżką. Szukamy najtańszego podgrafu Steinera dla grafu G i zbioru S, czyli takiego, w którym suma wag na krawędziach jest jak najmniejsza. Nietrudno się przekonać że podgraf ten jest drzewem, jak wskazuje nazwa omawianego problemu.

Problem drzewa Steinera jest naturalnym uogólnieniem problemu drzewa rozpinającego, który otrzymamy w przypadku gdy $S = V$, czyli zbiór terminali to zbiór wszystkich wierzchołków grafu. Jeśli $|S| = 2$, wówczas problem drzewa Steinera redukuje się do problemu najtańszej ścieżki w grafie pomiędzy dwoma wybranymi wierzchołkami. Podczas gdy zarówno problem drzewa rozpinającego, jak i problem najkrótszej ścieżki, można łatwo rozwiązać zachłannym algorytmem o czasie działania bliskim liniowemu, to problem drzewa Steinera jest problemem NP-trudnym. W prosty sposób można jednak uzyskać szybki algorytm który oblicza 2-aproksymację optymalnego drzewa. W tym celu tworzymy graf pomocniczy $\overline{G}[S] = \langle S, \binom{S}{2}, \overline{d} \rangle$. Jest to pełny graf ważony, gdzie wierzchołkami są wierzchołki z S, zaś wagami są odległości pomiędzy nimi w grafie G. Minimalne drzewo rozpinające graf $\overline{G}[S]$ daje 2-aproksymację drzewa Steinera w G. W dalszej części opracowania minimalne drzewo rozpinające grafu H będziemy oznaczać jako $\mathrm{MST}(H)$. Drzewo $\mathrm{MST}(\overline{G}[S])$ ma koszt co najwyżej dwukrotnie większy niż optymalne drzewo Steinera dla G i S. Co więcej, $\mathrm{MST}(\overline{G}[S])$ jest podgrafem Steinera dla grafu G i zbioru S, pod warunkiem że G jest grafem metrycznym, tzn $G = \overline{G}$, gdzie $\overline{G} = \overline{G}[V]$. Jeśli zaś warunek ten nie zachodzi, można przekształcić $\mathrm{MST}(\overline{G}[S])$ w podgraf Steinera dla G i S zastępując pojedyncze krawędzie odpowiadającymi im najkrótszymi ścieżkami.

Ze względu na swoją rangę problem drzewa Steinera doczekał się wielu algorytmów aproksymacyjnych ze współczynnikiem lepszym niż 2, które działają w czasie wielomianowym [99, 149, 136, 43]. Są to jednak skomplikowane algorytmy, narzucające dużo dodatkowej struktury na utrzymywane drzewa. O stopniu ich skomplikowania świadczy fakt, że autorzy najczęściej nie poświęcają wiele uwagi ich dokładnemu czasowi działania. Skomplikowane struktury trudno jest utrzymywać dynamicznie. Stąd we wszystkich pracach badających modele dynamiczne z ograniczonym budżetem zmian [94, 80, 82, B1] celem jest utrzymywanie $\mathrm{MST}(\overline{G}[S])$, zaś współczynnik aproksymacji 2 jest naturalną i trudną do pokonania barierą w tychże modelach. Najbardziej rozpowszechnionym modelem dynamicznym dla problemu drzewa Steinera jest model Imase i Waxmana, który nazywać będziemy modelem *niezmiennego grafu*. W modelu tym graf $G = \langle V, E, d \rangle$ dany jest w chwili $t = 0$ i pozostaje niezmienny. Zbiór terminali natomiast z czasem podlega modyfikacjom, oznaczymy zatem jako S_t zbiór terminali w chwili $t \in \{1, \ldots, k\}$ i przyjmiemy że $S_0 = \varnothing$. Aktualizacje zbioru terminali prezentowane są w sposób online: $(s_1, r_1) \ll (s_2, r_2) \ll \ldots \ll (s_k, r_k)$. Każda aktualizacja jest dana w formie pary (s_t, r_t), gdzie $s_t \in V$ oraz $r_t \in \{\textsc{add}, \textsc{del}\}$. Jeśli $r_t = \textsc{add}$, wówczas powiększamy zbiór terminali o s_t: $S_t = S_{t-1} \cup \{s_t\}$, zaś jeśli $r_t = \textsc{del}$, wówczas usuwamy s_t ze zbioru terminali: $S_t = S_{t-1} \setminus \{s_t\}$. W każdej turze t algorytm wylicza drzewo T_t, które jest podgrafem $\overline{G} = \overline{G}[V]$ oraz rozpina S_t. Będzie nas tu interesował jak najmniejszy amortyzowany budżet zmian, czyli wartość $\mathtt{B_{am}}$ spełniająca $|T_0 \oplus T_1| + \ldots + |T_{k-1} \oplus T_k| \leqslant \mathtt{B_{am}} \cdot k$. Współczynnikiem aproksymacji algorytmu (zwanym w literaturze także współczynnikiem kompetetywności) będzie wartość $\max_{t \in \{0,\ldots,k\}} \frac{\overline{d}(T_t)}{d(T_t^*)}$, gdzie $d(T_t^*)$ to koszt optymalnego drzewa Steinera w G dla S_t, zaś $\overline{d}(T_t)$ to koszt drzewa T_t w $\overline{G}$. Zaprezentowany wyżej model jest modelem w pełni dynamicznym. W modelu inkrementalnym dopuszczamy tylko $r_t = \textsc{add}$. W dekrementalnym modelu zawężamy się do $r_t = \textsc{del}$, musimy także rozpocząć od niepustego zbioru terminali $S_0 = S$ dla zadanego zbioru terminali S.

Imase i Waxman [94] szeroko przebadali model niezmiennego grafu. W klasycznym modelu online, gdzie $\mathtt{B_{am}} = 1$ zaś zbiór terminali zmienia się w sposób inkrementalny, Imase i Waxman pokazują ograniczenie dolne na współczynnik aproksymacji rzędu logarytmu z liczby terminali. Niemniej jednak, także w modelu inkrementalnym, jeśli zezwolimy na nieco większy budżet zmian rzędu $\mathcal{O}(\epsilon^{-1} \log(\epsilon^{-1}))$, to z pracy Megow *i inni* [125] dostaniemy $(2 + \epsilon)$-aproksymację, gdzie ϵ to dodatni parametr który można wybrać dowolnie. Co więcej, Gu *i inni* [80] pokazują, że nawet pesymistyczny budżet jednej zmiany na operację umożliwia stałą aproksymację. W pełni dynamicznym modelu Imase i Waxman pokazują 4-aproksymacyjny algorytm który osiąga

(amortyzowany) budżet zmian $\mathcal{O}(\sqrt{k})$. Gupta *i inni* [82] w nieco innym modelu poprawiają budżet Imase i Waxmana do stałego na operację, utrzymując w dalszym ciągu 4-aproksymację optymalnego drzewa. Jednak pomimo ogromnego zainteresowania budżetem zmian dla problemu drzewa Steinera, oraz licznych pozytywnych wyników w tym modelu, do momentu powstania pracy [B1], żadna praca nie oferowała podliniowych algorytmów dynamicznych. W pracy [B1] nie tylko poprawiamy wcześniejsze wyniki w modelu z ograniczonym budżetem zmian, ale także przekuwamy je na wydajne algorytmy dynamiczne.

B.1. **Algorytmy dynamiczne z ograniczonym budżetem zmian.** W modelu ograniczonego budżetu zmian dla problemu drzewa Steinera z niezmiennym grafem prezentujemy w pracy [B1] następujące wyniki, które stanowią nowy wkład w dziedzinę. Po pierwsze, nasza analiza umożliwia lepszą stałą aproksymacji w przypadku dekrementalnym, jak pokazuje poniższy lemat.

Lemat 10 (Lemat 4.13 w [B1])**.** *Istnieje algorytm dekrementalny dla problemu drzewa Steinera w modelu niezmiennego grafu, który utrzymuje* $(2+\epsilon)$*-aproksymację przy amortyzowanym budżecie zmian rzędu* $\mathcal{O}(\epsilon^{-1})$ *dla zadanego parametru* $\epsilon > 0$*.*

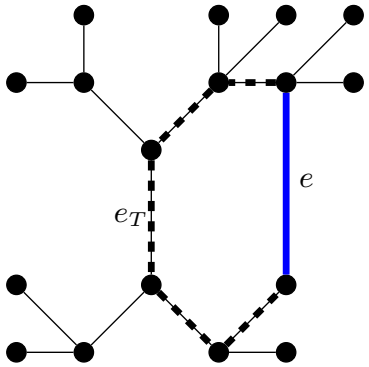
Powyższy wynik poprawia dotychczas najlepszy 4-aproksymacyjny dekrementalny algorytm ze stałym budżetem zmian, który wynika z pracy [94]. Oprócz tego oferujemy lepszy współczynnik aproksymacji oraz lepszy budżet zmian w modelu w pełni dynamicznym.

Lemat 11 (Lemat 4.18 i 4.19 w [B1])**.** *Istnieje algorytm w pełni dynamiczny dla problemu drzewa Steinera w modelu niezmiennego grafu, który utrzymuje* $(2+\epsilon)$*-aproksymację przy amortyzowanym budżecie zmian rzędu* $\mathcal{O}(\epsilon^{-1}\log D)$ *dla zadanego parametru* $\epsilon > 0$*, gdzie* D *to stosunek największej wagi w grafie* $\overline{G}$ *do najmniejszej.*

Jest to poprawa w stosunku do wspomnianego wcześniej 4-aproksymacyjnego algorytmu Imase i Waxmana z budżetem zmian $\mathcal{O}(\sqrt{k})$.

W pozostałej części niniejszego rozdziału nakreślimy główne pomysły oraz techniki prowadzące do powyższych wyników. Przedstawimy też algorytm inkrementalny, który różni się nieco od algorytmu wynikającego z pracy Megow *i innych* [125], a wprowadzone przez nas różnice pozwolą później na efektywną implementację. Powszechnie wiadomo, że każde drzewo rozpinające w grafie można przekształcić na optymalne za pomocą sekwencji podmian krawędzi drzewowych na niedrzewowe. Niemniej jednak, dokonanie wszystkich możliwych podmian nie zagwarantuje niskiego budżetu zmian. Dlatego głównym ciężarem w modelu ograniczonego budżetu zmian jest ostrożne dokonywanie tylko niezbędnych podmian, a także formalne zdefiniowanie które podmiany są dla algorytmu niezbędne. Lemat 10 oraz Lemat 11 bazują na dwóch prostych pomysłach. Po pierwsze, algorytm podmienia krawędź z drzewa na krawędź niedrzewową tylko jeśli ta ma znacznie mniejszy koszt od krawędzi podmienianej. Po wtóre, jeśli usuwany terminal ma duży stopień w aktualnym drzewie, to jego usunięcie można odłożyć w czasie. Aby doprecyzować pierwszy pomysł wprowadzamy pojęcie ϵ-ciężkiej oraz ϵ-efektywnej podmiany. Podmianą dla drzewa $T \subseteq \overline{G}$ nazwiemy parę krawędzi (e, e_T), $e \in \overline{G}, e_T \in T$, gdzie oba końce krawędzi e są wierzchołkami drzewa T, zaś e_T leży na ścieżce w T pomiędzy końcami krawędzi e. Na Rysunku 6 pokazany jest przykład podmiany.

Podmiana jest ϵ-ciężka, jeśli $\overline{d}(e_T) > \epsilon\overline{d}(T)/|V|$, czyli wówczas gdy waga e_T stanowi odpowiednio duży ułamek wagi całego drzewa T. Podmiana jest ϵ-efektywna, gdy $(1+\epsilon)\overline{d}(e) < \overline{d}(e_T)$, czyli wówczas gdy e waży znacząco mniej niż e_T. Jeśli podmiana nie spełnia choć jednego z tych dwóch warunków, to zysk z tej podmiany jest na tyle mały, że nie warto jej dokonywać. Powiemy zatem że podmiana jest (ϵ, c)-dobra jeśli jest zarówno ϵ-efektywna jak i (ϵ/c)-ciężka. W kontekście dostępnej literatury nie jest dużym zaskoczeniem, że jeśli w drzewie nie ma (ϵ, c)-dobrych podmian dla drzewa T, wówczas T nie odbiega znacząco kosztem od $\mathrm{MST}(\overline{G}[V(T)])$, gdzie $V(T)$ to zbiór wierzchołków drzewa T, co formalniej stwierdza poniższy lemat.



RYSUNEK 6. Przykładowa podmiana (e, e_T)

Lemat 12 (Lemat 3.2 w [B1])**.** *Niech $c > \epsilon \geqslant 0$ będą parametrami zaś T niech będzie drzewem w $\overline{G}$. Jeśli nie istnieje (ϵ, c)-dobra podmiana dla drzewa T, wówczas T jest $c(1+\epsilon)/(c-\epsilon)$-aproksymacją $MST(\overline{G}[V(T)])$. Ponadto, jeśli nie istnieje ϵ-efektywna podmiana dla drzewa T, wówczas T jest $(1+\epsilon)$-aproksymacją $MST(\overline{G}[V(T)])$.*

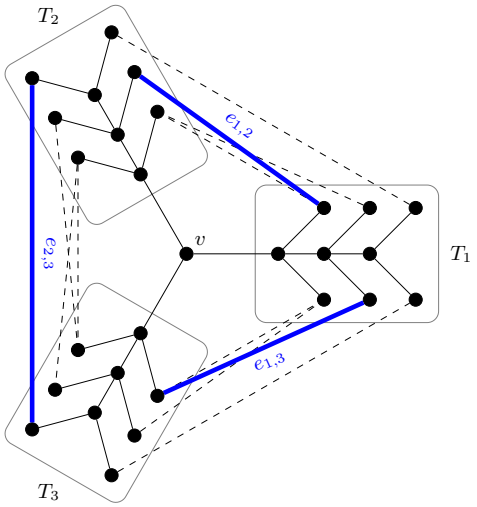
Kolejny lemat umożliwia pozostawienie w drzewie wierzchołków o dużym stopniu, które nie należą już do zbioru terminali. Jest on uogólnieniem przypadku wierzchołków stopnia przynajmniej 3 zawartego w pracy [94]. Nasz dowód nie wzoruje się jednak na [94], dzięki czemu możliwe jest to uogólnienie.

Lemat 13 (Lemat 3.3 w [B1])**.** *Niech $\epsilon \geqslant 0$ oraz $\eta \geqslant 2$ będą parametrami, zaś $T \subseteq \overline{G}$ niech będzie drzewem o zbiorze wierzchołków $S \cup N$, dla którego nie ma ϵ-efektywnych podmian, a każdy wierzchołek z N ma w T stopień przynajmniej η. Wówczas $\overline{d}(T) \leqslant (1+\epsilon)\frac{\eta}{\eta-1}\overline{d}(MST(\overline{G}[S]))$.*

Lemat 12 oraz Lemat 13 stanowią podstawę proponowanych przez nas algorytmów z ograniczonym budżetem zmian, które opiszemy w następnej kolejności.

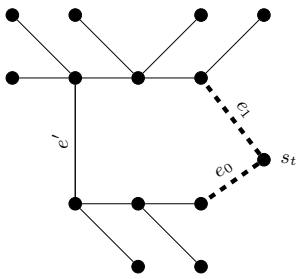
Algorytm dekrementalny. Algorytm dekrementalny w fazie inicjalizacji ustawia $T_0 = \mathrm{MST}(\overline{G}[S])$. Następnie w każdej turze t algorytm działa jak następuje. Niech aktualizacją w turze t będzie para (s_t, r_t) gdzie $r_t = \textsc{del}$, zaś η niech będzie parametrem algorytmu. Jeśli stopień s_t w T_{t-1} jest większy niż η, wówczas algorytm dekrementalny nie robi nic w turze t. Jeśli usuwany terminal s_t ma stopień $s \leqslant \eta$, algorytm usuwa go z drzewa T_{t-1}. Drzewo rozpada się wtedy na $s \leqslant \eta$ komponentów. Komponenty te łączone są w najtańszy możliwy sposób, to znaczy budowane jest minimalne drzewo rozpinające łączące te komponenty krawędziami z $\overline{G}$, po czym krawędzie łączące komponenty dodawane są do T_{t-1} (patrz Rysunek 7). Następnie algorytm dekrementalny wywołuje się rekurencyjnie na byłych sąsiadach v wierzchołka s_t w drzewie T_{t-1}, ale tylko gdy $v \notin S_t$. Drzewo T_t powstaje z T_{t-1} po zakończeniu wszystkich wywołań rekurencyjnych algorytmu dekrementalnego. Dobierzmy $\eta = 1 + \lceil \epsilon^{-1} \rceil$. Wtedy Lemat 13 zagwarantuje, że algorytm dekrementalny w turze t obliczy $2(1+\epsilon)^2$-aproksymację optymalnego drzewa Steinera w G dla S_t. Oznacza to że uzyskujemy $(2+\epsilon)$-aproksymację dla zadanego $\epsilon > 0$ jeśli odpowiednio dobierzemy parametry. Łatwo zauważyć że w trakcie k aktualizacji algorytm usunie terminal z drzewa co najwyżej k razy, każdy zaś usuwany wierzchołek ma stopień co najwyżej $\eta = \mathcal{O}(\epsilon^{-1})$, co przekłada się na łączną liczbę zmian $\mathcal{O}(k\epsilon^{-1})$, a zatem amortyzowany budżet zmian jest rzędu $\mathcal{O}(\epsilon^{-1})$.

Algorytm inkrementalny. Niech aktualizacją w turze t będzie para (s_t, r_t) gdzie $r_t = \textsc{add}$. Jeśli $s_t \in S_{t-1}$, algorytm inkrementalny nie robi nic w turze t. W przeciwnym wypadku algorytm w pierwszej kolejności łączy s_t z drzewem T_{t-1} najtańszą z dostępnych krawędzi. Następnie



RYSUNEK 7. Po usunięciu wierzchołka v z drzewa T_{t-1}, drzewo T_{t-1} rozpada się na poddrzewa T_1, T_2, T_3. Krawędź $e_{i,j}$, dla $i, j \in \{1, 2, 3\}$ oraz $i < j$, to najtańsza krawędź łącząca T_i z T_j w $\overline{G}$. Tworzymy graf pomocniczy, w którym wierzchołkami są poddrzewa T_i, zaś T_i z T_j połączone są krawędzią $e_{i,j}$. W grafie pomocniczym obliczamy minimalne drzewo rozpinające, po czym krawędzie tego drzewa użyte są aby połączyć poddrzewa T_i z powrotem w jedno drzewo.

algorytm dokonuje ϵ-efektywnych podmian w T_{t-1} do momentu gdy nie istnieją $(\epsilon/2, 1+\epsilon)$-dobre podmiany. Odbywa się to tak, iż algorytm rozważa wszystkie krawędzie e incydentne z s_t poza tą, którą już dodał do drzewa. Jeśli e uczestniczy w $(\epsilon/2, 1+\epsilon)$-dobrej podmianie, to algorytm podmienia ją na największą możliwą krawędź drzewową. Po dokonaniu podmian algorytm otrzymuje drzewo T_t. Przykładowy przebieg algorytmu inkrementalnego pokazany jest na Rysunku 8. Istotnym nowym wkładem w algorytm inkrementalny jest pokazanie w pracy [B1] iż rzeczywiście wystarczy rozważyć podmiany dla krawędzi e incydentnych z s_t, gdyż tylko dla takie mogą uczestniczyć w $(\epsilon/2, 1+\epsilon)$-dobrych podmianach. Jest to istotne z punktu widzenia późniejszej implementacji. Na mocy Lematu 12, drzewo T_t jest $2(1+\epsilon)$-aproksymacją optymalnego drzewa Steinera dla S_t. Dodatkowo pokazujemy w [B1] że algorytm inkrementalny wykonuje łącznie $\mathcal{O}(k\epsilon^{-1})$ podmian, co daje amortyzowany budżet zmian rzędu $\mathcal{O}(\epsilon^{-1})$.



RYSUNEK 8. Po dodaniu terminala s_t, algorytm łączy s_t z drzewem T_{t-1} najtańszą krawędzią e_0. Następnie algorytm znajduje podmianę (e_1, e') i dokonuje tej podmiany.

Algorytm w pełni dynamiczny. Algorytm w pełni dynamiczny łączy ze sobą algorytm dekrementalny z inkrementalnym. Rozważmy turę t i odpowiadającą jej aktualizację (s_t, r_t).

Jeśli $r_t = \text{DEL}$, wówczas uruchamiamy algorytm dekrementalny z parametrem $\eta = 1 + \lceil \epsilon^{-1} \rceil$.

Jeśli zaś r_t = ADD, wówczas postępujemy podobnie jak algorytm inkrementalny, są jednak dwie istotne różnice. Po pierwsze, koszt utrzymywanego przez nas drzewa może zmniejszyć się znacząco w przyszłości, dlatego nie możemy ignorować ϵ-lekkich podmian, czyli takich, które nie są ϵ-ciężkie. Po drugie, wierzchołki drzewowe nie będące terminalami mogą zmniejszyć swój stopień w drzewie na skutek podmiany, co może spowodować że należy je usunąć. Zatem w przypadku r_t = ADD, algorytm w pełni dynamiczny postępuje w sposób następujący. Jeśli s_t jest wierzchołkiem drzewa T_{t-1}, wówczas algorytm nie robi nic. W przeciwnym wypadku algorytm najpierw dodaje do T_{t-1} najtańszą krawędź łączącą s_t z drzewem T_{t-1}. Następnie, dopóki krawędź e incydentna z s_t uczestniczy w ϵ-efektywnej podmianie, algorytm podmienia e na najcięższą z krawędzi drzewowych które tworzą z nią podmianę. Na koniec algorytm usuwa wszystkie drzewowe wierzchołki które nie są terminalami i których stopień jest nie większy niż parametr η. Powstaje w ten sposób drzewo T_t. Łatwo zauważyć że po wykonaniu wszystkich kroków algorytmu, dla drzewa T_t nie istnieją ϵ-efektywne podmiany, zaś wszystkie wierzchołki T_t nie będące terminalami mają stopień większy niż η. Zatem Lemat 13 gwarantuje iż T_t jest $2(1+\epsilon)^2$-aproksymacją optymalnego drzewa Steinera w G dla S_t. Jak pokazujemy w pracy [B1], w tym przypadku budżet zmian wzrasta do $\mathcal{O}(\epsilon^{-1}\log D)$, gdzie D to stosunek największej wagi w grafie $\overline{G}$ do najmniejszej.

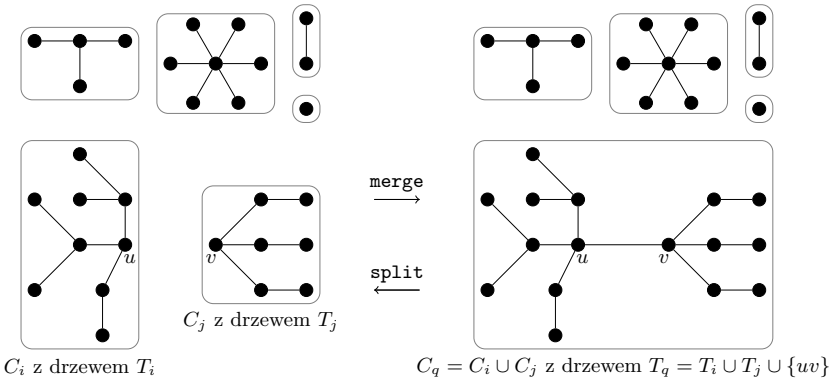
B.2. **Wyrocznie odległości.** Na drodze do efektywnej implementacji algorytmów z ograniczonym budżetem zmian zaprezentowanych w poprzednim podrozdziale pojawia się jeden zasadniczy problem. Mianowicie, algorytmy z ograniczonym budżetem zmian zaprezentowane w poprzednim podrozdziale zakładają dostęp to domknięcia metrycznego grafu G czyli grafu $\overline{G}$. Aby udostępnić algorytmom efektywny dostęp do grafu $\overline{G}$, wprowadzamy w pracy [B1] struktury danych nazywane *wyroczniami odległości*. Wyrocznie te w fazie inicjalizacji przetwarzają ważony graf G, aby móc potem efektywnie odpowiadać na zapytania o odległości w G. Interfejs wyroczni odległości zaprezentowanych w [B1] jest dostosowany do potrzeb algorytmów dynamicznych. Co więcej, algorytmy mają dostęp do grafu G wyłącznie poprzez wyrocznie odległości. Oferujemy dwa warianty wyroczni: inkrementalny oraz w pełni dynamiczny. Wariant inkrementalny wyroczni użyty będzie na potrzeby algorytmu inkrementalnego, zaś wariant w pełni dynamiczny posłuży do implementacji algorytmu w pełni dynamicznego.

Każda wyrocznia z zaproponowanych w [B1], w momencie inicjalizacji dostaje ważony graf $G = \langle V, E, d\rangle$, na którym przeprowadza fazę inicjalizacji. Wyrocznia utrzymuje podział wierzchołków V na klasy kolorów $C_1, \ldots, C_l$. Oznacza to że każdy wierzchołek przynależy do jakiejś klasy: $\bigcup_{i=1}^{l} C_i = V$ oraz że klasy są parami rozłączne: $C_i \cap C_j = \varnothing$ dla $i \neq j$. Każdy kolor może być aktywny bądź nieaktywny.

Inkrementalny wariant wyroczni umożliwia następujące operacje:

- `distance`(v, i) – podaj odległość z wierzchołka v do najbliższego wierzchołka o kolorze i,
- `nearest`(v, j) – podaj odległość od wierzchołka v do j-tego najbliższego aktywnego koloru (dla stałego j),
- `activate`(i) – aktywuj kolor i,
- `merge`(i, j) – połącz aktywne klasy kolorów C_i oraz C_j w aktywną klasę C_q, gdzie $q \in \{i, j\}$.

W pełni dynamiczny wariant udostępnia dodatkowo operację rozdzielania kolorów, niemniej jednak zarówno łączenie jak i rozdzielanie kolorów odbywa się w określony sposób. Mianowicie, każda klasa C_i reprezentująca kolor i powiązana jest z drzewem T_i rozpinającym C_i. Drzewa T_i nie muszą być w żaden sposób powiązane z grafem G. Łączenie kolorów jest jednocześnie łączeniem drzew im odpowiadających za pomocą zadanej krawędzi. Rozdzielanie koloru to podział odpowiadającego drzewa na dwa poprzez usunięcie z niego krawędzi. Przykład pokazany jest na Rysunku 9.



RYSUNEK 9. Przykład operacji $\texttt{merge}(i, j, u, v)$ oraz $\texttt{split}(q, u, v)$

Tak więc wariant w pełni dynamiczny udostępnia następujące operacje:

- $\texttt{distance}(v, i)$ – podaj odległość z wierzchołka v do najbliższego wierzchołka o kolorze i,
- $\texttt{nearest}(v, j)$ – podaj odległość od wierzchołka v do j-tego najbliższego aktywnego koloru (dla stałego j),
- $\texttt{activate}(i)$ – aktywuj kolor i,
- $\texttt{deactivate}(i)$ – dezaktywuj kolor i,
- $\texttt{merge}(i, j, u, v)$ – połącz aktywne klasy kolorów C_i oraz C_j o drzewach T_i oraz T_j w aktywną klasę C_q, gdzie $q \in \{i, j\}$ zaś $T_q = T_i \cup T_j \cup \{u, w\}$.
- $\texttt{split}(q, u, v)$ – jeśli $\{u, w\}$ jest krawędzią T_q oraz kolor q jest aktywny, wówczas podziel C_q na aktywne klasy C_i oraz C_j o drzewach będących spójnymi składowymi $T_q \setminus \{u, w\}$

Konstrukcja wyroczni jest wysokopoziomowa i modularna. Oznacza to, że pokazujemy najpierw wyrocznię abstrakcyjną, która opiera się na kilku modułach i spełnia kilka ogólnych własności które uzależniają moduły od siebie i łączą je w jedną strukturę. Każdy moduł może być implementowany na różne sposoby co daje różne instancje abstrakcyjnej wyroczni, różniące się od siebie zużyciem pamięci, współczynnikiem aproksymacji oraz czasami działania pojedynczych operacji. Wyabstrahowanie ogólnych własności spełnianych przez każdą instancję wyroczni jest w pracy [B1] niezwykle ważne, ponieważ implementacje algorytmów jawnie korzystają z tych własności. Konkretne instancje wyroczni używane w [B1] ujęte są w następujących trzech twierdzeniach.

Twierdzenie 14 (Twierdzenie 5.8 w [B1]). *Niech $G = (V, E, d)$ będzie grafem ważonym i niech $n = |V|$ oraz $m = |E|$. Istnieje 3-aproksymacyjna w pełni dynamiczna wyrocznia odległości dla G, której oczekiwany czas konstrukcji wynosi $\mathcal{O}(\sqrt{n}(m + n \log n))$, która zajmuje oczekiwaną pamięć rzędu $\mathcal{O}(n\sqrt{n} \log n)$, i która zużywa oczekiwany czas rzędu $\mathcal{O}(\sqrt{n} \log n)$ na operację.*

Twierdzenie 15 (Twierdzenie 5.14 w [B1]). *Niech $G = (V, E, d)$ będzie ważonym grafem planarnym i niech $n = |V|$ oraz D niech będzie stosunkiem największej odległości w G do najmniejszej. Dla dowolnego $\epsilon > 0$ istnieje $(1 + \epsilon)$-aproksymacyjna inkrementalna wyrocznia odległości dla G, której czas konstrukcji wynosi $\mathcal{O}(\epsilon^{-1} n \log^2 n \log D)$, która zajmuje pamięć rzędu $\mathcal{O}(\epsilon^{-1} n \log n \log D)$, i która zużywa oczekiwany czas rzędu $\mathcal{O}(\epsilon^{-1} \log^2 n \log D)$ na operację lub też czas rzędu $\mathcal{O}(\epsilon^{-1} \log^2 n \log \log n \log D)$ na operację w deterministycznym wariancie.*

Twierdzenie 16 (Twierdzenie 5.16 w [B1] dla $\rho = \sqrt{n}$). *Niech $G = (V, E, d)$ będzie ważonym grafem planarnym i niech $n = |V|$ oraz D niech będzie stosunkiem największej odległości w G do najmniejszej. Dla dowolnego $\epsilon > 0$ istnieje $(1 + \epsilon)$-aproksymacyjna w pełni dynamiczna wyrocznia odległości dla G, której czas konstrukcji wynosi $\mathcal{O}(\epsilon^{-1} n \log^2 n \log D)$, która zajmuje pamięć rzędu $\mathcal{O}(\epsilon^{-1} n \log^2 n \log D)$, i która zużywa czas rzędu $\mathcal{O}(\epsilon^{-1} \sqrt{n} \log n \log D)$ na operację.*

B.3. **Efektywne algorytmy dynamiczne.** Do momentu powstania pracy [B1] nie były znane żadne algorytmy dynamiczne dla problemu drzewa Steinera, brak jest zatem punktu odniesienia który dyktowałby jaka wydajność jest interesująca.

W modelu niezmiennego grafu, najprostszy algorytm dynamiczny policzyłby w fazie inicjalizacji domknięcie przechodnie grafu G, a następnie przy każdej aktualizacji przeliczałby $\mathrm{MST}(\overline{G}[S_t])$ statycznym algorytmem. Prowadzi to do aktualizacji w czasie $\widetilde{\mathcal{O}}(|S_t|^2)$. Zasadniczo od algorytmów dynamicznych oczekuje się, żeby były istotnie wydajniejsze niż aplikacja algorytmu statycznego dla każdej aktualizacji.

W przypadku drzew Steinera istnieje jednak wydajniejszy prosty algorytm dynamiczny. Jest on oparty na dynamicznym algorytmie przliczającym minimalne drzewo rozpinające na grafie aktualizowanym poprzez dodanie bądź usunięcie krawędzi [90]. Jest to niezwykle wydajny w pełni dynamiczny algorytm, gdzie czas aktualizacji jest rzędu $\mathcal{O}(\log^4 n)$ dla n-wierzchołkowego grafu. Algorytm ten można wykorzystać aby utrzymywać dynamicznie $\mathrm{MST}(\overline{G}[S_t])$ co daje 2-aproksymację optymalnego drzewa Steinera w G. Inicjalizujemy strukturę z pracy [90] grafem G_{MST} o n wierzchołkach i bez krawędzi, gdzie $n = |V|$ to liczba wierzchołków w grafie G dla którego chcemy aproksymować drzewo Steinera. Przy każdej aktualizacji w postaci (s_t, r_t), jeśli $r_t = \textsc{add}$, wówczas dodajemy do grafu G_{MST} wszystkie krawędzie łączące s_t z wierzchołkami S_{t-1} w grafie $\overline{G}$. Jeśli zaś $r_t = \textsc{del}$, wówczas krawędzie incydentne z s_t usuwamy z grafu G_{MST}. Takie podejście daje w pełni dynamiczny algorytm z aktualizacją w czasie $\widetilde{\mathcal{O}}(n)$.

Naszym celem w pracy [B1] jest zatem przełamanie naturalnej bariery $\mathcal{O}(n)$ dla czasu aktualizacji w algorytmach dynamicznych. Konkretne czasy podane w twierdzeniach poniżej są rezultatem implementacji algorytmów z Rozdziału B.1 za pomocą odpowiednich wyroczni z Rozdziału B.2.

Algorytm dekrementalny. Oferujemy dwa algorytmy dekrementalne. Pierwszy z nich jest implementacją dekrementalnego algorytmu z ograniczonym budżetem zmian (Lemat 10) za pomocą w pełni dynamicznej wyroczni dla grafów ogólnych (Twierdzenie 14). Dokładne czasy zawarte są w poniższym twierdzeniu.

Twierdzenie 17 (Twierdzenie 6.3 w [B1])**.** *Niech $G = (V, E, d)$ będzie grafem i niech $n = |V|$ oraz $m = |E|$ oraz niech $\epsilon > 0$. Niech S będzie początkowym zbiorem terminali z którego terminale są sukcesywnie usuwane. Istnieje dekrementalny algorytm, który po preprocessingu w oczekiwanym czasie $\mathcal{O}(\sqrt{n}(m + n\log n) + |S|\sqrt{n}\log^4 n)$, utrzymuje $(6+\epsilon)$-aproksymację drzewa Steinera dla S, obsługując każdą operację usunięcia z S w oczekiwanym amortyzowanym czasie $\mathcal{O}(\epsilon^{-1}\sqrt{n}\log n)$. Algorytm używa oczekiwaną pamięć rzędu $\mathcal{O}(n\sqrt{n}\log n)$.*

Nieco lepszy algorytm dekrementalny możliwy jest dla grafów planarnych, jeśli zaimplementujemy algorytm dekrementalny z ograniczonym budżetem zmian (Lemat 10) za pomocą w pełni dynamicznej wyroczni dla grafów planarnych (Twierdzenie 16).

Twierdzenie 18 (Twierdzenie 6.4 w [B1])**.** *Niech $G = (V, E, d)$ będzie grafem planarnym i niech $n = |V|$ oraz $m = |E|$ oraz niech $\epsilon > 0$ zaś D niech będzie stosunkiem największej odległości w G do najmniejszej. Niech S będzie początkowym zbiorem terminali z którego terminale są sukcesywnie usuwane. Istnieje dekrementalny algorytm, który po preprocessingu w oczekiwanym czasie $\mathcal{O}(\epsilon^{-1}n\log^2 n\log D + |S|\epsilon^{-0.5}\sqrt{n}\log^{2.5} n\log D)$, utrzymuje $(2+\epsilon)$-aproksymację drzewa Steinera dla S, obsługując każdą operację usunięcia z S w amortyzowanym czasie $\mathcal{O}(\epsilon^{-1.5}\sqrt{n}\log D)$.*

Głównym wyzwaniem w implementacji algorytmu dekrementalnego jest umiejętność szybkiego wyznaczenia krawędzi łączących składowe powstałe po usunięciu z drzewa wierzchołka o stopniu ograniczonym przez parametr η. Nie jest apriori jasne jak to zrobić, jako że kandydatów na krawędzie łączące jest zbyt wielu aby ich wszystkich przeglądać. Z pomocą przychodzi

tu dynamiczny algorytm utrzymujący minimalny las rozpinający [90]. Wylicza on zastępcze krawędzie, działając na specjalnie przygotowanym pomocniczym grafie, który dzięki ostrożnej analizie zawiera wszystkie potrzebne krawędzie i daje się efektywnie aktualizować.

Algorytm inkrementalny. Podobnie jak w przypadku dekrementalnym, otrzymujemy dwa różne algorytmy inkrementalne, implementując algorytm inkrementalny z ograniczonym budżetem zmian (opisany w Rozdziale B.1) za pomocą dwóch różnych wyroczni odległości. Zastosowanie wyroczni w pełni dynamicznej dla grafów ogólnych (Twierdzenie 14) da algorytm inkrementalny dla grafów ogólnych opisany poniższym twierdzeniem.

Twierdzenie 19 (Twierdzenie 6.10 w [B1])**.** *Niech $G = (V, E, d)$ będzie grafem ważonym, $n = |V|$ oraz niech $\epsilon > 0$. Niech także S będzie zmieniającym się przez dodawanie elementów zbiorem terminali, początkowo pustym. Wówczas istnieje algorytm inkrementalny, który po preprocessingu w oczekiwanym czasie $\mathcal{O}(\epsilon^{-1}\sqrt{n}(m + n\log(n/\epsilon)))$ utrzymuje $(6+\epsilon)$-aproksymację drzewa Steinera dla S, obsługując operacje dodania terminala w amortyzowanym czasie $\mathcal{O}(\epsilon^{-1}\sqrt{n})$. Algorytm zajmuje $\mathcal{O}(\epsilon^{-1}n\sqrt{n}\log n\log(n/\epsilon))$ pamięci.*

Alternatywą jest implementacja algorytmu inkrementalnego za pomocą inkrementalnej wyroczni dla grafów planarnych (Twierdzenie 15). Otrzymamy wtedy następujący algorytm inkrementalny.

Twierdzenie 20 (Twierdzenie 6.11 w [B1])**.** *Niech $G = (V, E, d)$ będzie planarnym grafem ważonym, $n = |V|$ oraz niech $\epsilon > 0$. Niech także S będzie zmieniającym się przez dodawanie elementów zbiorem terminali, początkowo pustym. Wówczas istnieje algorytm inkrementalny, który po preprocessingu w oczekiwanym czasie $\mathcal{O}(\epsilon^{-2}n\log n\log(n/\epsilon)\log D)$ utrzymuje $(2+\epsilon)$-aproksymację drzewa Steinera dla S, obsługując każdą operację dodania w amortyzowanym czasie $\mathcal{O}(\epsilon^{-2}\log^2 n\log\log n\log(n/\epsilon)\log D)$. Algorytm używa $\mathcal{O}(\epsilon^{-2}n\log n\log(n/\epsilon)\log D)$ pamięci.*

Pomimo że $(2+\epsilon)$-aproksymacyjny algorytm inkrementalny z budżetem zmian $\mathcal{O}(\epsilon^{-1}\log\epsilon^{-1})$ wynika ze wcześniejszej literatury [125], nikomu nie udało się efektywnie zaimplementować tego algorytmu. Przeszkodą jest tu potrzeba szybkiego znajdowania efektywnych i ciężkich podmian. Przypomnijmy, że algorytm inkrementalny z ograniczonym budżetem zmian z Rozdziału B.1 najpierw łączy nowy terminal s_t z drzewem T_{t-1} najtańszą krawędzią, co można osiągnąć jednym zapytaniem w postaci $\texttt{distance}(s_t, j)$ do wyroczni odległości. Następnie jednak algorytm inkrementalny przegląda krawędzie incydentne z s_t w poszukiwaniu $(\epsilon/2, 1+\epsilon)$-dobrych podmian. Nawet samo przejrzenie wszystkich krawędzi incydentnych z s_{t-1} stanowi większy koszt niż możemy sobie na to pozwolić. Dlatego potrzebne są tu nietrywialne pomysły które pozwolą zaaplikować wszystkie potrzebne podmiany w czasie podliniowym.

Podstawowym pomysłem jest tutaj następująca idea. Powiedzmy że interesują nas ζ-efektywne podmiany dla pewnego parametru $\zeta > 0$ (pamiętajmy że ζ jest rzędu ϵ). Rozważmy $\overline{G}' = (V, \binom{V}{2}, d')$ jako klikę z wagami d' zwracanymi przez konkretną wyrocznię odległości. Tak otrzymany graf $\overline{G}'$ jest grafem aproksymacji odległości w $\overline{G}$ widzianym przez wyrocznie. Zaokrąglimy teraz wagi krawędzi $\overline{G}'$ w górę do potęg $(1+\zeta)$. Tak otrzymany $\overline{G}'$ zawiera teraz $\mu(1+\zeta)$-aproksymację dystansów z $\overline{G}$, gdzie μ to współczynnik aproksymacji użytej wyroczni. Algorytm będzie szukał podmian w grafie $\overline{G}'$ (w pracy pokazujemy formalnie poprawność takiego kroku). Przyjmijmy także że wagi w $\overline{G}'$ są z przedziału $[1, D']$, gdzie D' będzie dobrym przybliżeniem stosunku D największej do najmniejszej wagi w oryginalnym grafie G. Każdej krawędzi $uv \in E(\overline{G}')$ przypiszemy jej poziom, zdefiniowany jako $\texttt{lvl}(uv) = \log_{1+\zeta} d'(uv)$ (zauważmy że wobec powyższego poziomy krawędzi są liczbami naturalnymi). Mamy zatem dostępne poziomy $0, 1, 2, \ldots, h = \log_{1+\zeta} D'$. W każdej turze t, dla każdego $j \in \{0, 1, \ldots, h\}$ definiujemy warstwę L_j,

w której znajdują się krawędzie o poziomie co najwyżej j. Dla każdej warstwy L_j inicjalizujemy wyrocznię odległości $\mathbb{D}_j$, gdzie kolory odpowiadają spójnym składowym T_t w warstwie L_j. Składowe te będą się zmieniać dynamicznie razem z drzewem T_t.

Powiedzmy, że nowy terminal s_t został już podpięty do drzewa T_t najtańszą krawędzią i należy teraz zaaplikować w T_t wszystkie interesujące nas podmiany. Główny pomysł jest taki, że będziemy szukać podmian idąc warstwami od dołu do góry. Powiedzmy zatem że rozważamy aktualnie warstwę L_j. Interesują nas podmiany (e, e') takie że $\texttt{lvl}(e') > j$ oraz $\texttt{lvl}(e) \leqslant j$. Niech C będzie spójną składową w warstwie L_j zawierającą s_t. Składowa C z definicji zawiera krawędzie o poziomie co najwyżej j, oraz każda ścieżka w T_t łącząca C z inną składową w L_j zawiera krawędź o poziomie większym niż j. Ponieważ s_t ma kolor składowej C, odpytujemy $\mathbb{D}_j$ o drugi najbliższy kolor do s_t za pomocą operacji $\texttt{nearest}(s_t, 2)$. Powiedzmy że wyrocznia znalazła odległość której odpowiada wierzchołek s'. Jeśli $\texttt{lvl}(s_t s') \leqslant j$, wówczas znaleźliśmy podmianę (e, e'), gdzie $e = s_t s'$ oraz e' jest najcięższą krawędzią na ścieżce z s_t do s' w T_t. Podmiana (e, e') jest ζ-efektywna jako że wagi krawędzi w sąsiednich warstwach różnią się przynajmniej o czynnik $(1 + \zeta)$. Po znalezieniu podmiany poprawiamy wyrocznie $\mathbb{D}_i$ które tego wymagają i szukamy następnej podmiany na tym samym poziomie j. Jeśli podmiany nie było, oznacza to że krawędzie do poziomu j nie tworzą efektywnej podmiany, zatem kontynuujemy szukanie podmian na poziomie $j + 1$, rozważając warstwę L_{j+1}.

W opisanym algorytmie czas zaaplikowania wszystkich efektywnych podmian jest proporcjonalny do liczby warstw ($\log_{1+\zeta} D'$) wymnożonej przez liczbę znalezionych podmian (amort. $\mathcal{O}(\epsilon^{-1})$ na dodanie terminala). Aby zredukować liczbę odwiedzonych warstw, definiujemy w pracy (bardzo niski) poziom $\texttt{lvl}_\perp(T) = \lfloor \log_{1+2\zeta} \frac{2\zeta d'(T)}{8\mu^2(1+2\zeta)^3 n} \rfloor$. Znaczenie tego poziomu jest takie, iż krawędzie na poziomie $\texttt{lvl}_\perp(T_t)$ mają tak niski koszt, że nie podmieniamy ich. Iterujemy się zatem przez warstwy $L_{\texttt{lvl}_\perp(T)}, \ldots, L_h$ (gdzie T ma maksymalną wagę spośród $T_1, \ldots, T_t$), co daje z grubsza $\mathcal{O}(\log n)$ warstw które musimy przeszukać.

Algorytm w pełni dynamiczny. Aby otrzymać algorytm w pełni dynamiczny umiejętnie łączymy algorytm inkrementalny z algorytmem dekrementalnym. Powstałe w pełni dynamiczne algorytmy dla problemu drzewa Steinera z niezmiennym grafem podsumowane są w następujących dwóch twierdzeniach.

Twierdzenie 21 (Twierdzenie 6.14 w [B1])**.** *Niech $G = (V, E, d)$ będzie grafem gdzie $n = |V|$, $m = |E|$ oraz $\epsilon > 0$. Niech D będzie stosunkiem największej wagi w G do najmniejszej. Niech S będzie początkowo pustym zbiorem terminali modyfikowanym przez wstawiania i usunięcia. Wówczas istnieje algorytm, który po preprocessingu w oczekiwanym czasie $\mathcal{O}(\sqrt{n}(m + \epsilon^{-1} n \log n \log D))$ utrzymuje $(6 + \epsilon)$-aproksymację drzewa Steinera dla S w G, obsługując każdą modyfikację w oczekiwanym czasie $\widetilde{\mathcal{O}}(\epsilon^{-2}\sqrt{n}\log^2 D)$. Algorytm używa $\mathcal{O}(n\sqrt{n}\log n\epsilon^{-1}\log D)$ pamięci.*

Współczynnik aproksymacji można tu poprawić, jeśli rozważymy grafy planarne i użyjemy odpowiedniej dla nich wyroczni odległości.

Twierdzenie 22 (Twierdzenie 6.16 w [B1])**.** *Niech $G = (V, E, d)$ będzie grafem planarnym gdzie $n = |V|$ oraz $\epsilon > 0$. Niech D będzie stosunkiem największej wagi w G do najmniejszej. Niech S będzie początkowo pustym zbiorem terminali modyfikowanym przez wstawiania i usunięcia. Wówczas istnieje algorytm, który po preprocessingu w czasie $\mathcal{O}(\epsilon^{-2} n \log^2 n \log^2 D)$ utrzymuje $(2+\epsilon)$-aproksymację drzewa Steinera dla S w G, obsługując każdą modyfikację w amortyzowanym czasie $\widetilde{\mathcal{O}}(\epsilon^{-2}\log^{2.5} D\sqrt{n})$. Algorytm zużywa pamięć $\mathcal{O}(\epsilon^{-2} n \log^2 n \log^2 D)$.*

B.4. **Dynamiczne drzewa Steinera za pomocą emulatorów dwudzielnych.** Algorytmy zaprezentowane w poprzednim rozdziale przełamują barierę liniowego czasu na aktualizację.

Niemniej jednak, za cenę wyższej aproksymacji, można otrzymać dynamiczne algorytmy jeszcze bardziej wydajne. Narzędziem pozwalającym uzyskać wydajniejsze algorytmy jest graf dwudzielny nazywany *emulatorem dwudzielnym*, który zdefiniowany jest dla dowolnego grafu ważonego. Intuicyjnie, emulator dwudzielny jest to rzadki graf który przechowuje informację o dystansach w oryginalnym grafie. Nie będziemy tu rozwijać bardziej koncepcji emulatora dwudzielnego, podamy tylko ostateczne algorytmy które udało się tym sposobem uzyskać.

Twierdzenie 23 (Twierdzenie 7.6 w [B1])**.** *Niech $G = (V, E, d)$ będzie grafem gdzie $n = |V|$, $m = |E|$ zaś $k \geqslant 1$ niech będzie całkowitym parametrem. Niech S będzie początkowo pustym zbiorem terminali modyfikowanym przez wstawiania oraz usunięcia. Wówczas istnieje algorytm, który po preprocessingu w oczekiwanym czasie $\mathcal{O}(kmn^{1/k})$ utrzymuje $(8k-4)$-aproksymację drzewa Steinera dla S, obsługując każdą modyfikację w oczekiwanym amortyzowanym czasie $\mathcal{O}(kn^{1/k}\log^4 n)$.*

Dla grafów planarnych sytuacja przedstawia się następująco.

Twierdzenie 24 (Twierdzenie 7.10 w [B1])**.** *Niech $G = (V, E, d)$ będzie grafem planarnym i niech $0 < \epsilon \leqslant 1$. Niech S będzie początkowo pustym zbiorem terminali modyfikowanym przez wstawiania oraz usunięcia. Wówczas istnieje algorytm, który po preprocessingu w czasie $\mathcal{O}(\epsilon^{-1}n\log^2 n)$ utrzymuje $(4+\epsilon)$-aproksymację drzewa Steinera dla S w G, obsługując aktualizacje w amortyzowanym czasie $\mathcal{O}(\epsilon^{-1}\log^6 n)$.*

C. Dynamiczne kolorowanie grafów przedziałowych

W problemie kolorowania grafów przedziałowych dostajemy na wejściu ciąg n przedziałów $\mathcal{I} = \{[a_i, b_i)\}_{i=1}^n$. Ciąg ten można zinterpretować jako graf $G_\mathcal{I}$, w którym dla każdego przedziału $I \in \mathcal{I}$ istnieje reprezentujący go wierzchołek v_I, zaś zbiór krawędzi to zbiór par wierzchołków $v_I v_J$ takich że $I \cap J \neq \emptyset$. Wtedy graf $G_\mathcal{I}$ nazywa się grafem przecięć zbioru $\mathcal{I}$ i należy do klasy grafów przedziałowych, natomiast $\mathcal{I}$ jest reprezentacją przedziałową grafu $G_\mathcal{I}$. W niniejszym opracowaniu rozważać będziemy również klasę grafów równoprzedziałowych: należą do niej grafy przedziałowe $G_\mathcal{I}$ gdzie dla każdego przedziału $[a_i, b_i) \in \mathcal{I}$ zachodzi $b_i = a_i + 1$. Właściwym k-kolorowaniem grafu $G = (V, E)$ nazwiemy przyporządkowanie $c : V \mapsto \{1, \ldots, k\}$ wierzchołkom grafu kolorów ze zbioru $\{1, \ldots, k\}$ w taki sposób, iż każde dwa sąsiadujące wierzchołki mają różne kolory, czyli jeśli $vw \in E$ to $c(v) \neq c(w)$. Mając dany zbiór przedziałów $\mathcal{I}$, jego właściwym k-kolorowaniem jest właściwe k-kolorowanie grafu $G_\mathcal{I}$, gdzie kolor wierzchołka v_I utożsamiamy z kolorem przedziału I. Liczbą chromatyczną grafu czy też zbioru przedziałów nazwiemy najmniejsze k dla którego istnieje właściwe k-kolorowanie. W problemie kolorowania celem jest znalezienie liczby chromatycznej grafu oraz odpowiadającego jej kolorowania. W ogólnych grafach, dla dowolnego $\epsilon > 0$, nawet $(n^{1-\epsilon})$-aproksymacja liczby chromatycznej jest problemem NP-trudnym [101, 150]. Natomiast w klasie grafów przedziałowych przy zadanej reprezentacji dla problemu kolorowania istnieje bardzo prosty zachłanny algorytm, który sortuje przedziały po początkach a następnie każdemu z nich przydziela najmniejszy dostępny kolor. Co więcej, wiadomo że klasa grafów przedziałowych jest podklasą grafów doskonałych, dla których liczba chromatyczna jest równa liczbie klikowej, czyli maksymalnej liczbie przedziałów przecinających się w jednym punkcie [75].

W artykułach [C1] i [C2] zajmujemy się problemem kolorowania dynamicznie zmieniającego się grafu przedziałowego, zaś naszym głównym celem jest ograniczenie budżetu zmian. Rozważamy zarówno model *inkrementalny*, jak i model *w pełni dynamiczny*.

W modelu *inkrementalnym* mamy zadane całkowitoliczbowe dodatnie parametry k i l oraz ciąg n przedziałów $\mathcal{I} = \{[a_i, b_i)\}_{i=1}^n$, taki że $\mathcal{I}$ jest k-kolorowalny. Ciąg ten definiuje $n+1$ instancji dla problemu kolorowania zdefiniowanych następująco: $\mathcal{I}_0 = \emptyset$, zaś $\mathcal{I}_j = \{[a_i, b_i)\}_{i=1}^j$ dla $j \in \{1, \ldots, n\}$. Zatem $\mathcal{I}_j$ różni się od $\mathcal{I}_{j-1}$ dokładnie jednym dodatkowym przedziałem $I_j = [a_j, b_j)$. W momencie

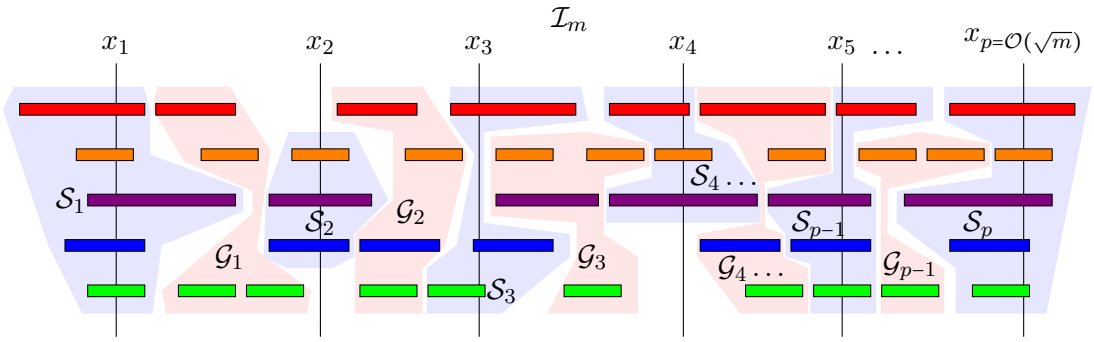
ujawnienia przedziału I_j dla algorytmu dostępna jest tylko instancja $\mathcal{I}_j$ nie są zaś znane przyszłe przedziały I_i dla $i > j$. Po ujawnieniu przedziału I_j zadaniem algorytmu jest wyliczenie właściwego l-kolorowania c_j dla $\mathcal{I}_j$, mając do dyspozycji l-kolorowanie c_{j-1} dla $\mathcal{I}_{j-1}$. Budżetem zmian $\mathtt{B}_j$ w turze j jest tutaj liczba przedziałów, dla których c_j oraz c_{j-1} przypisują różne kolory. Bardziej formalnie $\mathtt{B}_j = |\{i \in \{1, \ldots, j-1\} : c_{j-1}(I_i) \neq c_j(I_i)\}| + 1$.

W modelu *w pełni dynamicznym* także mamy zadane dodatnie parametry całkowitoliczbowe k i l oraz rozpoczynamy od pustej reprezentacji $\mathcal{I}_0 = \varnothing$ grafu przedziałowego. Instancja $\mathcal{I}_j$ dla $j \in \{1, \ldots, n\}$ także różni się od instancji $\mathcal{I}_{j-1}$ jednym przedziałem, przy czym w odróżnieniu od modelu inkrementalnego $\mathcal{I}_j$ otrzymujemy z $\mathcal{I}_{j-1}$ poprzez dodanie nowego przedziału lub usunięcie istniejącego przedziału z $\mathcal{I}_{j-1}$. Algorytm zna parametr k dla którego każda instancja $\mathcal{I}_j$ jest k-kolorowalna. Zadaniem algorytmu w kroku j znów jest wyliczenie właściwego l-kolorowania c_j dla $\mathcal{I}_j$, mając do dyspozycji instancję $\mathcal{I}_{j-1}$, usuwany bądź dodawany przedział I_j oraz właściwe l-kolorowanie c_{j-1} dla $\mathcal{I}_{j-1}$. Budżetem zmian $\mathtt{B}_j$ w turze j jest analogicznie liczba przedziałów dla których kolory w kolorowaniu c_{j-1} i kolorowaniu c_j są różne. Bardziej formalnie, w przypadku gdy $\mathcal{I}_j = \mathcal{I}_{j-1} \cup \{I_j\}$, zdefiniujemy budżet zmian w chwili j jako $\mathtt{B}_j = |\{I \in \mathcal{I}_{j-1} : c_{j-1}(I) \neq c_j(I)\}| + 1$. Natomiast w przypadku gdy $\mathcal{I}_j = \mathcal{I}_{j-1} \setminus \{I_j\}$ zdefiniujemy budżet zmian w chwili j jako $\mathtt{B}_j = |\{I \in \mathcal{I}_j : c_{j-1}(I) \neq c_j(I)\}|$.

Zarówno w modelu inkrementalnym jak i w pełni dynamicznym można rozważać zarówno pesymistyczny jak i amortyzowany budżet zmian, gdzie definicja dla obu modeli jest taka sama. Pesymistycznym (worst case) budżetem zmian będzie tu wartość $\mathtt{B}_{\mathtt{wc}} = \max\{\mathtt{B}_j\}_{j=1}^{n}$, zaś amortyzowany budżet zmian zdefiniujemy jako $\mathtt{B}_{\mathtt{am}} = (\sum_{j=1}^{n} \mathtt{B}_j)/n$.

Klasyczny model online odpowiada modelowi inkrementalnemu gdzie $\mathtt{B}_{\mathtt{wc}} = \mathtt{B}_{\mathtt{am}} = 1$, a zatem jedyna możliwa akcja to nadanie nowemu przedziałowi (czy też w ogólności wierzchołkowi) koloru. W tym przypadku nie ma znaczenia czy rozpatrujemy scenariusz pesymistyczny czy też amortyzowany. W modelu online w klasie grafów ogólnych znane są bardzo niekorzystne ograniczenia dolne na liczbę potrzebnych kolorów. Oznacza to, że jeśli chcemy utrzymywać właściwe l-kolorowanie, to zadany parametr l musi być odpowiednio wysoki. Nawet w łatwiejszym dla algorytmu modelu losowym mamy ograniczenie dolne $l \geqslant (\frac{n}{\log^2 n} \cdot k)$ [87]. Co więcej, także w klasie lasów gdzie $k = 2$ mamy ograniczenie dolne rzędu $l \in \Omega(\log n)$ [16]. Te niekorzystne ograniczenia skierowały uwagę badaczy na klasy grafów, które nie zawierają lasów. Najbardziej naturalną wśród nich jest klasa grafów przedziałowych oraz jej podklasa grafów równoprzedziałowych. Dla tych klas możliwe jest utrzymywanie l-kolorowania dla l będącego wyłącznie funkcją k. Dokładniej, w klasie grafów przedziałowych $l = 3k - 2$ kolorów jest niezbędne ale też wystarcza [102]. W przypadku grafów równoprzedziałowych, istnieje algorytm utrzymujący $l = (2k - 1)$-kolorowanie oraz pokazano, że $l \geqslant 3k/2$ [38, 65]. Bardzo ciekawym pytaniem jest jak można ograniczyć parametr l w tych klasach grafów jeśli możemy sobie pozwolić na budżet zmian większy niż jeden na operację. Pytanie to jest tym bardziej ciekawe w świetle wyniku Barby *i innych* [13], którzy pokazują, że nawet dla bardzo prostej klasy lasów, jeśli nalegamy na (dk)-kolorowanie dla zadanego $d > 1$ w modelu w pełni dynamicznym, wówczas amortyzowany budżet zmian jest ograniczony z dołu przez wielomian względem n, mianowicie $\mathtt{B}_{\mathtt{am}} \geqslant dn^{1/(d-1)}$. Inaczej mówiąc, każda stała aproksymacja liczby chromatycznej w klasie lasów wymagać będzie wielomianowego względem n amortyzowanego budżetu zmian.

C.1. **Inkrementalne kolorowanie grafów przedziałowych z ograniczonym budżetem zmian.** W pracy [C1] pokazujemy, że dla grafów przedziałowych dolne ograniczenie $l \geqslant (3k - 2)$ kolorów z algorytmu online [102] da się zmniejszyć wprowadzając umiarkowany budżet zmian. Pierwszym zaproponowanym przez nas wynikiem jest następujące twierdzenie.



Rysunek 10. Ilustracja podziału na grupy w dowodzie Twierdzenia 26 dla $k = 5$.

Twierdzenie 25 (Twierdzenie 5 w [C1])**.** *Istnieje inkrementalny algorytm który na k-kolorowalnym grafie przedziałowym utrzymuje $2k$-kolorowanie przy amortyzowanym budżecie zmian $\mathtt{B_{am}} \in \mathcal{O}(\log n)$.*

Powyższe twierdzenie jest uogólnieniem algorytmu online zaproponowanego w [102]. Głównym motorem algorytmu online [102] jest utrzymywanie podziału grafu przedziałowego na k lasów ścieżek, gdzie k jest parametrem problemu oznaczającym liczbę chromatyczną ostatecznego grafu. Dodatkową prostą obserwacją z naszej strony jest że na lasie ścieżek łatwo utrzymywać 2-kolorowanie przy budżecie zmian rzędu $\mathtt{B_{am}} \in \mathcal{O}(\log n)$.

W obliczu 3-aproksymacji przy budżecie $\mathtt{B_{am}} = 1$ oraz 2-aproksymacji przy budżecie $\mathtt{B_{am}} \in \mathcal{O}(\log n)$ nasuwa się pytanie jaki budżet wystarcza, aby utrzymywać optymalne kolorowanie w modelu inkrementalnym. W pracy [C1] udzielamy odpowiedzi na to pytanie.

Twierdzenie 26 (Twierdzenie 10 w [C1])**.** *Istnieje inkrementalny algorytm, który na grafie przedziałowym k-kolorowalnym utrzymuje k-kolorowanie przy amortyzowanym budżecie zmian $\mathtt{B_{am}} \in \mathcal{O}(k \cdot k! \cdot \sqrt{n})$.*

Przedstawimy teraz po krótce techniki oraz pomysły prowadzące do tego wyniku. Przypuśćmy że udało nam się utrzymać kolorowanie do pewnego kroku m, czyli mamy właściwe k-kolorowanie c_m dla instancji $\mathcal{I}_m$. Algorytm inkrementalny dowodzący Twierdzenia 26 opiera się w pierwszej kolejności na podziale instancji $\mathcal{I}_m$ na $q = \mathcal{O}(\sqrt{m})$ rozłącznych grup $\mathcal{G}_1, \ldots, \mathcal{G}_q$, gdzie każda grupa liczy $\mathcal{O}(\sqrt{m})$ przedziałów. Aby to osiągnąć wybranych zostaje $p = q + 1 = \mathcal{O}(\sqrt{m})$ punktów $x_1, \ldots, x_p$ na prostej. Dla każdego punktu x_i, $i \in \{1, \ldots, p\}$, niech $\mathcal{S}_i$ będzie zbiorem przedziałów z $\mathcal{I}_m$ które zawierają x_i (przykład pokazany jest na Rysunku 10). Każdy zbiór $\mathcal{S}_i$ jest separatorem, czyli po usunięciu przedziałów z $\mathcal{S}_i$ instancja $\mathcal{I}_m$ rozpada się na na dwie rozłączne instancje: przedziały kończące się przed x_i oraz przedziały zaczynające się po x_i. Żaden przedział pierwszej instancji nie może przecinać przedziału przynależnego do drugiej. Grupa $\mathcal{G}_i$ to przedziały zaczynające się po x_i oraz kończące przed x_{i+1}. Przykładowy podział zilustrowany jest na Rysunku 10. Nie jest trudno znaleźć taki podział dla $\mathcal{I}_m$, łatwo go także utrzymywać przez kolejne m operacji wstawiania nowego przedziału (tworząc w razie potrzeby dodatkowe punkty x_i). Głównym ciężarem jest tu pokazanie algorytmu inkrementalnego, który podczas kolejnych m operacji wstawienia przedziału wykona łącznie $\mathcal{O}(k \cdot k! \cdot m\sqrt{m})$ przekolorowań. Przypuśćmy zatem, że do instancji $\mathcal{I}_{m+j-1}$ wstawiamy przedział I_{m+j} dla $j \leqslant m$.

Rozważmy najpierw przypadek, gdy istnieje punkt podziału x_i, $i \in \{1, \ldots, p\}$, taki że $x_i \in I_{m+j}$: w tym przypadku I_{m+j} zostaje elementem separatora $\mathcal{S}_i$. Ze względu na to że separatorów jest $\mathcal{O}(\sqrt{m})$, zaś każdy z nich liczy co najwyżej k elementów, taka sytuacja wydarzy się co najwyżej $\mathcal{O}(k\sqrt{m})$ razy. Zatem za każdym razem, gdy przypadek ten zachodzi, algorytm może sobie pozwolić na przekolorowanie całej instancji $\mathcal{I}_{m+j}$. Przypadek ten zatem nie stanowi istoty problemu.

Rozważmy więc odwrotny przypadek, czyli przypadek gdy $I_{m+j} \in \mathcal{G}_i$ dla $i \in \{1, \ldots, q\}$. Jeżeli grupę $\mathcal{G}_i$ wraz z nowym przedziałem I_{m+j} da się pokolorować k kolorami tak, aby zachować kolorowanie c_{m+j-1} na separatorach $\mathcal{S}_i$ oraz $\mathcal{S}_{i+1}$, wówczas algorytm może to zrobić. Budżet zmian będzie wtedy pesymistycznie ograniczony przez $\mathcal{O}(\sqrt{m})$, gdyż $|\mathcal{G}_i| \in \mathcal{O}(\sqrt{m})$ oraz wyłącznie przedziały z $\mathcal{G}_i$ otrzymają nowy kolor.

Najtrudniejszym przypadkiem jest zatem sytuacja gdy $I_{m+j} \in \mathcal{G}_i$, ale grupy $\mathcal{G}_i$ wraz z nowym przedziałem nie da się pokolorować k-kolorami tak, aby nie zaburzyć kolorowania c_{m+j-1} na $\mathcal{S}_i$ oraz $\mathcal{S}_{i+1}$. Aby poradzić sobie z tym przypadkiem zaproponowaliśmy algorytm, który przekolorowuje instancję w następujący sposób. Wszystkie przedziały $I \in \mathcal{I}_{m+j-1}$ które zaczynają się najpóźniej w punkcie x_i, a więc wszystkie przedziały I poprzedzające separator $\mathcal{S}_i$ oraz przedziały przynależące do $\mathcal{S}_i$ dostają ten sam kolor co poprzednio: $c_{m+j}(I) = c_{m+j-1}(I)$. Poczynając od przedziału z $\mathcal{G}_i$ o najmniejszym początku, algorytm kontynuuje kolorowanie zachłannie na $\mathcal{G}_i$ oraz $\mathcal{S}_{i+1}$, kończąc na przedziale z $\mathcal{S}_{i+1}$ o największym początku. Otrzymuje w ten sposób kolorowanie c_{m+j} na $\mathcal{G}_i \cup \mathcal{S}_{i+1}$. Oznacza to, że kolory przedziałów na $\mathcal{S}_{i+1}$ zostają zmienione przez algorytm. Zmiana ta odpowiada pewnej permutacji zbioru kolorów $\{1, \ldots, k\}$. Algorytm następnie przykłada tę permutację do kolorów pozostałych przedziałów, czyli tych których początki są większe niż x_{i+1}, otrzymując właściwe k-kolorowanie c_{m+j} na całej już instancji $\mathcal{I}_{m+j}$. Taki krok algorytmu wymaga niestety $\mathcal{O}(m)$ przekolorowań. Kluczową obserwacją jest tutaj jednak, że dla konkretnej grupy $\mathcal{G}_i$, mało jest operacji wstawienia do $\mathcal{G}_i$, które zmieniają kolorowanie na $\mathcal{S}_{i+1}$ przy jednoczesnym zachowaniu kolorowania na $\mathcal{S}_i$. Przy założeniu, że w trakcie tych operacji $\mathcal{S}_i$ oraz $\mathcal{S}_{i+1}$ nie zmieniają się, będzie takich operacji co najwyżej $k!$. Z faktu, że grup jest $\mathcal{O}(\sqrt{m})$, wynika zatem że algorytm wykona łącznie $\mathcal{O}(k! \cdot \sqrt{m})$ kroków gdzie następuje $\mathcal{O}(m)$ przekolorowań, co daje łącznie $\mathcal{O}(k! \cdot m\sqrt{m})$ przekolorowań. Dodatkowy czynnik $\mathcal{O}(k)$ pozwala uwzględnić to, że separatory $\mathcal{S}_i$ oraz $\mathcal{S}_{i+1}$ mogą się w międzyczasie zmieniać.

Oprócz wyników wymienionych w Twierdzeniu 26 oraz w Twierdzeniu 25 prezentujemy w [C1] także ograniczenie dolne dla inkrementalnego budżetu zmian przy optymalnym kolorowaniu.

Twierdzenie 27 (Wniosek 21 w [C1])**.** *Inkrementalne utrzymywanie k-kolorowania grafu przedziałowego o liczbie chromatycznej ograniczonej przez $k \in \mathcal{O}(\sqrt{n})$ wymaga amortyzowanego budżetu zmian rzędu $\Omega(k)$.*

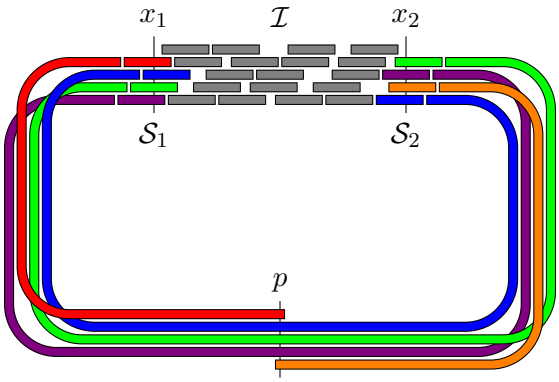
Innym ograniczeniem dolnym na inkrementalny budżet zmian przy optymalnym kolorowaniu jest ograniczenie rzędu $\Omega(\log n)$, które działa nawet w klasie grafów równoprzedziałowych.

Twierdzenie 28 (Twierdzenie 1 w [C1])**.** *Inkrementalne utrzymywanie optymalnego kolorowania na 2-kolorowalnym grafie równoprzedziałowym wymaga amortyzowanego budżetu zmian rzędu $\Omega(\log n)$.*

Powyższe ograniczenia dolne rozbiegają się znacznie z ograniczeniem górnym zaproponowanym w Twierdzeniu 26. Pytanie o to czy możliwe jest inkrementalne utrzymywanie optymalnego kolorowania na grafach przedziałowych przy amortyzowanym budżecie zmian rzędu $\mathcal{O}(f(k) \log n)$ pozostawiamy otwarte.

W pracy [C1] dla klasy grafów przedziałowych pokazujemy dodatkowo strukturę danych działającą w nieco innym modelu, gdzie kolorowanie utrzymywane jest niejawnie. Struktura ta nie ma zastosowania w modelu ograniczonego budżetu zmian. Przechowuje ona zbiór przedziałów, i umożliwia aktualizowanie tego zbioru poprzez dodanie nowego przedziału w czasie $\mathcal{O}(k^2 \log^3 n)$. Pozwala też odpytywać o kolor przedziału w czasie $\mathcal{O}(\log n)$, i gwarantuje, że odpowiedzi pomiędzy dwoma aktualizacjami będą zgodne z jakimś optymalnym kolorowaniem aktualnie przechowywanej instancji.

C.2. **Inkrementalne i w pełni dynamiczne kolorowanie grafów równoprzedziałowych z ograniczonym budżetem zmian.** W przeciwieństwie do grafów przedziałowych, dla klasy



RYSUNEK 11. Ilustracja do dowodu Twierdzenia 29 dla $k = 5$.

grafów równoprzedziałowych problem dynamicznego kolorowania z ograniczonym budżetem zmian udało się zamknąć z dokładnością do czynnika wielomianowego względem parametru k.

Pierwszy zaproponowany przez nas wynik dla dynamicznego kolorowania grafów równoprzedziałowych pochodzi z pracy [C1]. Prezentujemy algorytm inkrementalny, który utrzymuje $(k+1)$-kolorowanie i osiąga pesymistyczny budżet zmian $\mathcal{O}(k^2)$. Oznacza to, że w tej klasie grafów dając algorytmowi jeden nadmiarowy kolor jesteśmy w stanie dostać bez żadnej amortyzacji budżet zmian ograniczony wyłącznie funkcją liczby chromatycznej. W pracy [C1] przyjęliśmy i rozważaliśmy jedynie model inkrementalny. Jest jednak stosunkowo prostą obserwacją iż algorytm inkrementalny dowodzący tego twierdzenia uogólnia się do modelu w pełni dynamicznego.

Twierdzenie 29 (Twierdzenie 2 w [C1])**.** *Istnieje algorytm inkrementalny który utrzymuje $(k+1)$-kolorowanie k-kolorowalnego grafu równoprzedziałowego z pesymistycznym budżetem zmian $\mathcal{O}(k^2)$.*

Algorytm ten opiera się na naturalnej redukcji do problemu kolorowania grafów łukowych, czyli grafów przecięć łuków położonych na okręgu. Redukcja ta została wcześniej użyta w pracy [86] w kontekście algorytmów rozproszonych. Jest ona zilustrowana na Rysunku 11, opiszemy ją zaś poniżej nieco dokładniej. Po zastosowaniu tej redukcji na "małym fragmencie" danego grafu równoprzedziałowego skorzystamy z wyniku dla statycznego kolorowania grafów łukowych, przytoczonego pod postacią następującego lematu.

Lemat 30 (Twierdzenie 3.1 w Valencia-Pabon [147])**.** *Niech G będzie grafem łukowym, $\Lambda(G)$ niech oznacza maksymalną liczbę łuków przecinających się w jednym punkcie na okręgu, zaś $\lambda(G)$ niech oznacza minimalną liczbę łuków które pokrywają cały okrąg. Jeśli zachodzi $\lambda(G) \geqslant 5$, wówczas graf G jest $\lceil (1 + \frac{1}{\lambda(G)-2})\Lambda(G) \rceil$ - kolorowalny.*

Lemat 30 można nieco przeformułować, aby otrzymać następujący wniosek, będący podstawą naszego algorytmu.

Wniosek 31. *Niech G będzie grafem łukowym, $\Lambda(G)$ niech oznacza maksymalną liczbę łuków przecinających się w jednym punkcie na okręgu, zaś $\lambda(G)$ niech oznacza minimalną liczbę łuków które pokrywają cały okrąg. Jeżeli $\Lambda(G) = k$ oraz $\lambda(G) \geqslant k+2$, wówczas G jest $(k+1)$-kolorowalny.*

Przypuśćmy, że mamy daną reprezentację $\mathcal{I}_{j-1}$ grafu równoprzedziałowego, $(k+1)$-kolorowanie c_{j-1} zbioru $\mathcal{I}_{j-1}$ oraz że w j-tym kroku prezentowany jest algorytmowi nowy przedział I_j. Celem naszym jest przekolorowanie tylko fragmentu $\mathcal{I}_{j-1}$ o wielkości $\mathcal{O}(k^2)$ który zawiera I_j. Wybierzmy zatem punkty x_1 na lewo od I_j oraz x_2 na prawo od I_j w taki sposób, aby pomiędzy x_1 oraz x_2 znalazło się przynajmniej $(k+1)k$ i co najwyżej $(k+1)k+k$ przedziałów. Niech $\mathcal{S}_i$ będzie zbiorem przedziałów zawierających punkt x_i dla $i \in \{1, 2\}$ (jak pokazane na Rysunku 11), zaś

$\mathcal{I}$ niech będzie zbiorem przedziałów zaczynających się na prawo od x_1 i kończących na lewo od x_2. Załóżmy dla uproszczenia że przedziały $\mathcal{S}_1 \cup \mathcal{S}_2$ pokolorowane są kolorami ze zbioru $\{1, 2, \ldots, k\}$. Możemy wówczas przekształcić instancję $\mathcal{S}_1 \cup \mathcal{I} \cup \mathcal{S}_2$ w graf łukowy G który jest instancją statycznego problemu kolorowania grafów łukowych. Przedziały z $\mathcal{S}_1$ łączymy łukiem z przedziałami z $\mathcal{S}_2$ o tym samym kolorze, zaś jeśli przedział w $\mathcal{S}_1$ nie ma w $\mathcal{S}_2$ odpowiednika o tym samym kolorze, wówczas łączymy go łukiem z punktem p wybranym na okręgu poza przedziałami z $\mathcal{S}_1 \cup \mathcal{I} \cup \mathcal{S}_2$. Redukcję tą pokazuje Rysunek 11. Z faktu że $\mathcal{S}_1 \cup \mathcal{I} \cup \mathcal{S}_2$ jest k-kolorowalny oraz z faktu że $\mathcal{S}_1 \cup \mathcal{S}_2$ pokolorowane są k kolorami wynika, że dla tak skonstruowanego grafu G zachodzi $\Lambda(G) = k$ jak we Wniosku 31. Również ze względu na to, że $|\mathcal{I}| \geqslant (k+1)k$ oraz z faktu, że przedziały mają równą długość wynika że $\lambda(G) \geqslant k+2$. A zatem na mocy Wniosku 31 graf G da się pokolorować $(k+1)$-kolorami. Algorytm dokonuje tego kolorowania, a następnie permutuje kolory, aby pozostawić kolorowanie c_{j-1} niezmienione na $\mathcal{S}_1 \cup \mathcal{S}_2$. Wystarczy zatem, że algorytm przekoloruje dobrany fragment instancji $\mathcal{I}_{j-1}$ o wielkości $\mathcal{O}(k^2)$, aby uzyskać $(k+1)$-kolorowanie c_j dla instancji $\mathcal{I}_j$. Technicznym szczegółem, który jest zaadresowany w [C1] lecz który pominiemy w niniejszym opracowaniu, jest kwestia zagwarantowania że $\mathcal{S}_1 \cup \mathcal{S}_2$ pokolorowane są przez c_{j-1} kolorami ze zbioru $\{1, \ldots, k\}$.

Praca [C1] jako jeden z głównych problemów otwartych pozostawia kwestię budżetu zmian dla dokładnego kolorowania grafów równoprzedziałowych. W pracy [C2] zamykamy ten problem. W pierwszej kolejności pokazujemy, że w modelu w pełni dynamicznym, optymalny algorytm zmuszony jest zużyć liniowy amortyzowany budżet na operację.

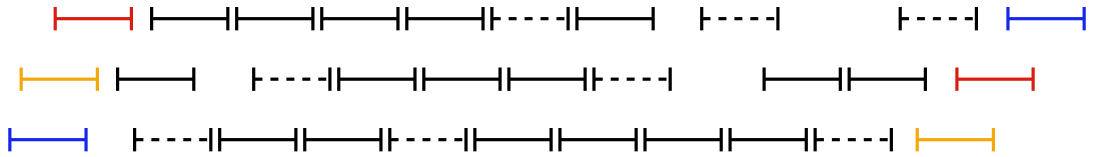
Twierdzenie 32 (Obserwacja 10 w [C2])**.** *Istnieje ciąg m aktualizacji generujący instancje 2-kolorowalnego grafu równoprzedziałowego, na którym dowolny w pełni dynamiczny algorytm utrzymujący 2-kolorowanie wykona łącznie $\Omega(m^2)$ przekolorowań.*

Model inkrementalny dla grafów równoprzedziałowych okazuje się znacznie ciekawszy. Tu pokazujemy algorytm z amortyzowanym budżetem zmian rzędu $\mathcal{O}(k^7 \log n)$ na operację. Jest to wykładnicza poprawa względem algorytmu z pracy [C1], zarówno ze względu na n jak i ze względu na k. Wynik ten pokazuje również kolosalną różnicę w trudności pomiędzy utrzymywaniem optymalnego kolorowania i kolorowania z jednym nadmiarowym kolorem.

Twierdzenie 33 (Twierdzenie 9 w [C2])**.** *Istnieje algorytm inkrementalny, który na inkrementalnie zmieniającym się k-kolorowalnym grafie równoprzedziałowym utrzymuje k-kolorowanie i którego łączny budżet zmian jest rzędu $\mathcal{O}(k^7 n \log n)$.*

W pracy [C2] całościowy argument dowodzący powyższego twierdzenia rozciąga się na 15 stronic, zatem w niniejszym opracowaniu przedstawimy tylko główne pomysły prowadzące do tego wyniku. Składa się on z kilku wyników pośrednich, z których każdy może znaleźć zastosowanie w problemach niezależnych od głównego problemu poruszanego w pracy czyli inkrementalnego kolorowania.

Pierwszym z tych pośrednich wyników jest ciekawy lemat dotyczący problemu *rozszerzania kolorowania częściowego* na grafach równoprzedziałowych. Problem rozszerzania kolorowania częściowego jest problemem statycznym, to znaczy cała instancja jest od początku dana na wejściu. W problemie tym na wejściu dany jest k-kolorowalny graf oraz właściwe k-kolorowanie dane na podzbiorze wierzchołków. Zadaniem jest stwierdzenie czy istnieje właściwe k-kolorowanie całego grafu które pokrywa się z kolorowaniem danym na podzbiorze wierzchołków. Problem ten był szeroko rozważany w klasach grafów, dla których problem kolorowania da się rozwiązać wielomianowo, a zatem w klasie grafów o stałej szerokości drzewiastej [25, 93], doskonałych [108], cięciwowych [93, 119, 118], przedziałowych [25], wreszcie równoprzedziałowych [120]. Ostatecznie pokazano, że gdy k jest stałe, wówczas istnieje wielomianowy algorytm rozszerzający kolorowanie dla klasy grafów o stałej szerokości drzewiastej, co implikuje wielomianowy algorytm również



Rysunek 12. Ilustracja treści Lematu 34 dla $k = 3$ oraz $k^2 - 1 = 8$.

dla klasy grafów równoprzedziałowych. Bez założenia że k jest stałe, nawet w klasie grafów równoprzedziałowych problem ten jest NP-trudny, nawet wtedy gdy kolorowanie dane jest tylko na pierwszych i ostatnich k przedziałach w porządku po pierwszej współrzędnej [120]. Nasz wynik związany z tym problemem ujawnia nietrywialną gamę instancji, dla których problem rozszerzania kolorowania na grafach równoprzedziałowych staje się łatwy.

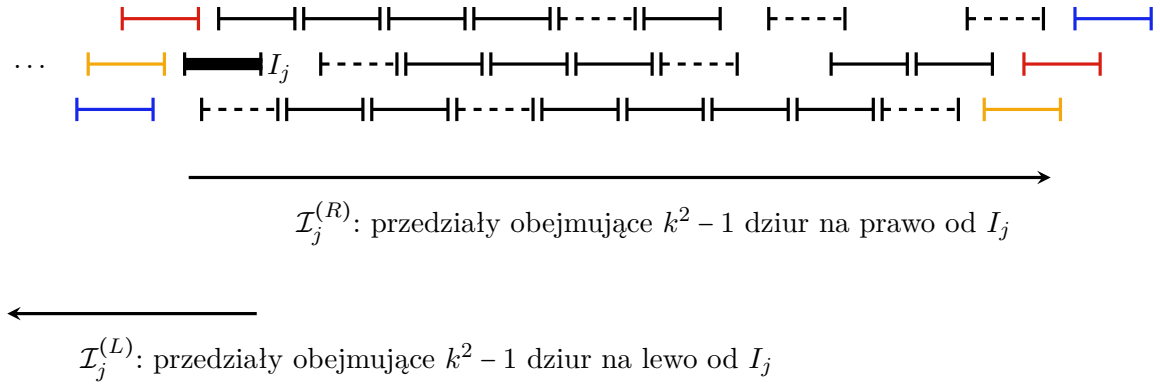
Lemat 34 (Lemat 5 w pracy [C2])**.** *Niech $\mathcal{J} = \{J_1, \ldots, J_n\}$ będzie k-kolorowalnym zbiorem przedziałów jednostkowych uporządkowanych po pierwszej współrzędnej i niech $n \geqslant 2k$. Niech $\mathcal{N}_k = \{J_1, \ldots, J_k\}$ oraz $\mathcal{M}_k = \{J_{n-k+1}, \ldots, J_n\}$. Niech $c' : \mathcal{N}_k \mapsto \{1, \ldots, k\}$ oraz $c'' : \mathcal{M}_k \mapsto \{1, \ldots, k\}$ będą odwzorowaniami bijektywnymi. Załóżmy też że istnieje zbiór $\mathcal{L}$ rozłącznych przedziałów jednostkowych taki że:*

- $|\mathcal{L}| = k^2 - 1$
- *dla każdego przedziału $J \in \mathcal{L}$ przedział J zaczyna się na prawo od końca J_k, zaś kończy się na lewo od początku J_{n-k+1}*
- *$\mathcal{J} \cup \mathcal{L}$ jest k-kolorowalny.*

Wówczas istnieje właściwe k-kolorowanie $c : \mathcal{J} \mapsto \{1, \ldots, k\}$ które pokrywa się z c' na $\mathcal{N}_k$ oraz pokrywa się z c'' na $\mathcal{M}_k$.

Intuicyjnie, powyższy lemat mówi, że jeśli k-kolorowalna reprezentacja $\mathcal{I}$ grafu równoprzedziałowego jest wystarczająco luźna, wówczas można dokolorować środek przy zadanym kolorowaniu na prefiksie długości k oraz sufiksie długości k. Luźność w tym przypadku oznacza, że pomiędzy prefiks i sufiks można dołożyć $k^2 - 1$ dodatkowych parami rozłącznych przedziałów nie zwiększając liczby chromatycznej. Te dodatkowe przedziały nazywać będziemy dziurami. Lemat 34 mówi zatem, że jeśli zbiór przedziałów jednostkowych ma przynajmniej $k^2 - 1$ dziur, wówczas da się uzupełnić kolorowanie dane na prefiksie i sufiksie o długości k. Przykład instancji o której mówi Lemat 34 pokazany jest na Rysunku 12 dla $k = 3$, gdzie przedziały należące do instancji zaznaczone są pełnymi liniami, zaś dodatkowe $k^2 - 1 = 8$ przedziałów (czyli dziury) zaznaczone jest liniami przerywanymi. Dowód Lematu 34 opiera się na tym, że kolorowanie c' jest kopiowane na kolejne przedziały oraz permutowane co jakiś czas (po każdym napotkaniu k dziur), tak aby na ostatnich k przedziałach dostać c''. Szczegóły tego dowodu pominiemy w niniejszym opracowaniu, aby skupić się na technice dowodzenia twierdzenia głównego.

Wróćmy zatem do modelu inkrementalnego, kiedy w momencie pojawienia się przedziału I_j mamy daną instancję $\mathcal{I}_{j-1}$ oraz jej właściwe k-kolorowanie c_{j-1}. Lemat 34 daje możliwość przekolorowania jedynie fragmentu $\mathcal{I}_{j-1}$ wokół I_j. Aby jednak na brzegach tego fragmentu mogło pozostać kolorowanie c_{j-1}, fragment ten powinien zawierać $k^2 - 1$ dziur. Rzeczywiście, pomijając pewne drobne kwestie techniczne, jedyne co proponowany przez nas algorytm robi to przekolorowanie takiego fragmentu. Dokładniej mówiąc, niech $I^{(p)}$ będzie poprzednikiem, zaś $I^{(s)}$ będzie następnikiem I_j w $\mathcal{I}_{j-1}$ w porządku po pierwszej współrzędnej. Następnie, zaczynając od $I^{(p)}$ i przetwarzając przedziały $\mathcal{I}_{j-1} \cup \{I_j\}$ w kolejności rosnących pierwszych współrzędnych algorytm znajdzie pierwszy przedział $I^{(q)}$ taki że pomiędzy $I^{(p)}$ i $I^{(q)}$ znajduje się $k^2 - 1$ dziur. Niech $\mathcal{I}_j^{(R)}$ będzie podzbiorem $\mathcal{I}_j$ przedziałów takich, że ich początek znajduje się pomiędzy początkiem $I^{(p)}$ a początkiem $I^{(q)}$ (jak pokazano na Rysunku 13). W następnej kolejności algorytm, zaczynając od $I^{(s)}$ i przetwarzając przedziały $\mathcal{I}_{j-1} \cup \{I_j\}$ w kolejności malejących



RYSUNEK 13. Ilustracja działania algorytmu inkrementalnego, przykład dla $k = 3$.

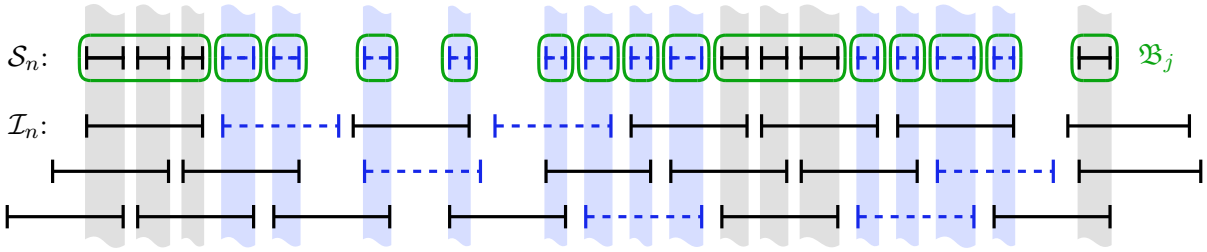
pierwszych współrzędnych, znajdzie pierwszy przedział $I^{(t)}$ taki że pomiędzy $I^{(t)}$ oraz $I^{(s)}$ znajduje się k^2-1 dziur. Niech $\mathcal{I}_j^{(L)}$ będzie podzbiorem $\mathcal{I}_j$ przedziałów takich, że ich początek znajduje się pomiędzy początkiem $I^{(t)}$ a początkiem $I^{(s)}$. Na mocy Lematu 34 dowolny z fragmentów $\mathcal{I}_j^{(L)}$ oraz $\mathcal{I}_j^{(R)}$ da się przekolorować nie zmieniając kolorowania c_{j-1} na $\mathcal{I}_j$ poza tym fragmentem. Algorytm przekolorowuje mniej liczny z fragmentów $\mathcal{I}_j^{(L)}$ oraz $\mathcal{I}_j^{(R)}$. Zatem budżet zmian algorytmu w kroku j wynosi $\mathtt{B}_j = \min\{|\mathcal{I}_j^{(L)}|, |\mathcal{I}_j^{(R)}|\}$. Fragmenty $\mathcal{I}_j^{(L)}$ oraz $\mathcal{I}_j^{(R)}$ rozpatrywane przez algorytm zilustrowane są na Rysunku 13.

Pozostaje pokazać że łączny budżet zmian opisanego wyżej algorytmu to rzeczywiście $\mathcal{O}(k^7 n \log n)$. Intuicyjnie, na cały proces dodawania przedziałów można spojrzeć jak na proces wypełniania dziur. Na początku, kiedy dziur jest dużo, fragmenty przekolorowywane przez algorytm będą małe. W miarę jak dziury będą wypełniane przedziałami, fragmenty przekolorowywane przez algorytm będą coraz liczniejsze, ale będzie to oznaczało że mało operacji dodania pozostało do wykonania. Jednak sformalizowanie tej intuicji, jak również dowód że łączny budżet zmian faktycznie wynosi $\mathcal{O}(k^7 n \log n)$, jest rzeczą dalece nietrywialną.

Jako narzędzie posłuży nam tu wprowadzona w pracy [C2] dwuosobowa gra, którą nazywamy *grą w żabkę*. Parametrami gry są początkowy rozmiar planszy N, oraz całkowitoliczbowe parametry $\kappa \geqslant 1$ i $\delta \geqslant 1$ które opiszemy nieco dalej. Planszą do gry w żabkę jest ciąg N liczb $R = \langle r_1, r_2, \ldots, r_N \rangle$ nazywanych też rangami. Początkowo wszystkie rangi są równe δ. Ciąg rang jest następnie modyfikowany przez gracza pierwszego nazywanego adwersarzem. Ruch adwersarza polega na wyborze dwóch sąsiadujących ze sobą rang r_i oraz r_{i+1}, usunięciu ich z ciągu, oraz wstawieniu na ich miejsce ich sumy $r_i + r_{i+1}$. A zatem plansza, która przed ruchem τ adwersarza miała postać $R_{\tau-1} = \langle r_1, r_2, \ldots, r_i, r_{i+1}, \ldots, r_N \rangle$ po ruchu przybiera postać $R_\tau = \langle r_1, r_2, \ldots, r_i + r_{i+1}, \ldots, r_N \rangle$. W tym momencie gracz drugi, którego nazwiemy żabką, ponosi pewien koszt ς_τ. Kosztem tym jest suma κ rang na prawo od r_{i+1} bądź suma κ rang na lewo od r_i, żabka zaś może wybrać dowolny z tych dwóch wariantów. Zadaniem żabki jest zminimalizować swój łączny koszt, adwersarz natomiast ma za zadanie, aby ten koszt był jak największy. W oczywisty sposób optymalną strategią żabki jest wybór w każdym kroku τ mniejszej sumy. Wtedy koszt żabki w ruchu τ wynosi $\varsigma_\tau = \min(\sum_{j=i-\kappa+1}^{i} r_j, \sum_{j=i+1}^{i+\kappa} r_j)$. Gra kończy się po $N-1$ ruchach. Łączny koszt żabki wynosi $\varsigma = \varsigma_1 + \ldots + \varsigma_{N-1}$. W pracy [C2] pokazujemy ograniczenie górne na łączny koszt żabki w zależności od parametrów N, κ oraz δ.

Twierdzenie 35 (Twierdzenie 8 w [C2])**.** *Przy optymalnej strategii łączny koszt żabki jest rzędu* $\varsigma \in \mathcal{O}(\kappa\delta N \log N)$ *przy każdej strategii adwersarza.*

Dowód powyższego twierdzenia jest dość żmudny i długi, dlatego pominiemy go w niniejszym opracowaniu. Przedstawimy teraz zarys dowodu Twierdzenia 33, pomijając kilka kwestii technicznych które ostatecznie utrudniają nieco analizę, nie są jednak istotne dla głównego pomysłu,

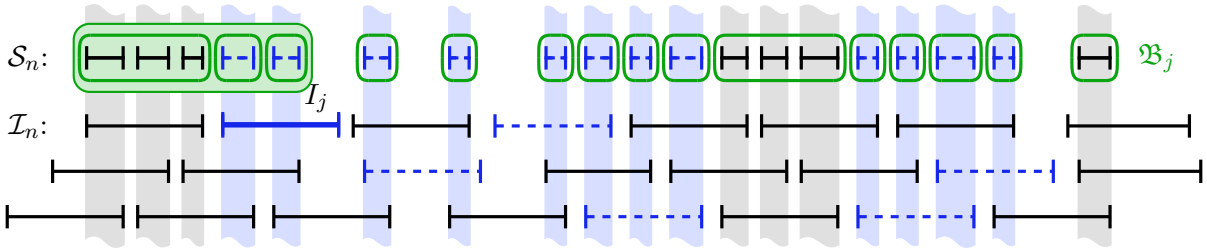


Rysunek 14. Ilustracja do dowodu Twierdzenia 33

który wykorzystujemy w dowodzie. W szczególności dla uproszczenia założymy że końcowa instancja $\mathcal{I}_n$ jest spójna i nie ma dziur, czyli że nie można w $\mathcal{I}_n$ wstawić dodatkowego przedziału pomiędzy pierwszy i ostatni w taki sposób aby powstała instancja wciąż była k-kolorowalna. Rozważymy teraz wszystkie k-kliki w końcowej instancji $\mathcal{I}_n$. Poprzez k-kilkę rozumiemy tu podzbiór k przedziałów które dzielą wspólny punkt. Każdą k-klikę w $\mathcal{I}_n$ reprezentuje przedział (niekoniecznie jednostkowy) punktów wspólnych dla wszystkich przedziałów kliki. Zbiór przedziałów reprezentujących k-kliki w końcowej instancji nazwiemy $\mathcal{S}_n$. Zbiór ten zilustrowany jest na Rysunku 14 ponad końcową instancją $\mathcal{I}_n$ (dla przykładu z rysunku zachodzi $k = 3$).

W chwili j, gdy pojawia się przedział I_j, daną mamy instancję $\mathcal{I}_{j-1}$, co oznacza że algorytm widzi tylko część przedziałów z ostatecznej instancji $\mathcal{I}_n$, pozostałe zaś przyszłe przedziały pojawią się w kolejnych krokach. Rysunek 14 przedstawia sytuację w chwili j w taki sposób, że przedziały z instancji $\mathcal{I}_{j-1}$ widoczne dla algorytmu w chwili j zaznaczone są czarną ciągłą linią, zaś przyszłe przedziały z $\mathcal{I}_n \setminus \mathcal{I}_{j-1}$ zaznaczone są niebieską przerywaną linią. Kliki z $\mathcal{S}_n$ w chwili j dzielą się na pełne i niepełne. Kilki pełne w chwili j to kliki zawarte w $\mathcal{I}_{j-1}$, czyli takie których wszystkie przedziały widoczne są dla algorytmu w chwili j. Kliki z $\mathcal{S}_n$ pełne w chwili j zaznaczone są na Rysunku 14 czarną ciągłą linią. Natomiast niepełne kliki z $\mathcal{S}_n$, czyli takie które zawierają przyszłe przedziały, zaznaczone są niebieską przerywaną linią. W dowodzie Twierdzenia 33 chcemy spojrzeć na proces dodawania przedziałów jak na proces wypełniania klik z $\mathcal{S}_n$. Na początku procesu wszystkie kilki są puste, w kolejnych zaś krokach wypełniane są coraz bardziej przez pojawiające się przedziały, wreszcie proces kończy się gdy wszystkie kliki są pełne. Poziom wypełnienia danej kliki $S \in \mathcal{S}_n$ w chwili j oznaczymy jako $\mathsf{lvl}_j(S) = |\{I \in \mathcal{I}_{j-1} : I \cap S \neq \varnothing\}|$ (jest to liczba przedziałów w klice S widocznych w chwili j). Dla każdej chwili j wprowadzimy też podział rodziny $\mathcal{S}_n$ na bloki. Podział ten będziemy nazywać $\mathfrak{B}_j$. W podziale tym sąsiadujące ze sobą pełne kliki umieszczone są w tym samym bloku podziału, natomiast każda niepełna klika stanowi odrębny blok, jak zaznaczono na Rysunku 14 kolorem zielonym. Bloki odpowiadające pojedynczym klikom niepełnym nazwiemy blokami niepełnymi, zaś bloki zawierające kliki pełne nazwiemy blokami pełnymi. Zauważmy że pełne kliki to takie kliki $S \in \mathcal{S}_n$ dla których $\mathsf{lvl}_j(S) = k$, zaś niepełne to takie, dla których $\mathsf{lvl}_j(S) < k$. W momencie dodania przedziału I_j, wszystkie kliki $S \in \mathcal{S}_n$ takie że $S \cap I_j \neq \varnothing$ zwiększają swój poziom $\mathsf{lvl}_j(S)$ o jeden. To oznacza, że w kroku j niektóre sąsiadujące ze sobą bloki z $\mathfrak{B}_j$ połączone zostaną w jeden. Na Rysunku 15 pokazane jest dodanie przedziału I_j oraz zaznaczone są bloki które na skutek tego dodania się połączą.

W tym momencie algorytm przekoloruje fragment $\mathcal{I}_j$ obejmujący $k^2 - 1$ dziur na lewo bądź na prawo od I_j, zaś budżet zmian wyniesie $B_j = \min\{|\mathcal{I}_j^{(L)}|, |\mathcal{I}_j^{(R)}|\}$. Główny pomysł, aby oszacować budżet B_j polega teraz na tym, aby użyć gry w żabkę, która opisuje zmiany powstałe w $\mathfrak{B}_j$ oraz da ograniczenie górne na budżet zmian. W tym celu kolejnym blokom z $\mathfrak{B}_j$ przypisujemy kolejne rangi w grze w żabkę. Przedziałami w bloku $\mathcal{B} \in \mathfrak{B}_j$ nazwiemy te przedziały $\mathcal{I}_j$ które wchodzą w skład jakiejś kliki w $\mathcal{B}$. Utrzymujemy niezmiennik, aby ranga r_i przypisana blokowi $\mathcal{B} \in \mathfrak{B}_j$ ograniczała z góry liczbę przedziałów w bloku $\mathcal{B}$. Łączenie dwóch kolejnych bloków podziału $\mathfrak{B}_j$ będzie odpowiadało ruchowi w grze w żabkę: dodamy wtedy sąsiednie rangi odpowiadające



Rysunek 15. Ilustracja do dowodu Twierdzenia 33

tym blokom. Zatem liczba przedziałów w nowo powstałym bloku będzie znów ograniczona przez jego rangę (z dokładnością do jednego przedziału I_j którego wcześniej nie było). Dobierzemy parametr κ tak, aby koszt żabki w tym kroku ograniczał z góry budżet algorytmu, to znaczy aby $\mathtt{B}_j \leqslant \varsigma_\tau$, gdzie krokowi j odpowiada ruch τ. Zastanówmy się ile bloków na prawo od I_j odwiedzi algorytm przekolorowując fragment $\mathcal{I}_j^{(R)}$. Rozważmy najpierw bloki niepełne. Tych algorytm odwiedzi co najwyżej k^3, jako że każdy z nich świadczy o istnieniu dziury, zaś co najwyżej k kolejnych bloków niepełnych świadczy o istnieniu tej samej dziury. Bloków pełnych algorytm odwiedzi mniej więcej tyle samo, ponieważ każdy z bloków pełnych bezpośrednio sąsiaduje z blokiem niepełnym. Oznacza to, że $\kappa = \mathcal{O}(k^3)$ wystarczy, aby koszt żabki szacował z góry budżet zmian. W rzeczywistości ze względu na liczne szczegóły techniczne pominięte w tym opisie, użyte przez nas κ będzie nieco większe, ale wciąż będzie wielomianową funkcją k. W szczególności każdemu blokowi w rzeczywistości przypisujemy kilka kolejnych rang, aby mieć zapas rang na wypadek, gdy dodanie przedziału nie połączy żadnych bloków. Ten i jemu podobne problemy sprawiają, że początkowa długość planszy N jest rzędu $\mathcal{O}(k^2 n)$, κ jest rzędu $\mathcal{O}(k^4)$ zaś $\delta = k$. Przy tak dobranych parametrach Twierdzenie 35 daje nam łączny koszt żabki rzędu $\mathcal{O}(k^7 n \log n)$. Łączny koszt żabki ogranicza natomiast z góry łączny budżet zmian algorytmu.

D. Dynamiczne algorytmy parametryzowane

Ostatni rozdział niniejszego opracowania poświęcony jest dynamicznym odpowiednikom znanych problemów parametryzowanych. Złożoność parametryzowana przykuła szerszą uwagę badaczy w latach osiemdziesiątych, kiedy to Downey i Fellows [58] wprowadzili teoretyczne podstawy tej dziedziny. Od tamtej pory cieszy się ona niesłabnącą popularnością, co znajduje odzwierciedlenie w zawrotnym tempie jej rozwoju. U podłoża tej dziedziny leży postrzeganie złożoności obliczeniowej nie tylko jako funkcji rozmiaru wejścia, ale także innych parametrów specyficznych dla danej instancji problemu. Dla problemów które w klasycznej teorii złożoności są NP-trudne, dodatkowy parametr pozwala lepiej zrozumieć gdzie tkwi istota trudności danego problemu. Słynna klasa FPT jest klasą problemów, które dają się rozwiązać w czasie wielomianowym względem rozmiaru instancji, gdzie od parametru zależy tylko współczynnik wielomianu nie zaś jego stopień. Innymi słowy, jeśli n to rozmiar wejścia zaś k to parametr, wówczas algorytmy FPT działają w czasie $\mathcal{O}(f(k) \cdot n^c)$ dla pewnej stałej c i funkcji f. Dziedzina złożoności parametryzowanej oferuje wiele ciekawych i nietrywialnych algorytmów FPT oraz wiele technik prowadzących do tych algorytmów. W niniejszym opracowaniu najbardziej będzie nas interesować technika kernelizacji oraz technika rozgałęziania.

Jak podają Cygan *i inni* [55], algorytm kernelizacyjny dla decyzyjnego problemu parametryzowanego dostaje na wejściu instancję problemu wraz z parametrem, a następnie w wielomianowym czasie oblicza równoważną instancję (jądro) tego samego problemu, której rozmiar ograniczony jest z góry funkcją parametru. Poprzez równoważność instancji rozumiemy tu iż odpowiedź jest

dla nich taka sama. Może nieco zaskakiwać, iż dla parametryzowanego problemu posiadanie algorytmu kernelizacyjnego równoważne jest z przynależnością do klasy FPT [55].

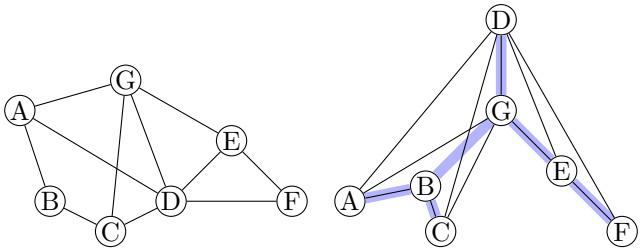
Technika rozgałęziania polega na konstruowaniu rozwiązania danej instancji za pomocą lokalnych decyzji które przyjmują formę drzewa rozgałęzień. Algorytm, patrząc na wejściową instancję, rozważa kilka możliwości jak lokalnie może wyglądać rozwiązanie. Dla każdej z tych możliwości algorytm redukuje instancję, a następnie rozgałęzia się rekurencyjnie względem zredukowanej instancji. Powstaje z tego procesu drzewo rozgałęzień. W typowych zastosowaniach zarówno stopień rozgałęzienia jak i głębokość drzewa można ograniczyć funkcją parametru, co w bezpośredni sposób daje algorytm FPT.

W stosunkowo nowej dziedzinie dynamicznych algorytmów FPT, celem jest utrzymywanie rozwiązania danego problemu FPT na dynamicznie zmieniającej się instancji dla tego problemu. Pojedyncza aktualizacja instancji zazwyczaj jest bardzo lokalną zmianą typu dodanie bądź usunięcie krawędzi w grafie, lub też zamiana litery jeśli wejściem dla problemu jest słowo. Najbardziej pożądany jest tu algorytm, który dokonuje pojedynczej aktualizacji w czasie zależnym wyłącznie od parametru. Za szybkie uznawane są również algorytmy dopuszczające czynniki polilogarytmicznie względem rozmiaru instancji, jednak jakościowo jest to już nieco inna klasa problemów. Jak zauważają Alman *i inni* [6], zarówno technikę kernelizacji, jak i technikę rozgałęziania daje się w przypadku wielu problemów zdynamizować. Dynamizacja ta polega na utrzymywaniu jądra bądź też drzewa rozgałęzień przy zmieniającej się dynamicznie instancji problemu. Zarówno jądro, jak i drzewo rozgałęzień są obiektami o rozmiarze ograniczonym funkcją parametru, co oznacza że można sobie pozwolić na całkowitą przebudowę tych obiektów po każdej aktualizacji i wciąż dostać efektywny algorytm.

Dla większości problemów rozważanych przez Almana *i innych* [6] parametrem problemu jest rozmiar rozwiązania. Dla takich problemów nie ma większego sensu wprowadzanie pojęcia budżetu zmian (można sobie bowiem pozwolić na całkowitą przebudowę rozwiązania). Jeśli jednak rozwiązaniem jest duży obiekt, jak na przykład drzewo rozpinające grafu, wówczas ma podstawy badanie ile zmian trzeba wprowadzić w rozwiązaniu po każdej aktualizacji, zwłaszcza jeżeli zależy nam na jawnym utrzymywaniu rozwiązania. Doskonałym przykładem jest tu dynamiczne utrzymywanie płytkiej dekompozycji drzewiastej badane przez nas w pracy [D1].

D.1. **Dynamiczna płytka dekompozycja drzewiasta z zastosowaniem do wykrywania długich ścieżek i cykli.** Głównym bohaterem pracy [D1] jest *płytka dekompozycja drzewiasta* grafu, dynamiczne jej utrzymywanie oraz zastosowanie do dynamicznych odpowiedników problemu k-ścieżki oraz k-cyklu. Zaczniemy więc od wprowadzenia głównego przedmiotu rozważań czyli płytkiej dekompozycji grafu. Płytka dekompozycja drzewiasta grafu $G = (V, E)$ jest lasem F ukorzenionych drzew. Zbiór wierzchołków lasu F jest tożsamy ze zbiorem V wierzchołków grafu G, zaś dla każdej krawędzi $uv \in E$ zachodzi zależność, że albo wierzchołek u jest przodkiem wierzchołka v w lesie F, albo tez wierzchołek v jest przodkiem wierzchołka u. Oznacza to w szczególności, że gdy G jest spójny, wówczas las F składa się z pojedynczego drzewa. Przykładowy graf (po lewej) oraz jego płytka dekompozycja drzewiasta (po prawej) pokazane są na Rysunku 16. Niech głębokość lasu F, oznaczana jako $\mathsf{depth}(F)$, będzie liczbą wierzchołków najdłuższej ścieżki pomiędzy korzeniem a liściem w F. *Głębokość drzewiasta* grafu G, oznaczana jako $\mathsf{td}(G)$, jest to najmniejsza możliwa głębokość płytkiej dekompozycji grafu G.

Głębokość drzewiasta jest ważnym parametrem z punktu widzenia złożoności parametryzowanej. Stanowi ona wariant centralnego parametru w złożoności parametryzowanej jakim jest *szerokość drzewiasta* [55]. Parametry te są dość ściśle powiązane, bowiem dla dowolnego grafu $G = (V, E)$ zachodzi $\mathsf{tw}(G) \leqslant \mathsf{td}(G) \leqslant \log|V| \cdot (\mathsf{tw}(G)+1)$, gdzie $\mathsf{tw}(G)$ oznacza szerokość drzewiastą grafu G. Oba te parametry powiązane są z dekompozycją grafu która przyjmuje postać drzewa (ew. lasu). W przypadku głębokości drzewiastej interesuje nas głębokość dekompozycji, zaś w



RYSUNEK 16. Przykładowy graf (po lewej) oraz jego płytka dekompozycja drzewiasta (po prawej). Krawędź BG jest krawędzią dekompozycji pomimo że nie jest krawędzią grafu.

przypadku szerokości drzewiastej mierzymy szerokość dekompozycji (definicję można znaleźć na przykład w [55]). Intuicyjnie parametry te zdefiniowane są tak, iż głębokość drzewiasta mierzy podobieństwo grafu do gwiazdy, podczas gry szerokość drzewiasta mierzy podobieństwo grafu do drzewa. Dość standardową techniką w dziedzinie złożoności parametryzowanej jest połączenie dekompozycji drzewiastej (płytkiej lub o małej szerokości) z programowaniem dynamicznym. Węzłom dekompozycji przypisywany jest zbiór stanów którego rozmiar ograniczony jest funkcją parametru (głębokości bądź też szerokości). Informacja zawarta w stanach dzieci jest wystarczająca aby efektywnie wyliczyć stany ich ojca. Dzięki tej technice wiele NP-trudnych problemów można rozwiązać algorytmem FPT ze względu na głębokość bądź też szerokość drzewiastą.

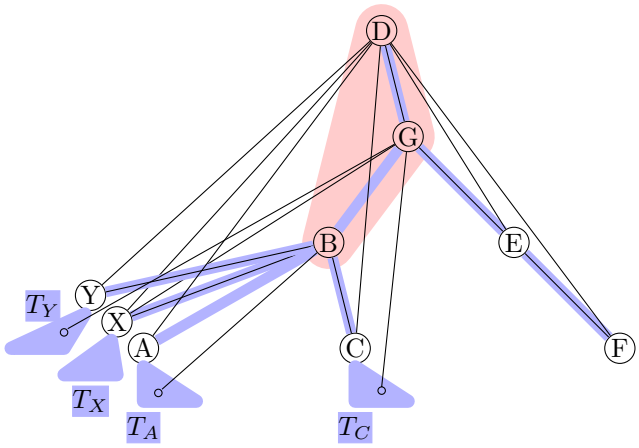
Standardowym narzędziem do sformalizowania tej techniki w jak największej ogólności są automaty drzewowe oraz logika na grafach. Dla szerokości drzewiastej najbardziej ogólna postać tej techniki przyjmuje oblicze twierdzenia Courcelle'a. Jest to bardzo głębokie i fundamentalne twierdzenie korzystające z formalizmu formuł logicznych MSO_2:

Twierdzenie 36 ([54, 55])**.** *Niech ϕ będzie formułą MSO_2, zaś G grafem na n wierzchołkach wraz z ewaluacją wszystkich wolnych zmiennych formuły ϕ. Ponadto załóżmy, ze mamy daną dekompozycję drzewiastą grafu G o szerokości t. Wówczas istnieje algorytm, który weryfikuje prawdziwość formuły ϕ dla grafu G w czasie $f(|\phi|, t) \cdot n$ dla pewnej obliczalnej funkcji f.*

Nie będziemy w niniejszym opracowaniu wprowadzać formalizmu MSO_2(wyczerpujący opis można znaleźć w książce [55]), warto jednak wspomnieć iż wyraża on wiele NP-trudnych problemów grafowych, np. problem 3-kolorowania czy też problem cyklu Hamiltona. Znane są również rozszerzone warianty Twierdzenia 36 pozwalające uwzględnić jeszcze większą gamę problemów.

Twierdzenie 36 prawdziwe jest również w wariancie dla głębokości drzewiastej w miejsce szerokości. W pracy [D1] podajemy szkic dowodu tego faktu, niemniej jednak fakt ten jest ogólnie znany w dziedzinie logiki i nie jest w żaden sposób zaskakujący. W świetle rosnącego zainteresowania algorytmami dynamicznymi dla problemów FPT dość naturalnym kierunkiem jest poszukiwanie efektywnego algorytmu dynamicznego, który jawnie utrzymywał będzie dekompozycję (płytką bądź o małej szerokości). Dostęp do dekompozycji pozwoli bowiem dynamicznie przeliczać stany programowania dynamicznego, co otwiera furtkę dynamizacji Twierdzenia 36 czy też jego odpowiednika dla głębokości drzewiastej. W pracy [D1] udało nam się to dla głębokości drzewiastej, która jest koncepcyjnie prostszym z dwóch omawianych parametrów (wcześniej udało się Dvořákowi *i innym* [60] ale w gorszej złożoności). Natomiast bardzo niedawno pojawiła się praca [106] która oferuje dynamiczny algorytm dla utrzymywania szerokości drzewiastej oraz dynamiczny wariant Twierdzenia 36.

Przejdźmy zatem do bardziej szczegółowego omówienia wyników z pracy [D1]. Podstawowym narzędziem które mamy do zaoferowania jest następujący algorytm dynamiczny dla utrzymywania płytkiej dekompozycji drzewiastej.



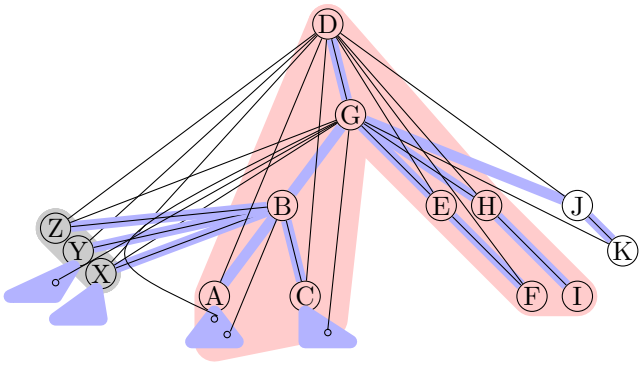
RYSUNEK 17. Przykład ilustrujący podstawowe definicje.

Twierdzenie 37 (Twierdzenie 1.1 w [D1])**.** *Niech d będzie dowolną liczbą naturalną. Niech G będzie grafem na n wierzchołkach dynamicznie modyfikowanym poprzez operacje dodania bądź usunięcia krawędzi. Istnieje struktura danych, która utrzymuje płytką dekompozycję drzewiastą o najmniejszej głębokości, pod warunkiem że $\mathsf{td}(G)$ pozostaje z góry ograniczone przez d. Wówczas struktura danych aktualizuje dekompozycję w pesymistycznym czasie $2^{\mathcal{O}(d^2)}$ na operację. Jeżeli jednak operacja powoduje że $\mathsf{td}(G) > d$, wówczas struktura danych wykrywa to i odmawia wstawienia krawędzi. Struktura ta zużywa $\mathcal{O}(d \cdot n)$ pamięci.*

Powyższe twierdzenie poprawia analogiczną strukturę danych zaproponowaną przez Dvořáka *i innych* [60], która oferuje identyczną funkcjonalność lecz z czasem $\mathcal{O}(f(d))$ na operację, gdzie f jest funkcją nie-elementarną. Wyjściowy pomysł na naszą strukturę danych wywodzi się z ogólnej techniki zaproponowanej przez Dvořáka *i innych*. Niemniej jednak poprawę daje nam głębsze zrozumienie kombinatoryki problemu płytkiej głębokości drzewiastej. Także operacje dodania i usunięcia krawędzi implementujemy zupełnie inaczej niż Dvořák *i inni*, co umożliwia nam uzyskanie czasu $2^{\mathcal{O}(d^2)}$ na operację. Warto tu wspomnieć, iż na drodze do jeszcze wydajniejszej struktury stoi pewna naturalna bariera. Konkretniej, najszybszy znany statyczny algorytm FPT obliczający optymalną płytką dekompozycję drzewiastą [133] działa w czasie $2^{\mathcal{O}(d^2)} \cdot m$, gdzie m to liczba krawędzi w grafie na wejściu. Oznacza to, że poprawienie czasu naszej struktury automatycznie dałoby lepszy algorytm dla statycznego problemu.

Przejdziemy teraz do szkicu dowodu Twierdzenia 37. Jako że dowód jest długi, miejscami żmudny oraz dość techniczny, zaprezentujemy jedynie kluczowe pomysły. Po pierwsze, chcemy pracować z dekompozycją która nie tylko jest optymalna globalnie, ale także lokalnie. Konkretniej, jeśli F jest ukorzenionym lasem zaś u jest wierzchołkiem lasu F, wówczas niech F_u oznacza poddrzewo F indukowane przez u oraz jego potomków w F. Niech także $\mathsf{desc}_F(u)$ oznacza zbiór potomków wierzchołka u w F wliczając u. Powiemy że płytka dekompozycja F jest *nadoptymalna* jeśli dla każdego wierzchołka u graf indukowany $G[\mathsf{desc}_F(u)]$ jest spójny, oraz $\mathsf{depth}(F_u) = \mathsf{td}(G[\mathsf{desc}_F(u)])$. Innymi słowy, w nadoptymalnej dekompozycji F, każde poddrzewo F_u jest optymalną dekompozycją dla $G[\mathsf{desc}_F(u)]$.

Aby pokazać kluczowe własności nadoptymalnych dekompozycji, ustalimy teraz graf G. Dla uproszczenia załóżmy że jest on spójny, a zatem jego nadoptymalna dekompozycja jest drzewem, nazwijmy ją więc T. Dla każdego wierzchołka u drzewa T zdefiniujemy jego *silnie osiągalny zbiór* $\mathsf{SReach}(u)$ jako $\mathsf{SReach}(u) = N_G(\mathsf{desc}_T(u))$, gdzie $N_G(X)$ oznacza otwarte sąsiedztwo w G zbioru $X \subseteq V$, a więc jest to zbiór wszystkich sąsiadów wierzchołków z X wyłączając wierzchołki z X. Przykładowo na Rysunku 17 zachodzi $\mathsf{SReach}(A) = \{B, D\}$ oraz $\mathsf{SReach}(C) = \mathsf{SReach}(X) = \mathsf{SReach}(Y) = \{B, D, G\}$. Intuicyjnie, silnie osiągalny zbiór dla wierzchołka u ma definiować



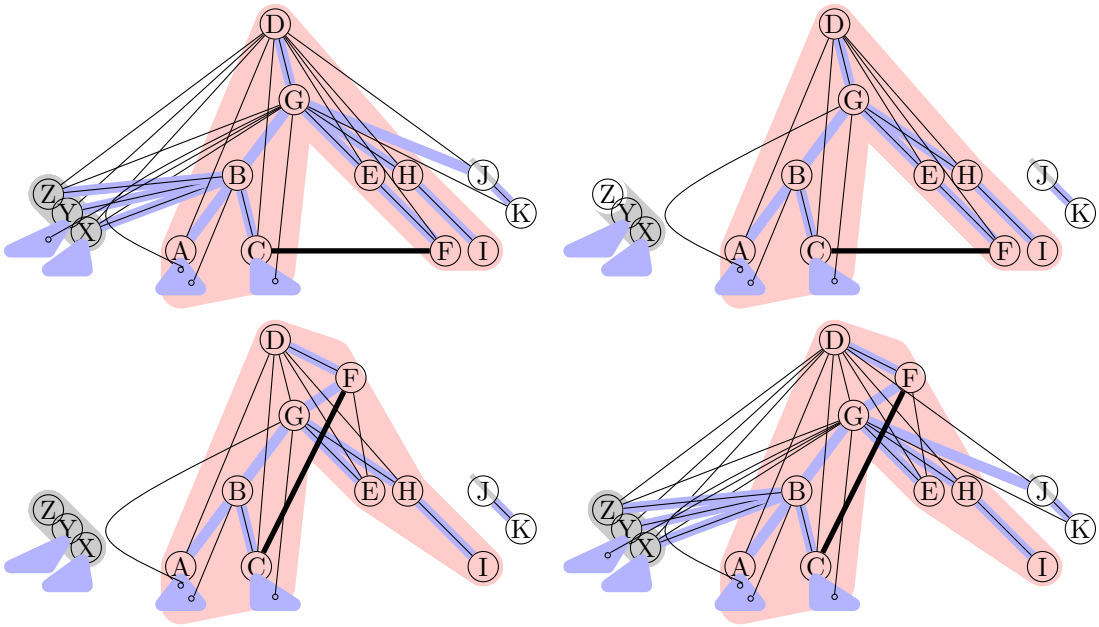
RYSUNEK 18. 2-rdzeń (na łososiowo) dla przykładowej płytkiej dekompozycji drzewiastej.

miejsce w dekompozycji w którym powinno się znajdować poddrzewo T_u: gdybyśmy usunęli T_u z T, a następnie chcieli je dodać z powrotem, wówczas należałoby uczynić u dzieckiem najgłębszego wierzchołka z $\mathsf{SReach}(u)$. Zauważmy, że dla dowolnego wierzchołka u zbiór $\mathsf{SReach}(u)$ układa się zawsze na ścieżce od u do korzenia w T. Ogólnie zbiór wierzchołków $X \subseteq V$ który leży na ścieżce od liścia do korzenia nazwiemy zbiorem *prostym*. Przykładowo na Rysunku 17 zbiór $\{B, D, G\}$ jest zbiorem prostym (zaznaczony na łososiowo). Rodzeństwem wierzchołka v w drzewie T nazwiemy dzieci ojca v wyłączając v. Przykładowo rodzeństwem wierzchołka C na Rysunku 17 są wierzchołki A, X oraz Y.

Zastanówmy się teraz, mając do dyspozycji nadoptymalną dekompozycję T oraz powyższe definicje, gdzie leży jądro problemu głębokości drzewiastej. Innymi słowy, chcemy wyodrębnić z G jak najmniejszy podgraf G', taki że $\mathsf{td}(G') = \mathsf{td}(G) = \mathsf{depth}(T) \leqslant d$. Przypuśćmy że wierzchołek u posiada w drzewie T przynajmniej k rodzeństwa o tym samym silnie osiągalnym zbiorze. Nazwijmy to rodzeństwo $u_1, \ldots, u_k$, gdzie $\mathsf{SReach}(u) = \mathsf{SReach}(u_i)$ dla $i \in \{1, \ldots k\}$. Można zaobserwować, że jeśli $k \geqslant d$, wówczas dowolna para wierzchołków $x, y \in \mathsf{SReach}(u)$ jest d-spójna, czyli istnieje przynajmniej $(d+1)$ wierzchołkowo rozłącznych ścieżek w G pomiędzy x oraz y. Oznacza to że po usunięciu T_u z grafu G, w każdej dekompozycji powstałego grafu o głębokości ograniczonej przez d zbiór $\mathsf{SReach}(u)$ musi być prosty. Ponadto, jeśli $\mathsf{depth}(T_u) \leqslant \mathsf{depth}(T_{u_i})$ dla $i \in \{1, \ldots, k\}$, wówczas po usunięciu T_u z grafu G głębokość drzewiasta powstałego grafu nie zmniejszy się względem głębokości G. Co więcej, możemy usunąć T_u z grafu G, policzyć w powstałym grafie nadoptymalną dekompozycję, i mamy gwarancję, że możemy odpowiednio wpiąć T_u do nowej dekompozycji nie zwiększając jej głębokości. Powyższa intuicja prowadzi do definicji rdzenia dekompozycji T grafu G, jednak wcześniej potrzebne nam jeszcze kilka definicji pomocniczych. Prefiksem K ukorzenionego drzewa T nazwiemy dowolny podzbiór wierzchołków T, taki że każdy wierzchołek należący do K ma swoich wszystkich przodków w K. Wierzchołek v nazwiemy *doczepem* prefiksu K, jeśli v nie należy do K zaś ojciec v należy do K. Zbiór doczepów prefiksu K oznaczymy jako $\mathsf{App}(K)$. Definicje te zobrazowane są na Rysunku 18, gdzie prefiks K zaznaczony jest kolorem łososiowym, zaś doczepy zaznaczone są na szaro. Zaznaczony na Rysunku 18 prefiks K jest jednocześnie 2-rdzeniem według poniższej definicji.

Definicja 38 (Definicja 2.2 w [D1])**.** *Niech $q \in \mathbb{N}$. Niepusty prefiks K dekompozycji T nazwiemy $q-$rdzeniem pary (G, T) jeżeli dla każdego $a \in \mathsf{App}(K)$ oraz każdego $X \subseteq \mathsf{SReach}(a)$ takiego że $|X| \geqslant 2$, wierzchołek a ma rodzeństwo składające się z przynajmniej q różnych wierzchołków $w \in K$, takich że $X \subseteq \mathsf{SReach}(w)$ oraz $\mathsf{depth}(T_w) \geqslant \mathsf{depth}(T_a)$.*

Nietrudną lecz niezwykle ważną własnością rdzeni jest następujący lemat, który gwarantuje istnienie małego rdzenia.



Rysunek 19. Kroki algorytmu dla operacji dodania krawędzi CF: obliczenie rdzenia K (lewo, góra), wyodrębnienie pografu $G[K]+CF$ (prawo, góra), przeliczenie dekompozycji na $G[K]+CF$ (lewo, dół) oraz podpięcie doczepów (prawo dół).

Lemat 39 (Lemat 2.1 w [D1])**.** *Dla dowolnej pary (G,T), gdzie T jest nadoptymalną dekompozycją grafu G o głębokości $\mathsf{depth}(T) \leqslant d$ oraz dla dowolnego $q \geqslant 2$ istnieje q-rdzeń K pary (G,T) o rozmiarze $(qd)^{\mathcal{O}(d)}$.*

Jak już argumentowaliśmy, d-rdzeń K pary (G,T), gdzie T jest nadoptymalną dekompozycją grafu G o głębokości $\mathsf{depth}(T) \leqslant d$, wyznacza jądro dla problemu głębokości drzewiastej. Jądrem tym jest $G[K]$, co oznacza, że $\mathsf{td}(G[K]) = \mathsf{td}(G)$. Co więcej, na mocy Lematu 39, jądro $G[K]$ jest wielkości $d^{\mathcal{O}(d)}$.

Przejdźmy teraz do aktualizacji grafu G poprzez dodanie krawędzi uv której nie ma w G (usunięcia obsłużymy analogicznie). Algorytm dynamiczny, w momencie aktualizacji, ma dostęp do nadoptymalnej płytkiej dekompozycji T grafu G która spełnia $\mathsf{depth}(T) \leqslant d$. Algorytm wyznacza $(d+2)$-rdzeń K dla (G,T), taki że $u,v \in K$. A zatem końce nowo dodawanej krawędzi uv należą do jądra $G[K]$. Następnie algorytm wylicza nadoptymalną dekompozycję T^K dla jądra powiększonego o nową krawędź, czyli dla grafu $G[K]+uv$. W tym celu algorytm korzysta ze statycznego algorytmu Reidla *i innych* [133], co daje czas na przeliczenie dekompozycji rzędu $2^{\mathcal{O}(d^2)}$. Wreszcie doczepy $w \in \mathsf{App}(K)$ zostają podczepione do T^K w miejsca wyznaczone przez $\mathsf{SReach}(w)$, gdzie $\mathsf{SReach}(w)$ pozostaje zbiorem prostym w T^K jak argumentowaliśmy wcześniej. Proces dodawania krawędzi zaprezentowany jest na Rysunku 19.

Jasnym jest, że pesymistyczny budżet zmian jest tu ograniczony przez rozmiar rdzenia czyli $d^{\mathcal{O}(d)}$. Znacznie mniej oczywiste jest jak zaimplementować ten algorytm aby cały proces aktualizacji zajął czas rzędu $2^{\mathcal{O}(d^2)}$. Implementacja opisana jest w Rozdziale 7 pracy [D1], jednak pominiemy ją w niniejszym opracowaniu. W Rozdziale 8 pracy [D1] wprowadzamy zaś język schematów konfiguracji, aby wzbogacić naszą strukturę danych o stany programowania dynamicznego. Ostatecznie daje nam to algorytm weryfikujący zadaną formułę ϕ w języku MSO_2na dynamicznie zmieniającym się grafie o głębokości drzewiastej co najwyżej d w czasie $\mathcal{O}(f(d,|\phi|))$ na aktualizację. Podobny wynik podali już wcześniej Dvořák *i inni* [60], jednak nie przykładali oni dużej wagi aby jak najlepiej wyznaczyć funkcję f. Dzięki dokładniejszej

analizie umożliwionej nam przez schematy konfiguracji, dla konkretnych problemów jesteśmy w stanie zazwyczaj podać lepsze oszacowanie czasu aktualizacji grafu niż wynikające z pracy Dvořáka *i innych*.

Jest tak w szczególności dla problemów rozważanych w dalszej części pracy, czyli dla problemu k-ścieżki oraz problemu k-cyklu. W problemie k-ścieżki dany jest graf G i parametr k, celem zaś jest stwierdzenie czyli w grafie G istnieje prosta ścieżka o długości k. W wyniku wzbogacenia naszej struktury danych z Twierdzenia 37 za pomocą schematów konfiguracji, dostajemy następujący dynamiczny algorytm dla problemu k-ścieżki w grafach o ograniczonej głębokości drzewiastej.

Lemat 40 (Lemat 8.7 w wersji arXiv pracy [D1])**.** *Istnieje struktura danych, która dla zadanego niezmiennego parametru k, aktualizuje n-wierzchołkowy graf G poprzez dodanie lub usunięcie pojedynczej krawędzi pod warunkiem że spełniony pozostaje warunek $\mathsf{td}(G) < k$. Struktura danych aktualizuje graf G w czasie $2^{\mathcal{O}(k^2)}$, oraz odpowiada na zapytanie czy w G jest k-ścieżka w czasie $\mathcal{O}(1)$. Jeżeli operacja wstawienia krawędzi powoduje że warunek $\mathsf{td}(G) < k$ przestaje być spełniony, wówczas struktura danych wykrywa to i odmawia wstawienia krawędzi. Struktura danych zajmuje $\mathcal{O}(n \cdot 2^{\mathcal{O}(k \log k)})$ pamięci.*

Minusem Lematu 40 jest fakt, że daje on efektywny algorytm dynamiczny dla problemu k-ścieżki tylko pod warunkiem, że głębokość drzewiasta grafu pozostaje ograniczona przez parametr k. Problem k-ścieżki okazuje się jednak inherentnie powiązany z głębokością drzewiastą grafu, co pokazuje następująca prosta i dobrze znana w dziedzinie złożoności parametryzowanej obserwacja.

Obserwacja 41. *Jeśli dla grafu G zachodzi $\mathsf{td}(G) \geqslant k$, wówczas w G istnieje k-ścieżka.*

Powyższa Obserwacja 41 w połączeniu z Lematem 40 pozwoli nam pozbyć się niewygodnego ograniczenia na głębokość drzewiastą. Obserwacja 41 i Lemat 40 są wynikami komplementarnymi, z których wynika że jeśli $\mathsf{td}(G) \geqslant k$ to w G na pewno istnieje k-ścieżka, zaś jeśli $\mathsf{td}(G) < k$, wówczas możemy użyć struktury danych Lematu 40 aby sprawdzić czy w grafie G istnieje k-ścieżka. Jedyny problem jaki pozostaje tu do rozwiązania, to jak efektywnie kontrolować kiedy warunek $\mathsf{td}(G) < k$ jest spełniony. Tu jednak przychodzi nam z pomocą dobrze znana w dziedzinie algorytmów dynamicznych technika leniwej obsługi aktualizacji łamiących niezmiennik, wprowadzona przez Eppstein *i innych* [64]. Ostatecznie mamy do zaoferowania następujący algorytm dynamiczny dla problemu k-ścieżki.

Twierdzenie 42 (Twierdzenie 1.2 w pracy [D1])**.** *Niech k będzie ustalonym parametrem. Niech G będzie dynamicznie zmieniającym się grafem na n wierzchołkach aktualizowanym poprzez wstawienie bądź usunięcie krawędzi. Istnieje struktura danych, która obsługuje aktualizacje na grafie G w amortyzowanym czasie $2^{\mathcal{O}(k^2)}$, oraz odpowiada na zapytanie czy w G jest k-ścieżka w czasie $\mathcal{O}(1)$. Struktura danych zajmuje $\mathcal{O}(n \cdot 2^{\mathcal{O}(k \log k)})$ pamięci. Struktura używa tablic haszujących aby uzyskać dostęp do krawędzi grafu, stąd czas na aktualizację jest czasem oczekiwanym.*

Warto zauważyć że w powyższym Twierdzeniu 42 nie pojawia się głębokość drzewiasta. Jedynym parametrem algorytmu dynamicznego jest parametr k oznaczający długość poszukiwanej k-ścieżki. Twierdzenie 42 poprawia dynamiczny algorytm dla k-ścieżki z pracy Almana *i innych* [6], który posiada czynniki logarytmiczne względem rozmiaru grafu.

Przejdziemy teraz do problemu k-cyklu. W odróżnieniu od problemu k-ścieżki, problem k-cyklu postawić można na dwa różne sposoby. W problemie dokładnie k-cyklu mamy dany graf G oraz parametr k i pytamy czy w grafie G istnieje cykl prosty o długości dokładnie k. Dla problemu dokładnie k-cyklu istnieje ograniczenie dolne w postaci $\Omega(n^\epsilon)$ na czas aktualizacji grafu przez algorytm dynamiczny (Twierdzenie 12.3 w pracy [D1]). Dlatego też w problemie k-cyklu rozważanym w pracy [D1] mamy na wejściu graf G oraz parametr k i pytamy czy w

grafie G istnieje cykl prosty o długości przynajmniej k. Dla tak postawionego problemu zachodzi następujący lemat.

Lemat 43 (Propozycja 6.2 w [129])**.** *Jeśli graf G jest dwuspójny oraz zachodzi $\mathtt{td}(G) \geqslant k^2$, wówczas w G istnieje cykl prosty na przynajmniej k wierzchołkach.*

Lemat 43 jest w pewnym sensie odpowiednikiem Obserwacji 41 dla problemu k-ścieżki, jednakże założenie dwuspójności czyni go dużo trudniejszym do zastosowania. Lemat 43 wymusza konieczność dynamicznego utrzymywania dwuspójnych składowych zmieniającego się grafu oraz ich dekompozycji. Jest to najbardziej skomplikowana technicznie część pracy [D1], ponieważ pod wpływem aktualizacji grafu dwuspójne składowe mogą się łączyć bądź też rozdzielać. Wymaga to dodania do struktury danych z Twierdzenia 37 funkcjonalności łączenia i dzielenia dekompozycji. Pojawia się wtedy szereg nowych problemów ze wzbogaceniem struktury o stany programowania dynamicznego oraz z kontrolowaniem które składowe łączyć bądź dzielić. Ostatecznie oferujemy następujący dynamiczny algorytm dla problemu k-cyklu.

Twierdzenie 44 (Twierdzenie 1.2 w pracy [D1])**.** *Niech k będzie ustalonym parametrem. Niech G będzie dynamicznie zmieniającym się grafem na n wierzchołkach, aktualizowanym poprzez wstawienie bądź usunięcie krawędzi. Istnieje struktura danych, która obsługuje aktualizacje na grafie G w amortyzowanym czasie $2^{\mathcal{O}(k^4)} + \mathcal{O}(k \log n)$, oraz odpowiada na zapytanie czy w G jest cykl długości przynajmniej k w czasie $\mathcal{O}(1)$. Struktura danych zajmuje $\mathcal{O}(n \cdot 2^{\mathcal{O}(k^2 \log k)})$ pamięci. Struktura używa tablic haszujących aby uzyskać dostęp do krawędzi grafu, stąd czas na aktualizację jest czasem oczekiwanym.*

Pokazujemy również, że w dynamicznym algorytmie dla problemu k-cyklu nie da się uniknąć w czasie aktualizacji czynnika logarytmicznego względem rozmiaru instancji (Wniosek 12.5 w pracy [D1]). Na koniec warto nadmienić, iż ograniczenie z Lematu 43 zostało bardzo niedawno poprawione do liniowego [40], co automatycznie daje poprawione czasy w twierdzeniu powyżej.

D.2. **Dynamiczne parametryzowane algorytmy tekstowe.** W pracy [D2] rozważamy klasyczne parametryzowane problemy tekstowe w dynamicznym modelu Almana *i innych* [6]. Konkretniej, dla wszystkich rozważanych tu problemów instancją jest lista słów $w_1, \ldots, w_n$ nad zadanym alfabetem Σ oraz całkowitoliczbowy parametr p. Instancja jest aktualizowana w czasie. Aktualizacją jest tu zmiana pojedynczej litery w wybranym słowie. Celem jest wydajna struktura danych obsługująca następujące operacje:

- $\mathtt{init}(w_1, \ldots, w_n, p)$ - inicjalizacja struktury danych dla instancji $w_1, \ldots, w_n$ i parametru p
- $\mathtt{update}(i, j, a)$ - aktualizacja zmieniająca j-tą literę i-tego słowa na literę a, zdefiniowana dla $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, |w_i|\}$, $a \in \Sigma$
- $\mathtt{query}()$ - zapytanie czy dla danej instancji istnieje rozwiązanie problemu

W pracy [D2] proponujemy strukturę jak wyżej dla trzech klasycznych parametryzowanych problemów tekstowych. Pierwszym z rozważanych przez nas problemów jest Problem Najbliższego Słowa. W problemie tym wszystkie słowa na wejściu mają tę samą długość równą l. Dla zgodności z oznaczeniem w pracy oraz ze znaczeniem problemu parametr p będziemy tu oznaczać literą d (jak z ang. *distance*). Pytamy o to, czy istnieje słowo w nad alfabetem Σ, dla którego odległość Hamminga do każdego ze słów na wejściu ograniczona jest przez d. Przez odległość Hamminga $\mathtt{hm}(u, w)$ pomiędzy parą słów u i w o tej samej długości rozumiemy liczbę pozycji na których słowa u i w różnią się od siebie literą. Interesuje nas zatem istnienie słowa w takiego że dla każdego $i \in \{1, \ldots, n\}$ zachodzi $\mathtt{hm}(w, w_i) \leqslant d$. Problem najbliższego słowa jest flagowym problemem w dziedzinie złożoności parametryzowanej i jako taki został dogłębnie zbadany dla wszystkich naturalnych dla niego parametrów: n, d, l oraz $|\Sigma|$. Dla parametryzacji przez d oraz Σ, Gramm *i inni* [78] podali algorytm z czasem działania $d^{\mathcal{O}(d)} \cdot (nl)^{\mathcal{O}(1)}$, zaś Ma i Sun [115] podali

algorytm z czasem działania $|\Sigma|^{\mathcal{O}(d)} \cdot (nl)^{\mathcal{O}(1)}$. Algorytmy te stały się książkowymi przykładami dla techniki rozgałęziania [55], a ich czasy działania i zależność względem d są optymalne przy założeniu hipotezy ETH [114].

W algorytmie Gramm *i innych* [78] z czasem działania $d^{\mathcal{O}(d)} \cdot (nl)^{\mathcal{O}(1)}$ rozmiar drzewa rozgałęzień ograniczony jest przez $(3d)^d$, co oznacza że można pozwolić sobie na przeliczenie całego drzewa rozgałęzień dla każdej aktualizacji instancji. Nie jest to jednak łatwe, ponieważ znalezienie pojedynczego rozgałęzienia zajmuje czas $\mathcal{O}(nl)$. Naszym celem jest zatem struktura danych, która efektywnie (czyli w czasie zależnym wyłącznie od $|\Sigma|$ oraz d) wyznaczy pojedyncze rozgałęzienia.

Aby lepiej przybliżyć istotę problemu, opiszemy najpierw statyczny algorytm rozgałęziający dla problemu najbliższego słowa, oparty na algorytmie z pracy [78]. Algorytm ten w pierwszej fazie sprawdza czy istnieją dwa słowa w_i oraz w_j takie że $\mathtt{hm}(w_i, w_j) > 2d$. Jeśli istnieją, wówczas algorytm kończy działanie i odpowiada, że rozwiązanie nie istnieje. Jeśli nie istnieją algorytm przechodzi do fazy drugiej, czyli fazy rozgałęziania. Algorytm startuje od dowolnego z wejściowych słów, i stopniowo modyfikuje je aby uzyskać rozwiązanie. Niech modyfikowane słowo nazywa się q. Początkowo $q = w_1$. Algorytm szuka słowa w_i, takiego że $\mathtt{hm}(q, w_i) > d$ (jeśli takiego nie ma to q jest rozwiązaniem). Następnie algorytm wyznacza listę pozycji P gdzie słowo w_i różni się literą od słowa q. Jeśli rozwiązanie istnieje, i wszystkie dotychczasowe modyfikacje q były zgodne z rozwiązaniem, to z pewnością na jednej z pozycji w P jest w tym rozwiązaniu litera taka jak w w_i. Algorytm sprawdza zatem każdą z pozycji: dla każdego $j \in P$ algorytm podmienia $q[j]$ na $w_i[j]$ i wywołuje się rekurencyjnie na tak zmodyfikowanym słowie q. Algorytm jednocześnie pilnuje, aby głębokość rekurencji nie przekroczyła d. Nietrudno pokazać że wówczas dla każdego wywołania rekurencyjnego zachodzi $|P| \leqslant 3d$. Jako że głębokość rekurencji jest ograniczona przez d, zaś liczba rozgałęzień przy pojedynczym wywołaniu ograniczona jest przez $3d$, stąd drzewo rozgałęzień ma rozmiar co najwyżej $(3d)^d$.

Zauważmy, że pierwsza faza, w której szukamy dwóch słów w odległości Hamminga większej niż $2d$, zaimplementowana wprost zajmuje czas $\mathcal{O}(n^2 l)$. Zauważmy także, iż każde wywołanie rekurencyjne, gdzie szukamy słowa w odległości Hamminga od q ograniczonej z dołu przez d, wymaga czasu $\mathcal{O}(nl)$. Wydajny algorytm dynamiczny nie może sobie pozwolić na takie czasy. Aby rozwiązać ten problem, oferujemy następującą strukturę danych, wykrywającą dalekie słowa w zmieniającej się w czasie instancji Problemu Najbliższego Słowa .

Lemat 45 (Lemat 6 w [D2]: struktura danych dla odległych słów)**.** *Istnieje randomizowana struktura danych, która utrzymuje słownik $\mathcal{S}$ zawierający n słów o długości l nad alfabetem Σ, która inicjalizuje się w czasie $2^{\mathcal{O}(d)} nl|\Sigma|$, oraz pozwala na aktualizacje w postaci zmiany pojedynczej litery w słowie z $\mathcal{S}$ w amortyzowanym czasie $2^{\mathcal{O}(d)}$. Struktura danych udostępnia zapytanie:*

- `QueryFarPair()`*: Czy istnieją słowa $s, s' \in \mathcal{S}$ takie że $\mathtt{hm}(s, s') > 2d$. Sturktura może błędnie odpowiedzieć tak, ale tylko wtedy gdy dla instancji $(\mathcal{S}, d)$ nie istnieje rozwiązanie.*

Ponadto, struktura danych umożliwia dostęp do utrzymywanego przez nią słowa $q \in \Sigma^l$ poprzez następujące metody:

- `Reset()`*: Przypisz na q pierwsze słowo w $\mathcal{S}$.*
- `UpdateCandidate`(i, a)*: Ustaw i-tą literę q na a.*
- `QueryFarWord()`*: Stwierdź czy istnieje słowo $s \in \mathcal{S}$ takie że $\mathtt{hm}(s, q) > d$, a jeśli tak, wówczas zwróć wskaźnik na s oraz zbiór pozycji na których s różni się od q.*

Metody `QueryFarPair()`*,* `Reset()`*,* `UpdateCandidate()`*,* `QueryFarWord()` *działają w pesymistycznym czasie $2^{\mathcal{O}(d)}$.*

Struktura danych z Lematu 45 udostępnia wszystkie metody potrzebne do zaimplementowania algorytmu rozgałęziającego opisanego wyżej. Dzięki temu otrzymujemy następujący wynik.

Twierdzenie 46 (Twierdzenie 5 w pracy [D2]). *Istnieje randomizowana struktura danych, która utrzymuje dynamicznie instancję* Problemu Najbliższego Słowa *modyfikowaną poprzez zmianę pojedynczej litery. Struktura inicjalizuje się w czasie $2^{\mathcal{O}(d)} \cdot nl$, dokonuje aktualizacji w amortyzowanym czasie $2^{\mathcal{O}(d)}$, oraz w czasie $d^{\mathcal{O}(d)}$ odpowiada na pytanie czy istnieje dla obecnej instancji rozwiązanie.*

W Twierdzeniu 46 można zapytanie zaimplementować także w czasie $|\Sigma|^{\mathcal{O}(d)}$, co również pokazujemy w pracy [D2]. Kluczem jednak do obu tych wyników jest struktura danych z Lematu 45. Opiszemy teraz zatem główne pomysły potrzebne aby strukturę tę uzyskać. Dla klarowności opisu przyjmiemy w niniejszym opracowaniu że pracujemy nad binarnym alfabetem $\Sigma = \{0, 1\}$, w pracy jednak pokazujemy jak za pomocą haszowania pozbyć się tego założenia. Główny pomysł polega na tym, że struktura danych wewnętrznie utrzymuje słowo $o \in \{0, 1\}^l$, które jest dobrym przybliżeniem rozwiązania jeśli tylko rozwiązanie istnieje. Uściślając, jeśli istnieje słowo którego odległość Hamminga do wszystkich słów z $\mathcal{S}$ jest ograniczona przez d, wówczas odległość Hamminga pomiędzy o a dowolnym słowem z $\mathcal{S}$ jest ograniczona przez $8d$. Okazuje się że słowo o jest takim przybliżeniem, jeśli tylko na każdej pozycji zgadza się ono z przynajmniej $\frac{n}{4}$ słowami z $\mathcal{S}$. Intuicyjnie, słowo o jest wybierane w wyniku pewnego rodzaju głosowania: litera $o[i]$ na pozycji $i \in \{1, \ldots, l\}$ jest wybierana jeśli wystarczająco dużo słów się z nią zgadza. Tak zdefiniowane słowo o łatwo jest efektywnie utrzymywać dynamicznie jeśli pozwolimy na amortyzację. Jeśli aktualnie wybrana litera $o[i] \in \{0, 1\}$ przestaje spełniać warunek, wówczas w czasie liniowym względem n wybierany jest nowy lider jako litera która najczęściej występuje na pozycji i. Nowy lider występuje przynajmniej $\frac{n}{2}$ razy w słowach z $\mathcal{S}$ na pozycji i, a zatem potrzeba $\frac{n}{4}$ modyfikacji zanim przestanie on spełniać warunek. Stąd amortyzowany czas aktualizacji słowa o wyniesie $\mathcal{O}(1)$. W tym samym czasie łatwo jest też dla każdego słowa z $\mathcal{S}$ utrzymywać listę pozycji, na których słowo różni się od o.

Opiszemy teraz jak wykorzystać o aby szybko znaleźć parę słów odległych o ponad $2d$ (czyli zaimplementować metodę `QueryFarPair()` z Lematu 45). Rozważmy dwa słowa $s_1, s_2 \in \mathcal{S}$. Niech X_1 będzie zbiorem pozycji, na których s_1 różni się od o, zaś X_2 niech analogicznie będzie zbiorem pozycji na których s_2 różni się od o (możemy założyć że $|X_1|, |X_2| \leqslant 8d$, inaczej rozwiązanie nie istnieje). Kluczowa obserwacja polega na tym, że $\texttt{hm}(s_1, s_2) > 2d$ wtedy i tylko wtedy gdy $|X_1 \Delta X_2| > 2d$, gdzie Δ oznacza różnicę symetryczną zbiorów. Wynika to z tego, że na pozycjach ze zbioru $X_1 \cap X_2$ oba słowa różnią się od o, a zatem przy założeniu binarnego alfabetu słowa te są sobie równe na tych pozycjach. Zatem pozostaje nam sprawdzić czy istnieją dwa słowa $s_1, s_2 \in \mathcal{S}$ takie że $|X_1 \Delta X_2| > 2d$. Tu z pomocą przychodzi nam ciekawe zastosowanie bardzo znanej techniki kodowania kolorami [55]. Zauważmy, że interesuje nas konkretne $16d$ pozycji na których jedno z dwóch szukanych słów różni się od o. Pokolorujemy zatem pozycje losowo na $16d$ kolorów, i jak nakazuje metoda kodowania kolorami będziemy liczyć na to, że interesujące nas pozycje otrzymają różne kolory. Niech $\texttt{col}(s)$ oznacza kolory pozycji, na których s różni się od o. Dla każdego podzbioru $C \subseteq \{1, \ldots, 16d\}$ będziemy utrzymywać zbiór słów $\Phi(C) = \{s \in \mathcal{S} : \texttt{col}(s) = C\}$. Są to słowa które różnią się od o na pozycjach o kolorach w C. Nie jest to trudne do utrzymania w czasie $2^{\mathcal{O}(d)}$ na operację, jeśli dla każdego słowa utrzymujemy już listę pozycji na których różni się ono od o. Aby znaleźć szukane słowa, iterujemy teraz po $X_1, X_2 \subseteq \{1, \ldots, 16d\}$ takich że $|X_1 \Delta X_2| > 2d$, i jeśli $\Phi(X_1)$ oraz $\Phi(X_2)$ są niepuste, wówczas na pewno znaleźliśmy szukaną parę słów. Możemy przeoczyć taką parę, jeśli interesujące nas pozycje nie są różnokolorowe, ale metoda kodowania kolorami daje narzędzia aby ograniczyć prawdopodobieństwo takiego zdarzenia. Pozostałe metody z Lematu 45 opierają się na podobnych argumentach.

Przejdziemy teraz do pozostałych problemów tekstowych badanych w pracy [D2]. Punktem wyjściowym jest tu meta-twierdzenie które zastosować można do szerokiej gamy dynamicznych problemów tekstowych.

Twierdzenie 47 (Twierdzenie 2 w pracy [D2]). *Niech Σ będzie skończonym alfabetem zaś $\mathcal{L} \subseteq \Sigma^\star$ niech będzie językiem definiowalnym w logice $FO[\Sigma, <]$. Wówczas istnieje struktura danych która dla słowa $w \in \Sigma^*$, aktualizowanego poprzez operację zmiany pojedynczej litery, utrzymuje odpowiedź na pytanie czy $w \in \mathcal{L}$. Struktura danych inicjalizuje się w czasie $\mathcal{O}(|w|)$, każde zaś zapytanie zajmuje pesymistyczny czas $\mathcal{O}(\log\log|w|)$.*

Twierdzenie 47 wynika wprost z połączenia wyniku Frandsena *i innych* [70] o dynamicznym problemie tekstowym dla nieokresowych półgrup z twierdzeniem Schützenbergera, McNaughtona, i Paperta [124, 138]. Nowym spostrzeżeniem jest to, że Twierdzenie 47 pozwala uzyskać wydajne dynamiczne struktury danych dla parametryzowanych problemów tekstowych. Pokazujemy to na kolejnych dwóch przykładach znanych parametryzowanych problemów tekstowych, którymi są:

(1) Problem Rozłącznych Podsłów Dla danego słowa $w \in \{1, \ldots, k\}^*$, gdzie k jest parametrem, rozstrzygnąć czy istnieje k parami rozłącznych (czyli nie nachodzących na siebie) podsłów $w_1, w_2, \ldots, w_k$ słowa w takich że dla każdego $i \in \{1, \ldots, k\}$, w_i ma długość przynajmniej 2 oraz zaczyna się i kończy symbolem i.
(2) Problem Odległości Edycyjnej Dla zadanego parametru k oraz dwóch słów $u, v \in \Sigma^*$ nad alfabetem Σ, rozstrzygnąć czy v można uzyskać z u za pomocą co najwyżej k operacji edycyjnych, z których każda to usunięcie, wstawienie lub podstawienie pojedynczego symbolu.

Problem Rozłącznych Podsłów wprowadzony został w pracy [30] jako krok pośredni w kierunku trudności kernelizacji dla Problemu Rozłącznych Cykli oraz Problemu Rozłącznych Ścieżek. Problem Odległości Edycyjnej jest zaś ważnym problemem z szeroką gamą zastosowań (przegląd można znaleźć w pracy [128]). Problem ten można rozwiązać w czasie $\mathcal{O}(n^2)$ używając techniki programowania dynamicznego (n jest tutaj łączną długością zadanych słów). Najlepszy obecnie algorytm dla Problemu Odległości Edycyjnej działa w czasie $\mathcal{O}(n^2/(\log n)^2))$ [122]. Przy założeniu hipotezy Strong ETH, nie istnieje silnie podkwadratowy algorytm dla tego problemu [12, 1, 41, 2]. W pracy [D2] badamy parametryzację tego problemu pożądaną odległością edycyjną k. Przy takiej parametryzacji Problem Odległości Edycyjnej da się rozwiązać w czasie $\mathcal{O}(n + k^2)$ [112], a nawet w podliniowym czasie gdy pozwalamy na aproksymację [15, 8, 74].

W pracy [D2] obserwujemy, że zarówno dla Problemu Rozłącznych Słów jak i dla Problemu Odległości Edycyjnej, język instancji na tak można zdefiniować w logice $FO[\Sigma, <]$ używając formuły której rozmiar jest ograniczony funkcją parametru. Stąd na mocy Twierdzenia 47 uzyskujemy struktury danych dla dynamicznych wariantów tych problemów z czasem aktualizacji $f(k) \cdot \log\log n$ (dla pewnej funkcji f). Zależność od parametru określona przez funkcję f jest jednak ogromna, dlatego w pracy [D2] podajemy również wydajniejsze struktury danych skrojone specjalnie dla tych dwóch problemów.

Twierdzenie 48 (Twierdzenie 3 w pracy [D2]). *Dynamiczny wariant* Problemu Rozłącznych Podsłów *ma strukturę danych, która inicjalizuje się w czasie $\mathcal{O}(k2^k + kn)$, obsługuje zapytanie w pesymistycznym czasie $\mathcal{O}(1)$, oraz modyfikuje słowo w pesymistycznym czasie $\mathcal{O}(k2^k \log\log n)$.*

Twierdzenie 49 (Twierdzenie 4 w pracy [D2]). *Dynamiczny wariant* Problemu Odległości Edycyjnej *ma strukturę danych która inicjalizuje się w czasie $\mathcal{O}(kn)$, obsługuje zapytanie w czasie $\mathcal{O}(1)$, oraz modyfikuje słowa w pesymistycznym czasie $\mathcal{O}(k^2 \log\log n)$.*

5. **Omówienie pozostałych osiągnięć naukowo-badawczych**

a) *Osiągnięcia nie wchodzące w skład doktoratu*

W tym rozdziale przedstawimy pozostałe osiągnięcia naukowo-badawcze autorki nie wchodzące w skład doktoratu. Poza głównym polem badań, którym są dla autorki niniejszego opracowania algorytmy dynamiczne oraz online, autorka posiada osiągnięcia naukowe w dziedzinach algorytmiki takich jak planowanie tras, złożoność parametryzowana, czy sparsyfikacja grafów. Omówione niżej osiągnięcia naukowe pogrupowane są według dziedzin. Artykuły opublikowane *przed* uzyskaniem stopnia doktora wspomniane w tym rozdziale oznaczone są ⋆.

Znajdowanie tras na płaszczyźnie Euklidesowej oraz w grafach planarnych. Pierwszą z omawianych w niniejszym opracowaniu dziedzin dodatkowych w które autorka wniosła wkład jest znajdowanie tras na płaszczyźnie Euklidesowej oraz w grafach planarnych. Jest to prężna dziedzina szeroko badana zarówno z teoretycznego jak i z eksperymentalnego punktu widzenia. Dziedzina ta jest bardzo ważna, znajduje ona zastosowanie między innymi w aplikacjach drogowych, w planowaniu rozkładów jazdy i tras pociągów i samolotów, w wizualizacji komputerowej, oraz w projektowaniu VLSI (ang. Very Large Scale Integration).

Wkład autorki w tę dziedzinę składa się z następujących dwóch prac:

[E1] Francois Dross, Krzysztof Fleszar, Karol Węgrzycki, Anna Zych-Pawlewicz.
Gap-ETH-Tight Approximation Schemes for Red-Green-Blue Separation and Bicolored Noncrossing Euclidean Travelling Salesman Tours.
ACM-SIAM Symposium on Discrete Algorithms (SODA), 1433–1463, 2023.
`https://doi.org/10.1137/1.9781611977554.ch52`

Pełny dowód jest zaprezentowany na repozytorium arXiv:
Francois Dross, Krzysztof Fleszar, Karol Węgrzycki, Anna Zych-Pawlewicz.
Gap-ETH-Tight Approximation Schemes for Red-Green-Blue Separation and Bicolored Noncrossing Euclidean Travelling Salesman Tours.
Computing Research Repository, abs/2209.08904, 2022.
`https://arxiv.org/abs/2209.08904`

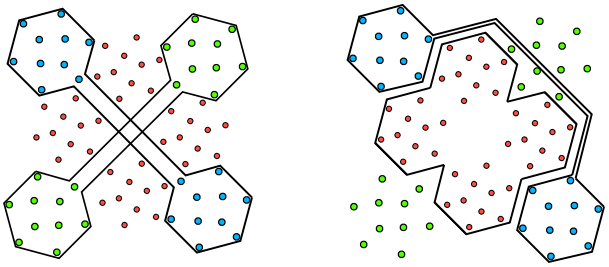
[E2] Holger Flier, Matŭs Mihalăk, Peter Widmayer, Anna Zych, Yusuke Kobayashi, Anita Schöbel
Selecting vertex disjoint paths in plane graphs.
Networks 66(2), 135–144, 2015.
`https://doi.org/10.1002/net.21618`

Wersja konferencyjna:
⋆ Holger Flier, Matŭs Mihalăk, Anita Schöbel, Peter Widmayer, Anna Zych
Vertex Disjoint Paths for Dispatching in Railways.
10th Workshop on Algorithmic Approaches for Transportation Modeling (ATMOS) 14, 61–73, 2010.
`https://doi.org/10.1002/net.21618`

[E3] ⋆ Holger Flier, Matŭs Mihalăk, Peter Widmayer, Anna Zych
Maximum Independent Set in 2-Direction Outersegment Graphs.
Graph-Theoretic Concepts in Computer Science - 37th International Workshop (WG), 6986 , 155–166, 2011.
`https://doi.org/10.1007/978-3-642-25870-1_15`

Przejdźmy do omówienia pracy [E1]. Wyobraźmy sobie, że mamy dane współrzędne domów na mapie, gdzie rodzina mieszkająca w danym domu należy do określonej wspólnoty. Celem jest

oddzielenie wspólnot płotami w taki sposób, aby nie rozdzielić członków tej samej wspólnoty, przy czym chcemy zminimalizować łączną długość płotów. W pracy [E1] badamy szeroką gamę problemów tej natury. Dane mamy n terminali (punktów na płaszczyźnie bądź też wierzchołków w grafie planarnym) pokolorowanych k kolorami, celem zaś jest znalezienie geometrycznych obiektów o minimalnej łącznej długości, które parami nie przecinają się oraz spełniają jakiś warunek dla każdego z kolorów. Na przykład dla wspomnianego wyżej problemu z domami, szukamy nieprzecinających się parami krzywych Jordana które oddzielają terminale o różnych kolorach.



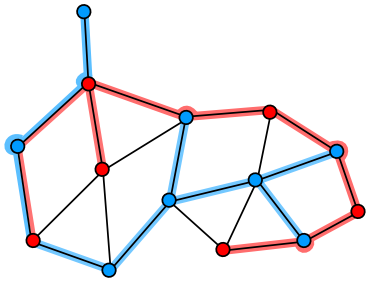
Rysunek 20. Przykład rozwiązania dla Problemu Separacji Czerwony-Niebieski-Zielony. Po lewej rozwiązanie jest niepoprawne, ponieważ krzywe Jordana przecinają się. Rozwiązanie po prawej stronie jest poprawnym rozwiązaniem optymalnym dla tej instancji.

Problemy które tu rozważamy są nie tylko naturalnymi uogólnieniami fundamentalnych i szeroko badanych problemów (takich jak na przykład problem drzewa rozpinającego oraz problem komiwojażera) na bazie których powstało wiele nowych technik o szerokim zastosowaniu. Są one także ważne ze względu na szeroki wachlarz zastosowań, na przykład w projektowaniu VLSI [142, 143, 111, 66] czy też w wizualizacji danych przestrzennych [7, 92, 63, 45, 134]. Przedstawimy teraz konkretne badane przez nas problemy i uzyskane dla nich wyniki.

W Problemie Separacji Czerwony-Niebieski-Zielony mamy dane n punktów na płaszczyźnie, z których każdy ma przypisany kolor czerwony, niebieski bądź zielony. Celem jest znalezienie dwóch nieprzecinających się krzywych Jordana, o minimalnej łącznej długości, które rozdzielają punkty o różnych kolorach. Przykład pokazany jest na Rysunku 20 po prawej. Warianty tego problemu, ze względu na jego liczna zastosowania, były szeroko badane [61, 146, 53, 135, 134]. Problem ten badany był do tej pory tylko dla dwóch kolorów i już w tej wersji jest NP-trudny [61]. Mata i Michel [123] zaproponowali algorytm $\mathcal{O}(\log n)$-aproksymacyjny, który został nieznacznie poprawiony przez Gudmundsson i Levcopoulos [81]. Wreszcie Arora i Chang [11] podali schemat aproksymacyjny (PTAS), który dla każdego $\epsilon > 0$ pozwala uzyskać $(1+\epsilon)$-aproksymację w czasie $n(\log n)^{\mathcal{O}(1/\epsilon)}$. Schemat Arora i Chang [11] opiera się na technice przepięć (ang. patching scheme), która jednak działa tylko dla dwóch kolorów, gdzie szukamy jednej krzywej Jordana. Kontrolowanie dwóch nieprzecinających się krzywych narzuca dużo nowych wyzwań które sprawiają, że uogólnienie wyniku [11] do trzech kolorów nie jest proste. W pracy [E1], po pierwsze, oferujemy alternatywną metodę przepięć (patching scheme) i dzięki niej uogólniamy wynik Arora i Chang do trzech kolorów. Po drugie zaś, poprawiamy czas działania algorytmu Arora i Chang do $2^{\mathcal{O}(1/\epsilon)} n \log^{\mathcal{O}(1)} n$, a zatem otrzymujemy EPTAS dla Problemu Separacji Czerwony-Niebieski-Zielony. Nasza nowa technika przepięć stosuje się także do innych pokrewnych problemów, w tym kolejnego problemu o którym mowa w pracy [E1].

W Problemie Dwóch Komiwojażerów mamy również dane n punktów na płaszczyźnie, tym razem każdy punkt pokolorowany jest na czerwono bądź niebiesko. Celem jest znalezienie dwóch nieprzecinających się tras Komiwojażera: po jednej dla punktów każdego z dwóch kolorów, minimalizując łączną długość tras. Problem ten nie doczekał się jeszcze wydajnych algorytmów,

pomimo że wpisuje się w modny trend znajdowania nieprzecinających się geometrycznych struktur [131, 18, 107]. Dla tego problemu również oferujemy EPTAS, także z czasem $2^{\mathcal{O}(1/\epsilon)}n\log^{\mathcal{O}(1)}n$. Czas ten jest optymalny jeśli przyjmiemy hipotezę Gap-ETH [103]. Oprócz tego podnosimy definicję problemu do grafów planarnych a następnie otrzymujemy dla niego PTAS w tej klasie grafów.



Rysunek 21. Ilustracja przykładowego rozwiązania dla Problemu Dwóch Drzew Rozpinających .

Kolejnym badanym przez nas problemem jest Problem Dwóch Drzew Rozpinających w grafach planarnych. Jest to naturalne uogólnienie klasycznego problemu drzewa rozpinającego. W Problemie Dwóch Drzew Rozpinających każdy wierzchołek grafu planarnego jest pokolorowany na niebiesko lub czerwono. Celem jest znalezienie dwóch nieprzecinających się drzew o minimalnym łącznym rozmiarze, z których jedno rozpina wszystkie czerwone wierzchołki, a drugie rozpina wszystkie niebieskie wierzchołki. Drzewa nie przecinają się, jeśli da się je narysować bez przecięć wewnątrz pogrubionego rysunku grafu na płaszczyźnie, jak na Rysunku 21. W pracy [E1] pokazujemy NP-trudność tego problemu oraz PTAS dla tego problemu.

Ostatnim poruszanym tu przez nas problemem jest Problem Kolorowych Nieprzecinających się Ścieżek . Mamy w nim dane n par punktów na płaszczyźnie, a zadaniem jest połączenie par ścieżkami tak, aby żadne dwie ścieżki nie przecinały się i aby zminimalizować łączną długość ścieżek. Problem ten otrzymał sporo uwagi badaczy [47, 113, 18, 130, 66, 132] zwłaszcza pod kątem aproksymacji, dlatego też nasz wynik, w którym pokazujemy NP-trudność tego problemu, bardzo dobrze uzupełnia wyniki dostępne w literaturze.

Różne warianty Problemu Kolorowych Nieprzecinających się Ścieżek na grafach planarnych badamy również w pracy [E2]. Problem ten w klasie grafów planarnych definiujemy następująco. Na wejściu mamy dany graf planarny wraz z zanurzeniem w płaszczyznę. Dane mamy również k par wierzchołków (terminali) $\{(s_i,t_i)\}_{i\in\{1,\ldots,k\}}$, oraz dla każdej pary (s_i,t_i) mamy zadany zbiór ścieżek pomiędzy nimi, który nazwiemy ścieżkami o kolorze i. Parametrem p oznaczymy maksymalną liczbę ścieżek w jednym kolorze. Dla tak zadanej instancji rozważamy trzy cele:

Dec stwierdzić czy można wybrać k rozłącznych wierzchołkowo ścieżek po jednej z każdego koloru

Maks stwierdzić ile maksymalnie można wybrać rozłącznych wierzchołkowo ścieżek po jednej z każdego koloru

Rund wybieramy ścieżki w rundach, przy czym w jednej rundzie wybieramy dowolny zbiór wierzchołkowo rozłącznych ścieżek: stwierdzić jaka jest najmniejsza liczba rund, taka że z każdego koloru w jakiejś rundzie zostanie wybrana ścieżka.

Naszą główną motywacją w pracy [E2] jest wydajne planowanie tras i rozkładów jazdy pociągów. Z tego punktu widzenia ma sens wyróżnić następujące rozważane przez nas warianty położenia terminali w grafie:

Dowolne dowolne pary terminali

Zewnętrzne wszystkie terminale położone są w dowolny sposób na ścianie zewnętrznej

	Dowolne	Zewnętrzne	Dwudzielne	Zagnieżdżone
Dec	NP-trudny dla $p \geqslant 3$, wielomianowy alg. dla $p \leqslant 2$	otwarty (trywialny dla $p = 1$)	wielomianowy alg.	
Maks	NP-trudny dla $p \geqslant 1$	otwarty		
Rund	APX-trudny		p-aproksymacja, APX-trudny dla $p \geqslant 2$, wielomianowy alg. dla $p = 1$	

Tablica 1. Wyniki dla różnych wariantów Problemu Kolorowych Nieprzecinających się Ścieżek na grafach planarnych

Dwudzielne wszystkie terminale położone są na ścianie zewnętrznej; dodatkowo przechodząc po ścianie zewnętrznej w kierunku przeciwnym do ruchu wskazówek zegara zaczynając od s_1, napotkamy po kolei terminale $s_1, s_2, \ldots, s_k, t_{\pi(1)}, \ldots, t_{\pi(k)}$, gdzie π jest dowolną permutacją na zbiorze kolorów.

Zagnieżdżone wszystkie terminale położone są na ścianie zewnętrznej; dodatkowo przechodząc po ścianie zewnętrznej w kierunku przeciwnym do ruchu wskazówek zegara zaczynając od s_1, napotkamy po kolei terminale $s_1, s_2, \ldots, s_k, t_k, \ldots, t_1$.

W pracy [E2] rozważamy wszystkie kombinacje wymienionych wyżej celi i możliwości położenia terminali. Uzyskane wyniki dla wszystkich możliwych scenariuszy umieszczone są w Tabeli a)

Złożoność parametryzowana i zwarta parametryzowana reprezentacja. W tej sekcji przedstawimy pozostałe (poza wymienionymi w głównym osiągnięciu naukowym) osiągnięcia autorki powiązane z dziedziną złożoności parametryzowanej. Osiągnięcia te zawarte są w niżej wymienionych artykułach.

[F1] MichałPilipczuk, Marek Sokołowski, Anna Zych-Pawlewicz.
Compact Representation for Matrices of Bounded Twin-Width.
39th International Symposium on Theoretical Aspects of Computer Science (STACS), 219 , 52:1–52:14, 2022.
https://doi.org/10.4230/LIPIcs.STACS.2022.52

Pełny dowód jest zaprezentowany na repozytorium arXiv:
MichałPilipczuk, Marek Sokołowski, Anna Zych-Pawlewicz.
Compact Representation for Matrices of Bounded Twin-Width.
Computing Research Repository, abs/2110.08106, 2021.
https://arxiv.org/abs/2110.08106

[F2] Nikolai Karpov, Marcin Pilipczuk, Anna Zych-Pawlewicz.
An Exponential Lower Bound for Cut Sparsifiers in Planar Graphs.
Algorithmica, 81 (10), 4029–4042, 2019.
https://doi.org/10.1007/s00453-018-0504-8

Wersja konferencyjna:
Nikolai Karpov, Marcin Pilipczuk, Anna Zych.
An Exponential Lower Bound for Cut Sparsifiers in Planar Graphs.
12th International Symposium on Parameterized and Exact Computation (IPEC), 89, 24:1–24:11, 2017.
https://doi.org/10.4230/LIPIcs.IPEC.2017.24

[F3] Johannes Blum, Yann Disser, Andreas Emil Feldman, Siddharth Gupta, Anna Zych-Pawlewicz.

On Sparse Hitting Sets: From Fair Vertex Cover to Highway Dimension.
17th International Symposium on Parameterized and Exact Computation (IPEC), 249 , 5:1–5:23, 2022.
https://doi.org/10.4230/LIPIcs.IPEC.2022.5

Pełny dowód jest zaprezentowany na repozytorium arXiv:
Johannes Blum, Yann Disser, Andreas Emil Feldman, Siddharth Gupta, Anna Zych-Pawlewicz.
On Sparse Hitting Sets: From Fair Vertex Cover to Highway Dimension.
Computing Research Repository, abs/2208.14132, 2022.
https://arxiv.org/abs/2208.14132

Ważnym wspólnym mianownikiem pierwszych dwóch prac jest zwarta reprezentacja grafów przy zadanym parametrze. Zwarta reprezentacja to bardzo istotny temat z punktu widzenia nowoczesnych aplikacji, zwłaszcza aplikacji mobilnych, które potrzebują szybkiego dostępu do ogromnych grafów (np. współczesnych sieci drogowych). Dla takich aplikacji to pamięć jest głównym ograniczeniem, choć szybki czas działania jest również pożądany. Często graf na którym operuje aplikacja nie może zmieścić się w pamięci urządzenia. Aby poradzić sobie z tym problemem stosuje się preprocessing, podczas którego graf jest kompresowany w taki sposób, aby nie utracić ważnych z punktu widzenia aplikacji własności.

Praca [F1] poświęcona jest kompresji grafów o małej szerokości bliźniaczej. Parametr ten został wprowadzony niedawno przez Bonnet *i innych* [35] i od razu zdobył ogromne zainteresowanie badaczy [31, 32, 33, 34, 36, 59, 71]. Prace te pokazują że parametr ten daje bogatą strukturę kombinatoryczną [31, 35, 59], jest przydatny algorytmicznie [32, 34, 35], a także ma głębokie powiązania z teorią modeli [33, 35, 36, 71]. W szczególności weryfikacja formuł logiki FO na grafach z ograniczoną szerokością bliźniaczą ma liniowy algorytm FPT [35]. Mała szerokość bliźniacza, w przeciwieństwie do wielu innych parametrów, nie oznacza że graf jest rzadki: na przykład klika ma szerokość bliźniaczą zero. Jednak każdy graf o szerokości bliźniaczej ograniczonej przez parametr d posiada macierz sąsiedztwa która jest d-uporządkowana. Taka macierz jest dla nas punktem wyjścia do zwartej reprezentacji, zajmującej pamięć liniową względem liczby wierzchołków n, która jednocześnie umożliwia szybki dostęp do krawędzi grafu. W pracy [F1] pokazujemy strukturę danych, która zajmuje $\mathcal{O}_d(n)$ bitów pamięci oraz odpowiada na zapytanie o konkretną wartość w d-uporządkowanej macierzy (krawędź grafu) w czasie $\mathcal{O}_d(\log\log n)$.

W pracy [F2] interesuje nas innego rodzaju zwarta reprezentacja, która w literaturze nosi nazwę *sieci imitującej*, wprowadzonej w pracy [85]. Niech G będzie grafem z wagami na krawędziach zaś $Q \subseteq V(G)$ niech będzie zbiorem k terminali. Dla podziału zbioru terminali $Q = S \uplus \bar{S}$, *minimalnym S-rozdzielającym przekrojem* nazwiemy minimalny przekrój pomiędzy S i $\bar{S}$. *Siecią imitującą* dla pary (G, Q) nazwiemy ważony krawędziowo graf G', taki że $Q \subseteq V(G')$ oraz wagi S-rozdzielających przekrojów dla wszystkich $S \subseteq Q$ są takie same w G i w G'. Hagerup *i inni* [85] obserwują, iż dla dowolnego grafu G można w prosty sposób uzyskać sieć imitującą rozmiaru $2^{2^{k+1}}$. Wynik ten został poprawiony do $2^{\binom{k-1}{\lfloor (k-1)/2 \rfloor}}$ [100, 46]. Krauthgamer i Rika [109] adaptują wynik z [85] aby uzyskać sieć imitującą o rozmiarze $\mathcal{O}(k^2 2^{2k})$ dla grafów planarnych. Co więcej, w tej samej pracy pokazują oni ograniczenie dolne na rozmiar sieci imitującej dla grafów ogólnych w postaci $2^{\Omega(k)}$. Ograniczenie dolne dla grafów planarnych, do momentu powstania pracy [F2] było zaś postaci $\Omega(k^2)$ [109]. Sporo wysiłku zostało też włożone w zrozumienie, jak duża musi być sieć imitująca dla grafów planarnych gdy wszystkie terminale leżą na ograniczonej liczbie ścian [110, 76]. W pracy [F2] naszym głównym osiągnięciem jest uzupełnienie tych wyników, poprzez pokazanie ograniczenia dolnego na rozmiar sieci imitującej dla grafów planarnych w postaci 2^{k-1}.

W pracy [F3], nie powiązanej z dwoma poprzednimi, rozważamy zaś szeroką gamę problemów parametryzowanych nazywanych w literaturze problemami sprawiedliwymi. Konkretniej, zajmujemy się tu Problemem Rzadkiego Zbioru Reprezentantów (ang. Sparse Hitting Set). Zbiorem reprezentantów dla rodziny zbiorów $\mathcal{F}$ nazwiemy zbiór $X \subseteq \bigcup \mathcal{F}$ który przecina się niepusto ze wszystkimi zbiorami z $\mathcal{F}$. W Problemie Rzadkiego Zbioru Reprezentantów na wejściu mamy system zbiorów $(V, \mathcal{F}, \mathcal{B})$, gdzie zbiór V nazywany jest uniwersum, zaś $\mathcal{F}, \mathcal{B}$ są rodzinami podzbiorów V. Celem jest znalezienie zbioru reprezentantów X dla rodziny $\mathcal{F}$ który minimalizuje maksymalne przecięcie ze zbiorem z $\mathcal{B}$. Maksymalne przecięcie optymalnego rozwiązania ze zbiorem w $\mathcal{B}$ nazywamy tu *rzadkością* instancji i oznaczamy literą k-jest to nasz centralny parametr.

Problem Rzadkiego Zbioru Reprezentantów uogólnia Problem Sprawiedliwego Pokrycia Wierzchołkowego, gdzie dla zadanego grafu szukamy pokrycia wierzchołkowego, które minimalizuje maksymalne przecięcie z sąsiedztwem wierzchołka w grafie. Problem ten parametryzowano całą gamą parametrów, wliczając głębokość drzewiastą, szerokość drzewiastą, szerokość modularną[104, 121] a także rozmiar rozwiązania[95]. Najbardziej naturalnym jednak parametrem dla Problemu Sprawiedliwego Pokrycia Wierzchołkowego jest rzadkość k. Wcześniejsze prace pozostawiły jako problem otwarty pytanie, czy dla Problemu Sprawiedliwego Pokrycia Wierzchołkowego parametryzowanego rzadkością istnieje algorytm FPT lub chociaż XP. W pracy [F3] odpowiadamy na to pytanie negatywnie, pokazując NP-trudność dla dowolnej stałej $k \geqslant 4$ w klasie grafów planarnych. Z pozytywnej zaś strony pokazujemy $(2 - \frac{1}{k})$ aproksymację dla tego problemu.

Innym znanym przedstawicielem Problemu Rzadkiego Zbioru Reprezentantów jest Problem Pokrycia r-Krótkich Ścieżek. Jest to ważny problem z punktu widzenia coraz szerzej badanego parametru *wymiar autostradowy*, który modeluje sieci transportu [5]. Jako że sieci transportu wydają się mieć ograniczony wymiar autostradowy [14], powstało wiele wydajnych algorytmów dla grafów z małym wymiarem autostradowym [68, 5, 67, 69, 57, 57, 17, 96, 39, 44]. Aby wykorzystać wynikającą z niego strukturę grafu, wszystkie znane algorytmy rozwiązują najpierw powiązany Problem Pokrycia r-Krótkich Ścieżek. Dotychczas znany był algorytm XP dla tego problemu, nie było jednak wiadomo czy można liczyć na algorytm FPT jeśli parametrem jest rzadkość k. W pracy [F3] pokazujemy że Problem Pokrycia r-Krótkich Ścieżek parametryzowany rzadkością jest W[1]-trudny. Z pozytywnej jednak strony oferujemy $\mathcal{O}(\log n)$-aproksymację dla tego problemu.

b) *Osiągnięcia wchodzące w skład doktoratu*

Re-optymalizacja. W ramach doktoratu autorka badała model reoptymalizacji dla problemów NP-zupełnych. W modelu tym zakładamy, że oprócz instancji konkretnego problemu optymalizacyjnego(jak na przykład problem drzew Steinera), dostajemy na wejściu podpowiedź w postaci optymalnego rozwiązania dla tej instancji, oraz lokalną modyfikację tej instancji (w postaci np. dodania lub usunięcia krawędzi). Wyposażeni w podpowiedź, chcemy szybko obliczyć dobre rozwiązanie dla zmodyfikowanej instancji.

Model ten udało się z dużym sukcesem przyłożyć to takich problemów jak problem drzewa Steinera [G3, G5, G6], problem najkrótszego nadsłowa [G4], problem zbioru niezależnego o maksymalnej wadze, kliki o maksymalnej wadze, zbioru dominującego o minimalnej wadze, oraz pokrycia zbiorami i wierzchołkowego o minimalnej wadze [G7]. Ogólne techniki i narzędzia reoptymalizacji zostały omówione w pracy [G2]. Wszystkie wyżej wymienione prace są częścią doktoratu autorki. Wyniki te stały się inspiracją do dalszych badań nad re-optymalizacją, w związku z czym powstała kolejna praca doktorska [141] i szereg publikacji [77, 22, 28, 29, 26,

4, 52, 27]. Artykuły [77, 22] są bezpośrednią kontynuacją badań nad re-optymalizacją drzew Steinera zamieszczonych w doktoracie autorki.

[G1] Anna Zych.
Reoptimization of NP-hard Problems.
PhD thesis, ETH Zurich, Zürich, Switzerland, 2012.
https://hdl.handle.net/20.500.11850/72820

[G2] Anna Zych-Pawlewicz.
Reoptimization of NP-hard problems.
Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday, Lecture Notes in Computer Science, 11011 , 477–494, 2018.
https://doi.org/10.1007/978-3-319-98355-4_28

[G3] Davide Biló, Anna Zych
New Advances in Reoptimizing the Minimum Steiner Tree Problem.
Mathematical Foundations of Computer Science 2012 - 37th International Symposium (MFCS), 7464, 184–197, 2012.
https://doi.org/10.1007/978-3-642-32589-2_19

[G4] Davide Biló, Hans Joachim Böckenhauer, Dennis Komm, Richard Královic, Tobias Mömke, Sebastian Seibert, Anna Zych.
Reoptimization of the Shortest Common Superstring Problem.
Algorithmica, 61 (2), 227–251, 2011.
https://doi.org/10.1007/s00453-010-9419-8

Wersja konferencyjna:
Davide Biló, Hans Joachim Böckenhauer, Dennis Komm, Richard Královic, Tobias Mömke, Sebastian Seibert, Anna Zych.
Reoptimization of the Shortest Common Superstring Problem.
Combinatorial Pattern Matching (CPM), 20th Annual Symposium, 5577, 78–91, 2009.
https://doi.org/10.1007/978-3-642-02441-2_8

[G5] Davide Biló, Anna Zych.
New Reoptimization Techniques applied to Steiner Tree Problem.
Electron. Notes Discret. Math., 37 , 387–392, 2011.
https://doi.org/10.1016/j.endm.2011.05.066

[G6] Davide Biló, Hans Joachim Böckenhauer, Juraj Hromkovic, Richard Královic, Tobias Mömke, Peter Widmayer, Anna Zych.
Reoptimization of Steiner Trees.
11th Scandinavian Workshop on Algorithm Theory (SWAT) 5124, 258–269, 2008.
https://doi.org/10.1007/978-3-540-69903-3_24

[G7] Davide Biló, Peter Widmayer, Anna Zych.
Reoptimization of Weighted Graph and Covering Problems.
6th International Workshop on Approximation and Online Algorithms(WAOA) 5426, 201–213, 2008.
https://doi.org/10.1007/978-3-540-93980-1_16

6. **Informacja o wykazywaniu się istotną aktywnością naukową realizowaną w więcej niż jednej uczelni lub instytucji naukowej, w szczególności zagranicznej.**

W latach 2007–2012 autorka w ramach doktoratu pracowała na uczelni ETH w Zurychu. Najintensywniej współpracowała tam z następującymi badaczami, z których pierwszy i trzeci są wciąż aktywni naukowo:

- Davide Biló
- Peter Widmayer
- Matŭs Mihalăk
- Holger Flier

W wyniku aktywności autorki w trakcie pobytu na ETH powstały liczne publikacje, część których znalazła się w rozprawie doktorskiej autorki. Wyniki te wymienione są w Rozdziale 5. Autorka w dalszym ciągu prowadzi aktywną współpracę zagraniczną. Poniżej wymienione są wyjazdy autorki na współpracę naukową z ostatnich lat.

- 14.8.2033 – 18.8.2023, University of Sheffield (Wielka Brytania):
  współpraca z prof. Yannem Disser oraz z dr Andreasem Feldmann
- 05.07–23.07.2022 oraz 06.07.2019–23.07.2019, University of Sassari (Włochy):
  współpraca z prof. Davide Biló
- 05.08.2018–12.08.2018, University of Darmstadt (Niemcy):
  współpraca z prof. Yannem Disser oraz z dr Andreasem Feldmann
- 03.02.2018–13.02.2018, University of Copenhagen (Dania):
  współpraca z prof. Jacobem Holm oraz z prof. Evą Rotenberg

Ponadto jeszcze przed doktoratem w Zurychu autorka odbyła roczny staż w Philips Research Campus w Eindhoven współpracując z prof. Milanem Petkovič, co zaowocowało kilkoma publikacjami i patentem.

7. **Informacja o osiągnięciach dydaktycznych, organizacyjnych oraz popularyzujących naukę lub sztukę.**

a) *Regularna aktywność dydaktyczna*

Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski, Polska:

(1) Ćwiczenia do wykładu Algorytmy i Struktury Danych:
2013, 2014, 2016, 2017, 2018, 2019, 2020, 2022, 2023
(2) Ćwiczenia do wykładu Prawdopodobieństwo i Statystyka:
2013, 2014, 2016, 2021, 2022, 2023
(3) Ćwiczenia do wykładu Algorytmika:
2017, 2018, 2019, 2020, 2021, 2022, 2023
(4) Koordynator Seminarium Magisterskiego:
2018, 2019, 2020, 2021, 2022
(5) Koordynator, wykład i ćwiczenia do przedmiotu Algorytmy i Struktury Danych dla bioinformatyki:
2014, 2016
(6) Laboratorium do wykładu Algorytmy i Struktury Danych:
2012, 2013, 2014

b) *Wypromowani magistranci*

(1) Kamil Ćwintal, Algorytmiczne metody dla dynamicznego utrzymywania orientacji o ograniczonym stopniu wyjściowym na lasach (2022)
(2) Jarosław Brojek, Problem maksymalnego ważonego skojarzenia online z twardym budżetem na liczbę zmian - ograniczenie górne i dolne (2022)
(3) Mateusz Rychlicki, Dynamiczne struktury danych dla parametryzowanych problemów tekstowych (2022)
(4) Bartłomiej Wróblewski, Statyczny i przyrostowy parametryzowany problem LONG CYCLE (2020)
(5) Paweł Banaszewski, Eksperymentalna ewaluacja planarnych wyroczni odległości na grafach map drogowych (2018)
(6) Marek Żochowski, Dynamiczne grafowe algorytmy parametryzowane (w trakcie)

c) *Wypromowane licencjaty*

(1) Jędrzej Olkowski, Parametryzowana złożoność dla sprawiedliwych wersji znanych problemów (2023)

d) *Popularyzacja*

(1) Udział w programie "Szkoła Orłów" jako mentor Jędrzeja Olkowskiego
(2) Referat na popularyzatorskiej konferencji Operations Research (OR) w Berlinie (2017)
(3) Referat na dniach otwartych w IX LO im. Klementyny Hoffmanowej (2017)
(4) Team leader polskiej drużyny na CEOI, (Jena 2014)
(5) Wykłady dla uzdolnionych licealistów w ramach Krajowego Funduszu na Rzecz Dzieci (2013)

Literatura

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015.

[2] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 375–388. ACM, 2016.

[3] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014.

[4] Jeffrey Aborot, Henry N. Adorna, and Jhoirene B. Clemente. On self-reducibility and reoptimization of closest substring problem. *CoRR*, abs/1603.02457, 2016.

[5] Ittai Abraham, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 782–793. SIAM, 2010.

[6] Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. In *Proceedings of the 44^{th} International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 41:1–41:16. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2017.

[7] Basak Alper, Nathalie Henry Riche, Gonzalo Ramos, and Mary Czerwinski. Design study of LineSets, a novel set visualization technique. *IEEE Trans. Vis. Comput. Graphics*, 17(12):2259–2267, 2011.

[8] Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 990–1001. IEEE, 2020.

[9] Elliot Anshelevich, Anirban Dasgupta, Jon Kleinberg, Eva Tardos, Tom Wexler, and Tim Roughgarden. The price of stability for network design with fair cost allocation. In *In FOCS*, pages 295–304, 2004.

[10] Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

[11] Sanjeev Arora and Kevin L. Chang. Approximation schemes for degree-restricted MST and red-blue separation problems. *Algorithmica*, 40(3):189–210, 2004.

[12] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018.

[13] Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. Dynamic graph coloring. In *Algorithms and data structures*, volume 10389 of *Lecture Notes in Comput. Sci.*, pages 97–108. Springer, Cham, 2017.

[14] Holger Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes. In transit to constant time shortest-path queries in road networks. In *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 46–59. SIAM, 2007.

[15] Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 316–324. ACM, 2003.

[16] Dwight R. Bean. Effective coloration. *The Journal of Symbolic Logic*, 41(2):469–480, 1976.

[17] Amariah Becker, Philip N Klein, and David Saulpic. Polynomial-time approximation schemes for k-center, k-median, and capacitated vehicle routing in bounded highway dimension. In *26th Annual European Symposium on Algorithms (ESA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[18] Sergey Bereg, Krzysztof Fleszar, Philipp Kindermann, Sergey Pupyrev, Joachim Spoerhase, and Alexander Wolff. Colored non-crossing euclidean steiner forest. In Khaled M. Elbassioni and Kazuhisa Makino, editors, *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, volume 9472 of *Lecture Notes in Computer Science*, pages 429–441. Springer, 2015.

[19] Marshall W. Bern and Paul E. Plassmann. The Steiner problem with edge lengths 1 and 2. *Inf. Process. Lett.*, 32(4):171–176, 1989.

[20] Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized $O(\log^2 n)$ replacements. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 947–959. SIAM, Philadelphia, PA, 2018. `https://doi.org/10.1137/1.9781611975031.61`.

[21] Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015. `https://doi.org/10.1007/978-3-662-47672-7_14`.

[22] Davide Bilò. New algorithms for steiner tree reoptimization. In *International Colloquium on Automata, Languages and Programming*, 2018.

[23] Vittorio Bilò, Michele Flammini, and Luca Moscardelli. The price of stability for undirected broadcast network design with fair cost allocation is constant. In *FOCS*, pages 638–647. IEEE Computer Society, 2013.

[24] Benjamin E. Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, 2008. `https://doi.org/10.1145/1360443.1360462`.

[25] M. Biró, M. Hujter, and Zs. Tuza. Precoloring extension. i. interval graphs. *Discrete Mathematics*, 100(1):267–279, 1992.

[26] Hans-Joachim Böckenhauer, Karin Freiermuth, Juraj Hromkovič, Tobias Mömke, Andreas Sprock, and Björn Steffen. Steiner tree reoptimization in graphs with sharpened triangle inequality. *J. of Discrete Algorithms*, 11:73–86, feb 2012.

[27] Hans-Joachim Böckenhauer, Juraj Hromkovic, and Dennis Komm. Reoptimization of hard optimization problems. In *Handbook of Approximation Algorithms and Metaheuristics*, 2018.

[28] Hans-Joachim Böckenhauer, Juraj Hromkovič, and Andreas Sprock. *Knowing All Optimal Solutions Does Not Help for TSP Reoptimization*, pages 7–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[29] Hans-Joachim Böckenhauer, Juraj Hromkovič, and Andreas Sprock. On the hardness of reoptimization with multiple given solutions. *Fundam. Inf.*, 110(1–4):59–76, jan 2011.

[30] Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011.

[31] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1977–1996. SIAM, 2021.

[32] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: Max Independent Set, Min Dominating Set, and Coloring. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2021.

[33] Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. *CoRR*, abs/2102.03117, 2021.

[34] Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. *CoRR*, abs/2107.02882, 2021.

[35] Édouard Bonnet, Eun Jung Kim, Stéphan Thomasse, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *IEEE 61st Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 601–612. IEEE Computer Society, 2020.

[36] Édouard Bonnet, Jaroslav Nešetřil, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. *CoRR*, abs/2102.06880, 2021.

[37] Glencora Borradaile, Philip N. Klein, and Claire Mathieu. An O(n log n) approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms*, 5(3), 2009.

[38] Bartłomiej Bosek, Stefan Felsner, Kamil Kloch, Tomasz Krawczyk, Grzegorz Matecki, and Piotr Micek. On-line chain partitions of orders: a survey. *Order*, 29(1):49–73, 2012.

[39] Vladimir Braverman, Shaofeng H-C Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in excluded-minor graphs and beyond. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2679–2696. SIAM, 2021.

[40] Marcin Brianski, Gwenaël Joret, Konrad Majewski, Piotr Micek, Michal T. Seweryn, and Roohani Sharma. Treedepth vs circumference. *Comb.*, 43(4):659–664, 2023.

[41] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97. IEEE Computer Society, 2015.

[42] Niv Buchbinder, Danny Segev, and Yevgeny Tkach. Online algorithms for maximum cardinality matching with edge arrivals. *Algorithmica*, 81(5):1781–1799, may 2019.

[43] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, February 2013.

[44] Martin Böhm, Ruben Hoeksma, Nicole Megow, Lukas Nölke, and Bertrand Simon. On hop-constrained steiner trees in tree-like metrics. *SIAM Journal on Discrete Mathematics*, 36(2):1249–1273, 2022.

[45] Thom Castermans, Mereke van Garderen, Wouter Meulemans, Martin Nöllenburg, and Xiaoru Yuan. Short plane supports for spatial hypergraphs. *Journal of Graph Algorithms and Applications*, 23(3):463–498, 2019.

[46] Erin W. Chambers and David Eppstein. Flows in one-crossing-minor-free graphs. *J. Graph Algorithms Appl.*, 17(3):201–220, 2013.

[47] Timothy M. Chan, Hella-Franziska Hoffmann, Stephen Kiazyk, and Anna Lubiw. Minimum length embedding of planar graphs at fixed vertex locations. In Stephen Wismath and Alexander Wolff, editors, *Proc. 21st Int. Symp. Graph Drawing (GD'13)*, volume 8242, pages 376–387, 2013.

[48] Kamalika Chaudhuri, Constantinos Daskalakis, Robert D. Kleinberg, and Henry Lin. Online bipartite perfect matching with augmentations. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*, pages 1044–1052. IEEE, 2009. https://doi.org/10.1109/INFCOM.2009.5062016.

[49] Shiri Chechik. Improved distance oracles and spanners for vertex-labeled graphs. In Leah Epstein and Paolo Ferragina, editors, *ESA*, volume 7501 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2012.

[50] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022.

[51] Ashish Chiplunkar, Sumedh Tirodkar, and Sundar Vishwanathan. On randomized algorithms for matching in the online preemptive model. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015*, pages 325–336, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[52] Jhoirene B. Clemente and Henry N. Adorna. Reoptimization of the closest substring problem under pattern length modification, 2017.

[53] Martin C Cooper. The tractability of segmentation and scene analysis. *International Journal of Computer Vision*, 30(1):27–42, 1998.

[54] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[55] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[56] Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 101–107. SIAM, 2013. https://doi.org/10.1137/1.9781611973105.7.

[57] Yann Disser, Andreas Emil Feldmann, Max Klimm, and Jochen Könemann. Travelling on graphs with small highway dimension. *Algorithmica*, 83(5):1352–1370, 2021.

[58] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness i: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[59] Jan Dreier, Jakub Gajarský, Yiting Jiang, Patrice Ossona de Mendez, and Jean-Florent Raymond. Twin-width and generalized coloring numbers. *CoRR*, abs/2104.09360, 2021.

[60] Zdeněk Dvořák, Martin Kupec, and Vojtěch Tůma. A dynamic data structure for MSO properties in graphs with bounded tree-depth. In *Proceedings of the 22nd Annual European Symposium on Algorithms, ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2014.

[61] Peter Eades and David Rappaport. The complexity of computing minimum separating polygons. *Pattern Recognit. Lett.*, 14(9):715–718, 1993.

[62] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972. https://doi.org/10.1145/321694.321699.

[63] Alon Efrat, Yifan Hu, Stephen Kobourov, and Sergey Pupyrev. Mapsets: Visualizing embedded and clustered graphs. *J. Graph Algorithms and Applications*, 19(2):571–593, 2015.

[64] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification. I. Planary testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996.

[65] Leah Epstein and Meital Levy. Online interval coloring and variants. In *Proceedings of the 32nd International Conference on Automata, Languages and Programming*, ICALP'05, page 602–613, Berlin, Heidelberg, 2005. Springer-Verlag.

[66] Jeff Erickson and Amir Nayyeri. Shortest non-crossing walks in the plane. In *Proc. 22nd ACM-SIAM Symp. Discrete Algorithms (SODA'11)*, pages 297–308, 2011.

[67] Andreas Emil Feldmann. Fixed-parameter approximations for k-center problems in low highway dimension graphs. *Algorithmica*, 81(3):1031–1052, 2019.

[68] Andreas Emil Feldmann, Wai Shing Fung, Jochen Könemann, and Ian Post. A $(1+\varepsilon)$-embedding of low highway dimension graphs into bounded treewidth graphs. *SIAM Journal on Computing*, 47(4):1667–1704, 2018.

[69] Andreas Emil Feldmann and David Saulpic. Polynomial time approximation schemes for clustering in low highway dimension graphs. *J. Comput. Syst. Sci.*, 122:72–93, 2021.

[70] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997.

[71] Jakub Gajarský, Michal Pilipczuk, and Szymon Toruńczyk. Stable graphs of bounded twin-width. *CoRR*, abs/2107.03711, 2021.

[72] Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 26–37, 2019.

[73] Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In *SODA*, pages 942–951, 2008.

[74] Elazar Goldenberg, Tomasz Kociumaka, Robert Krauthgamer, and Barna Saha. Gap edit distance via non-adaptive queries: Simple and optimal. *CoRR*, abs/2111.12706, 2021.

[75] Martin Charles Golumbic. Chapter 8 - interval graphs. In Martin Charles Golumbic, editor, *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*, pages 171–202. Elsevier, 2004.

[76] Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved Guarantees for Vertex Sparsification in Planar Graphs. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[77] Keshav Goyal and Tobias Mömke. Robust reoptimization of steiner trees. *Algorithmica*, 82, 07 2020.

[78] Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for Closest String and related problems. *Algorithmica*, 37(1):25–42, 2003.

[79] Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. Online perfect matching and mobile computing. In Selim G. Akl, Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures, 4th International Workshop, WADS '95, Kingston, Ontario, Canada, August*

16-18, 1995, Proceedings, volume 955 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 1995. https://doi.org/10.1007/3-540-60220-8_62.

[80] Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: maintaining a constant-competitive Steiner tree online. In *STOC*, pages 525–534, 2013.

[81] Joachim Gudmundsson and Christos Levcopoulos. A Fast Approximation Algorithm for TSP with Neighborhoods and Red-Blue Separation. In Takano Asano, Hideki Imai, D. T. Lee, Shin-ichi Nakano, and Takeshi Tokuyama, editors, *Computing and Combinatorics*, pages 473–482, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[82] Anupam Gupta and Amit Kumar. Online Steiner tree with deletions. In Chandra Chekuri, editor, *SODA*, pages 455–467. SIAM, 2014.

[83] Anupam Gupta, Martin Pál, R. Ravi, and Amitabh Sinha. Boosted sampling: approximation algorithms for stochastic optimization. In *STOC*, pages 417–426, 2004.

[84] Manoj Gupta and Richard Peng. Fully Dynamic $(1+\epsilon)$-Approximate Matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 548–557. IEEE Computer Society, 2013. https://doi.org/10.1109/FOCS.2013.65.

[85] Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *J. Comput. Syst. Sci.*, 57(3):366–375, 1998.

[86] Magnús M. Halldórsson and Christian Konrad. Improved distributed algorithms for coloring interval graphs with application to multicoloring trees. *Theoretical Computer Science*, 811:29–41, 2020. Special issue on Structural Information and Communication Complexit.

[87] Magnús M. Halldórsson and Mario Szegedy. Lower bounds for on-line graph coloring, 1994.

[88] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 21–30, New York, NY, USA, 2015. Association for Computing Machinery.

[89] Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster. Distance oracles for vertex-labeled graphs. In *ICALP*, pages 490–501, 2011.

[90] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[91] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973. https://doi.org/10.1137/0202019.

[92] Ferran Hurtado, Matias Korman, Marc J. van Kreveld, Maarten Löffler, Vera Sacristán, Akiyoshi Shioura, Rodrigo I. Silveira, Bettina Speckmann, and Takeshi Tokuyama. Colored spanning graphs for set visualization. *Comput. Geom.*, 68:262–276, 2018. Special issue in memory of Ferran Hurtado.

[93] J Kratochv Il. Precoloring extension with fixed color bound. 1994.

[94] Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991.

[95] Ashwin Jacob, Venkatesh Raman, and Vibha Sahlot. Deconstructing parameterized hardness of fair vertex deletion problems. In *International Computing and Combinatorics Conference*, pages 325–337. Springer, 2019.

[96] Aditya Jayaprakash and Mohammad R Salavatipour. Approximation schemes for capacitated vehicle routing on graphs of bounded treewidth, bounded doubling, or highway dimension. *arXiv preprint arXiv:2106.15034*, 2021.

[97] Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random nc. *Combinatorica*, 6(1):35–48, 1986.

[98] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358. ACM, 1990. https://doi.org/10.1145/100216.100262.

[99] Marek Karpinski and A. Zelikovsky. New approximation algorithms for the steiner tree problems. *Journal of Combinatorial Optimization*, 1:47–65, 03 1997.

[100] Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Inf. Process. Lett.*, 114(7):365–371, 2014.

[101] Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for MaxClique, chromatic number and Min-3Lin-Deletion. In *Automata, languages and programming. Part I*, volume 4051 of *Lecture Notes in Comput. Sci.*, pages 226–237. Springer, Berlin, 2006.

[102] Henry A. Kierstead and William T. Trotter, Jr. An extremal problem in recursive combinatorics. *Congr. Numer.*, 33:143–153, 1981.

[103] Sándor Kisfaludi-Bak, Jesper Nederlof, and Karol Wegrzycki. A Gap-ETH-Tight Approximation Scheme for Euclidean TSP. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 351–362. IEEE, 2022.

[104] Dusan Knop, Tomás Masarík, and Tomás Toufar. Parameterized complexity of fair vertex evaluation problems. In *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[105] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, page 1272–1287, USA, 2016. Society for Industrial and Applied Mathematics.

[106] Tuukka Korhonen, Konrad Majewski, Wojciech Nadara, Michał Pilipczuk, and Marek Sokołowski. Dynamic treewidth, 2023.

[107] Irina Kostitsyna, Bettina Speckmann, and Kevin Verbeek. Non-crossing geometric steiner arborescences. In Yoshio Okamoto and Takeshi Tokuyama, editors, *Proc. 28th Int. Symp. Algorithms & Computation (ISAAC'17)*, volume 92 of *LIPIcs*, pages 54:1–54:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

[108] Jan Kratochvíl and András Sebö. Coloring precolored perfect graphs. *J. Graph Theory*, 25:207–215, 1997.

[109] Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1789–1799. SIAM, 2013.

[110] Robert Krauthgamer and Inbal Rika. Refined vertex sparsifiers of planar graphs. *CoRR*, abs/1702.05951, 2017.

[111] Yoshiyuki Kusakari, Daisuke Masubuchi, and Takao Nishizeki. Finding a noncrossing Steiner forest in plane graphs under a 2-face condition. *J. Comb. Optim.*, 5(2):249–266, 2001.

[112] Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *J. Comput. Syst. Sci.*, 37(1):63–78, 1988.

[113] Thomas M. Liebling, François Margot, Didier Müller, Alain Prodon, and Lynn Stauffer. Disjoint paths in the plane. *ORSA J. Comput.*, 7(1):84–88, 1995.

[114] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM J. Comput.*, 47(3):675–702, 2018.

[115] Bin Ma and Xiaoming Sun. More efficient algorithms for closest string and substring problems. *SIAM J. Comput.*, 39(4):1432–1443, 2009.

[116] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 253–262. IEEE Computer Society, 2013.

[117] Konrad Majewski, Michał Pilipczuk, and Marek Sokołowski. Maintaining CMSO2 Properties on Dynamic Structures with Bounded Feedback Vertex Number. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:13, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[118] Dániel Marx. Parameterized coloring problems on chordal graphs. *Theoret. Comput. Sci.*, 351(3):407–424, 2006.

[119] Dániel Marx. *Precoloring Extension on Chordal Graphs*, pages 255–270. 12 2006.

[120] Dániel Marx. Precoloring extension on unit interval graphs. *Discrete Applied Mathematics*, 154(6):995–1002, 2006.

[121] Tomáš Masařík and Tomáš Toufar. Parameterized complexity of fair deletion problems. *Discrete Applied Mathematics*, 278:51–61, 2020.

[122] William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980.

[123] Cristian S. Mata and Joseph S. B. Mitchell. Approximation algorithms for geometric tour and network design problems (extended abstract). In Jack Snoeyink, editor, *Proceedings of the Eleventh Annual Symposium on Computational Geometry, Vancouver, B.C., Canada, June 5-12, 1995*, pages 360–369. ACM, 1995.

[124] Robert McNaughton and Seymour Papert. *Counter-free automata*. MIT Press, 1971.

[125] Nicole Megow, Martin Skutella, Jose Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. In *ICALP*, pages 689–700. 2012.

[126] Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput*, 28:402–408, 1996.

[127] Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, page 248–255, USA, 2004. IEEE Computer Society.

[128] Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[129] Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity — Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

[130] Evanthia Papadopoulou. k-Pairs non-crossing shortest paths in a simple polygon. *Int. J. Comput. Geom. Appl.*, 9(6):533–552, 1999.

[131] Valentin Polishchuk and Joseph S. B. Mitchell. Thick non-crossing paths and minimum-cost flows in polygonal domains. In Jeff Erickson, editor, *Proceedings of the 23rd ACM Symposium on Computational Geometry, Gyeongju, South Korea, June 6-8, 2007*, pages 56–65. ACM, 2007.

[132] Valentin Polishchuk and Joseph S. B. Mitchell. Thick non-crossing paths and minimum-cost flows in polygonal domains. In *Proc. 23rd ACM Symp. Comput. Geom. (SoCG'07)*, pages 56–65, 2007.

[133] Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In *Proceedings of the 41st International Colloquium Automata, Languages, and Programming, ICALP 2014*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014.

[134] Iris Reinbacher, Marc Benkert, Marc van Kreveld, Joseph SB Mitchell, Jack Snoeyink, and Alexander Wolff. Delineating boundaries for imprecise regions. *Algorithmica*, 50(3):386–414, 2008.

[135] George Retsinas, Georgios Louloudis, Nikolaos Stamatopoulos, and Basilis Gatos. Efficient document image segmentation representation by approximating minimum-link polygons. In *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*, pages 293–298. IEEE, 2016.

[136] Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM J. Discret. Math.*, 19(1):122–134, May 2005.

[137] Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 118–126. SIAM, 2007. `http://dl.acm.org/citation.cfm?id=1283383.1283397`.

[138] Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control.*, 8(2):190–194, 1965.

[139] Shay Solomon. Fully dynamic maximal matching in constant update time.

[140] Shay Solomon and Nicole Wein. Improved dynamic graph coloring. In *26th European Symposium on Algorithms*, volume 112 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 72, 16. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.

[141] Andreas Sprock. *Analysis of hard problems in reoptimization and online computation*. PhD thesis, ETH Zurich, Zürich, Switzerland, 2013.

[142] J. Takahashi, H. Suzuki, and T. Nishizeki. Algorithms for finding non-crossing paths with minimum total length in plane graphs. In *International Symposium on Algorithms and Computation*, pages 400–409. Springer, 1992.

[143] J. Takahashi, H. Suzuki, and T. Nishizeki. Finding shortest non-crossing rectilinear paths in plane regions. In *International Symposium on Algorithms and Computation*, pages 98–107. Springer, 1993.

[144] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51:993–1024, 2004.

[145] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

[146] Godfried T Toussaint. *An optimal algorithm for computing the relative convex hull of a set of points in a polygon*. MacGill University. School of Computer Science, 1986.

[147] Mario E. Valencia-Pabon. Revisiting tucker's algorithm to color circular-arc graphs. *Electronic Notes in Discrete Mathematics*, 7:198–201, 2001. Brazilian Symposium on Graphs, Algorithms and Combinatorics.

[148] Yajun Wang and Sam Chiu-wai Wong. Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming*, pages 1070–1081, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[149] A.Z. Zelikovsky. An 11/6-approximation algorithm for the steiner problem on graphs. In Jaroslav Neŝetril and Miroslav Fiedler, editors, *Fourth Czechoslovakian Symposium on Combinatorics, Graphs and Complexity*, volume 51 of *Annals of Discrete Mathematics*, pages 351–354. Elsevier, 1992.

[150] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *STOC'06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 681–690. ACM, New York, 2006.