Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Tomasz Waleń

# Algorytmy dokładne i aproksymacyjne dla wybranych problemów kombinatorycznych w biologii obliczeniowej

*rozprawa doktorska*

Promotor rozprawy

prof. dr hab. Krzysztof Diks

Instytut Informatyki
Uniwersytet Warszawski

Listopad 2009

Oświadczenie autora rozprawy:
oświadczam, że niniejsza rozprawa została napisana przeze mnie samodzielnie.

Listopad 10, 2009 . . . . . .
*data* *Tomasz Waleń*

Oświadczenie promotora rozprawy:
niniejsza rozprawa jest gotowa do oceny przez recenzentów.

Listopad 10, 2009 . . . . . .
*data* *prof. dr hab. Krzysztof Diks*

## Streszczenie

W niniejszej rozprawie zajmujemy się zagadnieniami algorytmicznymi inspirowanymi problemami kombinatorycznymi rozważanymi w biologii obliczeniowej.

Pierwsza część rozprawy jest poświęcona problemowi wyznaczania minimalnego wspólnego podziału słów (problem MCSP). Problem ten pozwala na zdefiniowanie pewnej miary podobieństwa pomiędzy słowami i jest blisko związany ze znanym problemem SBR — sortowania przez odwracanie. Głównym wynikiem tego rozdziału jest algorytm $O(k)$–aproksymacyjny dla problemu MCSP, gdzie $k$ oznacza maksymalną liczbę powtórzeń pojedynczego symbolu w słowach wejściowych.

W drugiej części rozprawy rozważamy problem MAX-NLS, który polega na wyznaczeniu dla zadanego zbioru grafów uliniowionych ich najliczniejszego wspólnego podgrafu. Problem ten polega na badaniu podobieństwa pomiędzy grafami, a w biologii obliczeniowej może być wykorzystywany do badania podobieństwa przestrzennych struktur cząsteczek ncRNA. Głównym wynikiem jest algorytm $O(\log m_{opt})$–aproksymacyjny i działający w czasie $O(kn^2)$, gdzie $k$ jest liczbą wejściowych grafów, $n$ — maksymalną liczbą wierzchołków w grafie, natomiast $m_{opt}$ oznacza liczbę krawędzi w optymalnym rozwiązaniu.

W trzeciej części rozprawy badamy problem wyznaczania uwarunkowanych cykli Eulera. Problem ten polega na obliczeniu czy graf zawiera cykl Eulera spełniający dodatkowe warunki, to znaczy, czy zawiera (lub omija) zadany zbiór ścieżek. Podajemy liniowy algorytm, który dla grafu prostego $G$ oraz zbioru ścieżek $P$ stwierdza, czy graf posiada cykl Eulera zawierający każdą ze ścieżek ze zbioru $P$, oraz dowodzimy, że problem jest NP-zupełny w przypadku multigrafów. Podajemy również liniowy algorytm, który dla prostego grafu $G$ oraz zabronionej ścieżki $\pi$ stwierdza, czy graf posiada cykl Eulera unikający ścieżki $\pi$. Problem uwarunkowanych cykli Eulera pojawił się w biologii obliczeniowej jako element analizy metody sekwencjonowania DNA przez hybrydyzację.



*Słowa kluczowe:* sortowanie przez odwracanie, SBR, MCSP, grafy uliniowione, uwarunkowane cykle Eulera, złożoność obliczeniowa algorytmów, algorytmy aproksymacyjne

*Klasyfikacja tematyczna ACM:* F.2.2, G.2.1, G.2.2

## Abstract

In this thesis we discuss algorithmic challenges inspired by combinatorial problems studied in computational biology.

In the first chapter we study the Minimum Common String Partition Problem (MCSP). This problem defines a distance measurement between words and it is closely related to another famous problem — SBR (sorting by reversals). The main result of this chapter is $O(k)$-approximation algorithm for the MCSP problem, in which each symbol is allowed to appear up to $k$ times in each word.

The second chapter is devoted to considerations related to problem MAX-NLS — the problem of finding the largest nested linear graph that occurs in a given set of linear graphs. It can be used for finding similarities between secondary structures of ncRNA molecules. The main result of this chapter is $O(\log m_{opt})$-approximation algorithm running in $O(kn^2)$ time, where $k$ is the number of input graphs, $n$ the maximal number of vertices in a graph, and $m_{opt}$ is the size of the optimal solution.

In the third chapter we present a problem of finding an Eulerian cycle satisfying some constraints, i.e. a cycle that includes (or excludes) a given set of paths. We propose a linear-time algorithm for computing an Eulerian cycle in an arbitrary simple graphs that includes a given set of paths, and prove that this problem is NP-complete for multigraphs. We also study some of the "negative" constraints. We show a linear-time algorithm for computing an Eulerian cycle that excludes a given path.



*Keywords:* sorting by reversals, SBR, minimum common string partition, MCSP, linear graphs, complexity of algorithms, approximating algorithms

*ACM Classification:* F.2.2, G.2.1, G.2.2

# Spis treści

# Wstęp

W niniejszej rozprawie zajmujemy się zagadnieniami algorytmicznymi inspirowanymi problemami kombinatorycznymi rozważanymi w biologii obliczeniowej. W biologii obliczeniowej kładzie się główny nacisk na takie wykorzystanie nowoczesnych metod i narzędzi obliczeniowych, aby umożliwiały nowe odkrycia biologiczne. Stąd w naturalny sposób szczególna uwaga jest poświęcona praktycznym zastosowaniom badań. W naszej pracy skupiamy się głównie na teoretycznych aspektach rozważanych problemów, rozważając ich bardzo uproszczone modele matematyczne. Można traktować takie dwa podejścia jako zupełnie przeciwstawne, jednak moim zdaniem uzupełniają się wzajemnie.

Większości problemów na które napotyka biologia obliczeniowa, nawet w przypadku zastosowania bardzo uproszczonego modelu matematycznego, pozostaje trudna do rozwiązania. Jednak bez dokładnego zbadania prostych wariantów tych problemów trudno marzyć o zdecydowanych postępach w rozwiązaniach bardziej złożonych wersji problemów. Okazuje się, że badając w ten sposób problemy z jednej dziedziny, możemy przypadkowo odkryć zastosowania w zupełnie odległych dziedzinach. Przykładowo, techniki, które służą do badania podobieństwa pomiędzy sekwencjami DNA, znalazły zastosowanie w wykrywaniu podobieństw w pracach studenckich, artykułach prasowych, itp.

Podobne rozumowanie można przeprowadzić w odwrotnym kierunku, wykazując, że dzięki praktycznym zastosowaniom możemy wskazać, które z problemów są kluczowe dla postępu badań.

W kolejnych rozdziałach skupiamy uwagą na problemach, których źródłem jest porównywanie różnego rodzaju struktur biologicznych.

W pierwszym rozdziale zajmujemy się problemem wyznaczania minimalnego wspólnego podziału słów (problem MCSP). Problem ten jest blisko związany z problemem SBR (sorting by reversals), który był wskazywany jako narzędzie pozwalające ocenić podobieństwo pomiędzy różnymi sekwencjami genów. Początkowo problem SBR był badany przy przyjęciu założenia, że każda sekwencja wejściowa jest permutacją (czyli każdy gen jest unikalny). Jednak takie założenie okazało się bardzo nienaturalne. Niestety usunięcie tego założenia bardzo utrudnia efektywne rozwiązywanie problemu SBR (nawet w przypadku alfabetu

składającego się tylko z dwóch znaków problem jest NP-trudny). Jedną z metod przezwyciężenia tych trudności było wprowadzenie problemu $k$-SBR, w którym każdy symbol może pojawić co najwyżej $k$ razy w słowach wejściowych (czyli każdy gen może wystąpić w co najwyżej $k$ wariantach). Głównym wynikiem tego rozdziału jest podanie algorytmu $O(k)$–aproksymacyjnego dla problemu $k$-MCSP, co w konsekwencji prowadzi do uzyskania algorytmu $O(k)$–aproksymacyjnego dla problemu $k$-SBR. Wynik ten został zaprezentowany podczas konferencji WAOA 2006 [28, 30].

W drugim rozdziale rozważamy problem MAX-NLS, który był próbą zdefiniowania innej miary podobieństwa. Większość typowych miar koncentruje się na analizie tekstowego opisu sekwencji, czy struktur. Okazuje się jednak, że niekiedy jest to bardzo mylące. Właściwsze wydaje się utożsamianie obiektów o podobnych kształtach. Motywacją w tym przypadku było badanie własności ncRNA (non-coding RNA), gdzie często różne sekwencje nukleotydów tworzą cząsteczki RNA o tym samym kształcie, a w konsekwencji pełnią podobną rolę. W problemie MAX-NLS wejściowe obiekty reprezentuje się jako grafy uliniowione (czyli grafy, których wierzchołki utożsamione są z kolejnymi liczbami całkowitymi), których krawędzie stanowią o kształcie obiektów. Rozwiązaniem problemu jest najliczniejszy wspólny podgraf będący emanacją wspólnych cech zadanych grafów. Jego rozmiar jest wyznacznikiem podobieństwa pomiędzy wejściowymi grafami. Głównym wynikiem jest algorytm $O(\log m_{opt})$–aproksymacyjny rozwiązujący problem MAX-NLS. Wynik ten został zaprezentowany podczas konferencji CPM 2006 [31].

W trzecim rozdziale rozważamy problemy wyznaczania uwarunkowanych cykli Eulera. Problem ten był inspirowany jedną z metod używanych do odczytywania sekwencji DNA. Rozważamy kilka problemów dotyczących cykli Eulera spełniających dodatkowe warunki. W szczególności problem obliczania, czy graf posiada cykl Eulera zawierający (lub w innym wariancie problemu, nie zawierający) zadany zbiór ścieżek. Podajemy liniowy algorytm, który dla prostego grafu $G$ oraz zbioru ścieżek $P$ stwierdza, czy graf posiada cykl Eulera zawierający każdą ze ścieżek ze zbioru $P$, oraz dowodzimy że problem jest NP-zupełny w przypadku multigrafów. Podajemy również liniowy algorytm, który dla prostego grafu $G$ oraz zabronionej ścieżki $\pi$ stwierdza, czy graf posiada cykl Eulera unikający ścieżki $\pi$.

# Podziękowania

Chciałbym bardzo gorąco podziękować mojemu promotorowi prof. Krzysztofowi Diksowi. Jestem mu głęboko wdzięczny za pomoc i wsparcie, których udzielił mi podczas pisania niniejszej rozprawy.

Równie gorąco dziękuję moim kolegom Petrowi Kolmanowi oraz Marcinowi Kubicy, z którymi wspólnie pracowałem nad problemami przedstawionymi w niniejszej pracy.

# Podstawowe pojęcia, oznaczenia i przyjęte założenia

W pracy posługujemy się oznaczeniami i pojęciami zdefiniowanymi w książce [12]. Dla pełności opisu, poniżej przedstawiamy jednak definicje najważniejszych używanych pojęć.

## Grafy

*Graf nieskierowany* definiujemy jako parę skończonych zbiorów $(\mathbf{V}, \mathbf{E})$. Elementy zbioru $\mathbf{V}$ nazywamy wierzchołkami. Zbiór $\mathbf{E}$ zawiera nieuporządkowane pary wierzchołków, czyli $\mathbf{E} \subseteq \{\{u, v\} : u, v \in \mathbf{V} \text{ i } u \neq v\}$. Elementy zbioru $\mathbf{E}$ nazywamy krawędziami. Dla grafu $G$, przez $\mathbf{V}(G)$ oznaczamy zbiór jego wierzchołków, a przez $\mathbf{E}(G)$ zbiór jego krawędzi. Jeśli bezpośrednio z kontekstu wynika o jakim grafie mowa, będziemy oznaczać zbiory jego wierzchołków i krawędzi odpowiednio przez $\mathbf{V}$ i $\mathbf{E}$.

W przypadku, gdy krawędziom dodatkowo przyporządkujemy kierunek, używamy pojęcia *grafu skierowanego*. W tym przypadku zbiór krawędzi $\mathbf{E}$ zawiera uporządkowane pary różnych wierzchołków, czyli $\mathbf{E} \subseteq \{(u, v) : u, v \in \mathbf{V} \text{ i } u \neq v\}$.

W przypadku grafów skierowanych, przez $indeg(v)$ i $outdeg(v)$ oznaczamy, odpowiednio, stopień wejściowy i wyjściowy wierzchołka $v \in \mathbf{V}$, czyli liczby krawędzi odpowiednio wchodzących do wierzchołka $v$, oraz krawędzi wychodzących z wierzchołka $v$. Zbiór wierzchołków sąsiadujących z $v \in \mathbf{V}$ oznaczamy przez $adj(v)$. Formalnie $adj(v) = \{u : \{v, u\} \in \mathbf{E}\}$ w przypadku grafów nieskierowanych i $adj(v) = \{u : (v, u) \in \mathbf{E} \text{ lub } (u, v) \in \mathbf{E}\}$ dla grafów skierowanych.

Niekiedy przydatne jest dopuszczenie wielokrotnych krawędzi w grafie. W tym wypadku piszemy, że mamy do czynienia z *multigrafem* (skierowanym lub nieskierowanym). W takim wypadku zamiast o zbiorze krawędzi piszemy o multizbiorze krawędzi.

Od tego miejsca, przez $(u, v)$ będziemy oznaczali zarówno krawędź w grafie

skierowanym, jak i nieskierowanym. W przypadku grafu nieskierowanego kolejność wierzchołków w parze $(u, v)$ jest nieistotna.

*Ścieżką* w grafie $G = (\mathbf{V}, \mathbf{E})$ nazywamy każdy skończony ciąg wierzchołków $\langle v_0, \ldots, v_k \rangle$ i taki, że $v_i \in \mathbf{V}$ oraz $(v_i, v_{i+1}) \in \mathbf{E}$, dla $i = 0, \ldots, k-1$. Jeden wierzchołek może występować wielokrotnie na ścieżce. *Długość ścieżki* definiujemy jako liczbę krawędzi w niej zawartych. *Cykl* to ścieżka $\langle v_0, \ldots, v_k, v_{k+1} = v_0 \rangle$, która rozpoczyna się i kończy w tym samym wierzchołku.

Piszemy, że nieskierowany graf $G$ jest *spójny*, jeśli dla dowolnych wierzchołków $x, y \in \mathbf{V}$, $x \neq y$, istnieje ścieżka pomiędzy $x$ i $y$.

Przez *odległość* $d(x, y)$ wierzchołków $x, y \in \mathbf{V}$, oznaczamy minimalną długość ścieżki łączącej $x$ i $y$, lub $+\infty$, jeśli nie istnieje żadna ścieżka pomiędzy $x$ i $y$.

Ścieżka jest *elementarna*, jeśli nie zawiera powtórzeń wierzchołków.

Cykl $\langle v_0, \ldots, v_k, v_{k+1} = v_0 \rangle$ jest *elementarny*, jeśli dla każdych $0 \leq i \neq j \leq k$ mamy $v_i \neq v_j$.

Piszemy, że nieskierowany graf $H = (\mathbf{V}', \mathbf{E}')$ jest *podgrafem* grafu $G = (\mathbf{V}, \mathbf{E})$, jeśli $\mathbf{V}' \subseteq \mathbf{V}$ oraz $\mathbf{E}' \subseteq \mathbf{E}$. Dla grafu $G = (\mathbf{V}, \mathbf{E})$ piszemy, że $H$ jest *podgrafem indukowanym* przez zbiór wierzchołków $X \subseteq V$, jeśli $\mathbf{V}(H) = X$, oraz $\mathbf{E}(H) = \{(u, v) : (u, v) \in \mathbf{E} \text{ oraz } u, v \in X\}$.

*Drzewem* nazywamy spójny graf $T$, który nie zawiera cykli. Wierzchołki drzewa będziemy nazywali *węzłami*. Drzewo z wyróżnionym węzłem nazywamy drzewem z korzeniem. Wyróżniony węzeł to właśnie korzeń drzewa. Niech $T$ będzie drzewem z korzeniem $r$. *Głębokością węzła* $v$ w drzewie nazywamy odległość pomiędzy korzeniem $r$ a węzłem $v$ i oznaczamy przez $d(v)$. Piszemy, że $x$ jest *ojcem* węzła $y$ w drzewie $T$, jeśli $x$ i $y$ są połączone krawędzią oraz $d(x) + 1 = d(y)$. Ojca węzła $x$ oznaczamy przez $parent(x)$. Dla korzenia wartość $parent$ nie jest zdefiniowana. Piszemy, że $y$ jest *synem* węzła $x$, jeśli $x$ i $y$ są połączone krawędzią oraz $d(x) + 1 = d(y)$. Zbiór wszystkich synów węzła $x$ oznaczamy przez $children(x) = \{y : y \text{ jest synem } x\}$. Piszemy, że $x$ jest *przodkiem* (*potomkiem*) węzła $y$, jeśli istnieje ścieżka pomiędzy $x$ i $y$, oraz $d(x) \leq d(y)$ $(d(x) \geq d(y))$.

*Drzewo rozpinające* nieskierowanego grafu $G$, to drzewo $T$ składające się z krawędzi grafu $G$ i takie, że $\mathbf{V}(T) = \mathbf{V}(G)$, $\mathbf{E}(T) \subseteq \mathbf{E}(G)$.

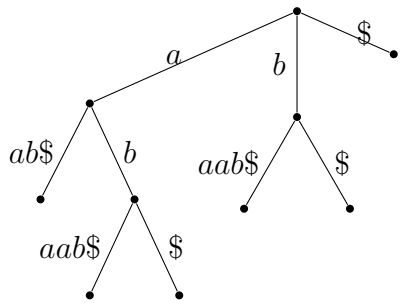
*Ścieżka Eulera* w grafie $G$, to ścieżka $P$, która zawiera każdą krawędź grafu $G$ dokładnie raz. *Cykl Eulera*, to ścieżka Eulera, która rozpoczyna się i kończy w tym samym wierzchołku. Piszemy, że graf jest *eulerowski*, jeśli istnieje w nim cykl Eulera.

# Słowa

**Słowem** nad skończonym alfabetem $\Sigma$ nazywamy dowolny skończony ciąg symboli z tego alfabetu. Dla słowa $s = a_1, a_2, \ldots, a_n$, przez $s[i]$ oznaczamy $i$–ty znak słowa $s$, czyli $a_i$, natomiast przez $|s|$ oznaczamy jego długość równą $n$. Słowo puste oznaczamy przez $\epsilon$. Oczywiście $|\epsilon| = 0$. Dla skończonego alfabetu $\Sigma$, przez $\Sigma^*$ oznaczamy zbiór wszystkich słów złożonych z symboli $\Sigma$, a przez $\Sigma^+$ oznaczamy ten sam zbiór z pominięciem słowa pustego (czyli $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$). *Konkatenację* (*złączenie*) dwóch słów $s_1$ i $s_2$ oznaczamy przez $s_1 s_2$. Piszemy, że $x$ jest *prefiksem* (*sufiksem*) słowa $s$, jeśli $s = xs'$ ($s = s'x$), dla pewnego słowa $s'$. O $x$ mówimy, że jest podsłowem słowa $s$, jeśli $s = s'xs''$, dla pewnych słów $s'$, $s''$, a fakt ten zapisujemy $x \sqsubseteq s$. Jeśli $x$ jest *podsłowem* $s$ oraz $x \neq s$, to używamy oznaczania $x \sqsubset s$. Piszemy, że słowo $s$ jest *nadsłowem* $x$, jeśli $s = s'xs''$, dla pewnych słów $s'$, $s''$. *Podsłowo* $s$ rozpoczynające się na pozycji $i$ i kończące na pozycji $j$ oznaczamy przez $s[i..j]$, dla $1 \leq i \leq j \leq n$. Liczbę wystąpień słowa $x$ w $s$ oznaczamy przez $\#_x(s)$, czyli $\#_x(s) = |\{i : 1 \leq i \leq |s|+1-|x|, \text{ oraz } s[i..(i+|x|-1)] = x\}|$

Drzewo sufiksowe dla danego słowa $w$, to drzewo z korzeniem, które w sposób efektywny reprezentuje zbiór wszystkich sufiksów $w$. Krawędzie drzewa sufiksowego są etykietowane podsłowami $w$. Każda ścieżka od korzenia do liścia w drzewie odpowiada pewnemu sufiksowi. Przykładowe drzewo sufiksowe zostało przedstawione na rysunku 1. Podstawową cechą drzewa sufiksowego jest fakt, że jego rozmiar jest liniowy względem długości słowa. Więcej informacji dotyczących drzew sufiksowych można odnaleźć w książce [15]. Istnieje wiele liniowych algorytmów konstrukcji drzew sufiksowych [35, 41] dla alfabetów o stałym rozmiarze, niezależnym od długości słowa. W ostatnim czasie zaproponowano również liniowe algorytmy do konstrukcji drzew sufiksowych w przypadku, gdy rozmiar alfabetu słowa jest wielomianowo ograniczony ze względu na rozmiar słowa (czyli np. $\Sigma = \{1..|w|^c\}$, dla pewnej stałej $c$) [18].

Dla węzła $v$ w drzewie sufiksowym, przez $L_v$ oznaczamy słowo otrzymane przez złączenie etykiet krawędzi na ścieżce od korzenia drzewa do węzła $v$.




Rysunek 1: Przykład drzewa sufiksowego dla słowa $w = abaab\$$.

# Rozdział 1

# Problem Minimalnego Wspólnego Podziału Słów

## 1.1 Wprowadzenie

W ostatnich latach wiele uwagi poświęcono metodom mierzenia podobieństwa pomiędzy słowami przy użyciu różnorodnych metryk. Wynika to głównie z zastosowania tego typu problemów w biologii obliczeniowej [4, 25]. Jedna z takich metod, to *sortowanie przez odwracanie* (ang. *Sorting by Reversals*, w skrócie SBR).

Problem sortowania przez odwracanie polega na obliczeniu dla zadanych słów $A$ i $B$, najkrótszej sekwencji odwróceń podsłów, które przekształcą słowo $A$ w $B$. Odwrócenie $\rho(i,j)$, $1 \leq i < j \leq n$, to operacja, która przekształca słowo $A = a_1 \dots a_n$ w słowo $A' = a_1 \dots a_{i-1} a_j a_{j-1} \dots a_i a_{j+1} \dots a_n$ (czyli operacja, która odwraca kolejność symboli podsłowa $a_i \dots a_j$ w słowie $A$). Najkrótszą sekwencję odwróceń przkształcającą $A$ w $B$ nazywamy *odległością* pomiędzy tymi słowami.

Istnieje również wersja tego problemu, w której każdy symbol posiada dodatkowo znak $+$ lub $-$, a odwrócenie zamienia znaki na przeciwne.

Większość prac dotyczących problemu SBR odnosi się do przypadku, gdy zadane słowa są permutacjami. Dla permutacji bez znaków problem SBR jest NP-trudny [7]. Co zadziwiające, dla permutacji ze znakami problem jest znacznie prostszy. Istnieje wielomianowy algorytm rozwiązujący problem SBR dla permutacji ze znakami [25]. Odległość pomiędzy dwiema permutacjami ze znakami można wyznaczyć w czasie liniowym [2, 3], natomiast pełne rozwiązanie (czyli także najkrótszą sekwencję odwóceń) otrzymuje się w czasie $\mathcal{O}(n^{3/2} \log n)$ [40].

W przypadku gdy pozwalamy na powtórzenia symboli w słowach wejściowych, problem staje się znacznie trudniejszy obliczeniowo. Christie i Irving [9] pokazali, że problem SBR jest NP-trudny nawet dla słów nad alfabetem binar-

nym. Najlepszy algorytm aproksymacyjny dla ogólnego problemu SBR ma współczynnik aproksymacji $\mathcal{O}(\log n \log^* n)$ [13, 14].

Wariant pośredni problemu to $k$-SBR, w którym zakłada się, że w w każdym z wejściowych słów, każdy symbol występuje co najwyżej $k$ razy. Oczywiście 1-SBR, to oryginalny problem SBR, w którym słowa wejściowe są permutacjami. Dla $k = 2$, problem 2-SBR jest NP-trudny [8], jednak w tym przypadku istnieją efektywne algorytmy aproksymacyjne. Dla problemów 2-SBR i 3-SBR istnieją algorytmy $\mathcal{O}(1)$-aproksymacyjne [8, 11, 22].

W tym rozdziale przedstawiamy dwa algorytmy aproksymacyjne dla problemu $k$-SBR. W obu przypadkach wykorzystujemy bezpośrednio aproksymacyjne rozwiązania problemu minimalnego wspólnego podziału słów (ang. *Minimal Common String Partition*, w skrócie MCSP), który definiujemy w dalszej części rozdziału. Rozpoczynamy od algorytmu $\mathcal{O}(k^2)$–aproksymacyjnego dla problemu $k$-SBR, zaproponowanego przez Kolmana w pracy [27]. Algorytm ten jest modyfikacją algorytmu zachłannego i działa w czasie $\mathcal{O}(kn)$. Następnie przedstawiamy efektywną implementację tego algorytmu działającą w czasie liniowym ze względu na rozmiar danych wejściowych, niezależnym od $k$. W dalszej częsci rozdziału omawiamy algorytm $\mathcal{O}(k)$–aproksymacyjny, w którego analizie wykorzystujemy analizę problemu *Minimum Hitting Set*. Nasza implementacja tego algorytmu działa w czasie liniowym. Oba powyższe algorytmy są wynikiem współpracy z P. Kolmanem i zostały opublikowane w pracach [28, 29, 30].

## 1.2 Definicje

Przypomnijmy, dla danego słowa $S = a_1 \ldots, a_n$ wynikiem operacji $\rho(i, j)$, $1 \leq i \leq j \leq n$, jest słowo $S' = a_1..a_{i-1}\ a_j a_{j-1}..a_i\ a_{j+1}..a_n$. Przykładowo, wynikiem operacji $\rho(2, 4)$ na słowie $S = \texttt{abcdef}$ jest słowo $S' = \texttt{adcbef}$. Jeśli symbole słowa $S$ dodatkowo posiadają znaki $\{+, -\}$, to operacja $\rho(i, j)$ zamienia znaki symboli $a_i..a_j$ na przeciwne. Na przykład, wynikiem operacji $\rho(2, 4)$ na słowie $S = \texttt{+a-b+c+d+e-f}$ jest słowo $S' = \texttt{+a-d-c+b+e-f}$.

Dla danego słowa $P = a_1 \ldots a_n$, przez $-P$ oznaczamy jego odwrócenie, czyli wynik operacji $\rho(1, n)$ na słowie $P$.

Słowa $A$ i $B$ są *zgodne*, jeśli każdy znak alfabetu $c \in \Sigma$ występuje tyle samo razy w $A$ co w $B$, czyli $\#_c(A) = \#_c(B)$, dla każdego $c \in \Sigma$.

*Podziałem słowa* $A$ nazywamy taki ciąg niepustych słów $\mathcal{A} = (A_1, \ldots, A_p)$, dla którego $A = A_1 A_2 \ldots A_p$. Słowa $A_i$ podziału $\mathcal{A}$ nazywamy *blokami*. Liczbę bloków w podziale zapisujemy jako $|\mathcal{A}|$, $i$-ty blok podziału zapisujemy jako $\mathcal{A}(i)$.

Słowo $\delta = c_1 c_2$, $c_1, c_2 \in \Sigma$, składające się z dokładnie dwóch symboli, nazywamy *duetem*. Dla słowa $A$, przez $duos(A)$ oznaczamy zbiór wszystkich duetów

zawartych w $A$. Przykładowo, jeśli $A = ababc$, to $duos(A) = \{ab, ba, bc\}$. Analogicznie, dla podziału $\mathcal{A}$, przez $duos(\mathcal{A})$ oznaczamy zbiór $\bigcup_{A_i \in \mathcal{A}} duos(A_i)$.

Dla słowa $A = a_1, \ldots, a_n$ oraz duetu $\delta = c_1c_2$, definiujemy operację *cięcia* słowa przy użyciu duetu $\delta$, $\textsc{Cut}(A, \delta)$, jako najmniej liczny podział $\mathcal{A} = (A_1, A_2, \ldots, A_p)$ słowa $A$, taki, że żadne ze słów $A_i$ nie zawiera duetu $\delta$ jako podsłowa. Przykładowo, jeśli $A = abbabab$, $\delta = ba$, to $\textsc{Cut}(A, \delta) = (abb, ab, ab)$. Analogicznie definiujemy operację CUT dla podziału słowa. Dla $\mathcal{A} = A_1 .., A_p$ i duetu $\delta$, wynikiem operacji $\textsc{Cut}(\mathcal{A}, \delta)$ jest podział $\textsc{Cut}(A_1, \delta)+\ldots+\textsc{Cut}(A_p, \delta)$, gdzie $+$ oznacza operację sklejenia dwóch ciągów. Przykładowo, jeśli $\mathcal{A} = (abb, ab, ab)$, $\delta = ab$, to $\textsc{Cut}(\mathcal{A}, \delta) = (a, bb, a, b, a, b)$.

Pojęcie cięcia rozszerzamy również do cięcia przy użyciu zbioru duetów $\Phi$, co w przypadku słowa $A$ (podziału $\mathcal{A}$) oznacza podział otrzymany przez zastosowanie kolejnych cięć duetami $\delta \in \Phi$. Przykładowo, jeśli $A = abbabab$, $\Phi = \{ab, ba\}$, to $\textsc{Cut}(A, \Phi) = (a, bb, a, b, a, b)$.

Dla danego podziału $\mathcal{A} = (A_1, \ldots, A_p)$ jego zbiorem cięć $\textsc{Bord}(\mathcal{A})$ nazywamy zbiór duetów "leżących" pomiędzy kolejnymi blokami, czyli powstałych przez złączenie ostatniej litery bloku i pierwszej litery następnego bloku. Przykładowo, dla podziału $\mathcal{A} = (abba, bcd, a, c)$, otrzymujemy $\textsc{Bord}(\mathcal{A}) = \{ab, da, ac\}$.

W tym rozdziale zajmujemy się następującymi problemami:

**Problem 1.1** (SBR)

**Dane:** *Dwa zgodne słowa $A$, $B$.*

**Wynik:** *Najkrótsza sekwencja operacji $\rho(l_1, r_1), \ldots, \rho(l_m, r_m)$, która przekształca słowo $A$ w $B$.*

**Problem 1.2** (MCSP)

**Dane:** *Dwa zgodne słowa $A$, $B$.*

**Wynik:** *Najmniej liczne podziały słów $A$ i $B$, $\mathcal{A} = A_1, \ldots A_m$, $\mathcal{B} = B_1, \ldots B_m$, takie, że istnieje bijekcja $\phi : \{1, \ldots, m\} \to \{1, \ldots, m\}$, dla której $A_i = B_{\phi(i)}$.*

Rozważamy również dwa warianty problemu MCSP, w których uznajemy bloki $X$ i $Y$ za równoważne, gdy $X = Y$ lub $X = -Y$.

**Problem 1.3** (RMCSP)

**Dane:** *Dwa zgodne słowa $A$, $B$.*

**Wynik:** *Najmniej liczne podziały słów $A$ i $B$, $\mathcal{A} = A_1, \ldots A_m$, $\mathcal{B} = B_1, \ldots B_m$, takie, że istnieje bijekcja $\phi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, dla której $A_i = B_{\phi(i)}$ lub $A_i = -(B_{\phi(i)})$.*

**Problem 1.4** (SMCSP)

**Dane:** *Dwa zgodne słowa $A$, $B$, w których każdy symbol dodatkowo posiada znak $\{+, -\}$.*

**Wynik:** *Analogiczny podział jak w przypadku problemu* RMCSP *(przyjmujemy, że operacja odwracania zmienia znaki symboli w odwracanych podsłowach na przeciwne).*

**Problem 1.5** (SSBR)

**Dane:** *Dwa zgodne słowa $A$, $B$, w który każdy symbol dodatkowo posiada znak $\{+, -\}$.*

**Wynik:** *Analogiczna sekwencja operacji jak w przypadku problemu* SBR*, przy założeniu zmiany znaków symboli odwracanych słów.*

W niniejszym rozdziale będziemy zajmować się wariantami powyższych problemów ograniczonymi do przypadków, w których każdy symbol $c \in \Sigma$ występuje co najwyżej $k$ razy w każdym ze słów wejściowych.

**Problem 1.6** ($k$-SBR)

**Dane:** *Dwa zgodne słowa $A$, $B$ i takie, że $\#_c(A) = \#_c(B) \leq k$, dla każdego $c \in \Sigma$.*

**Wynik:** *Analogiczna sekwencja jak w przypadku problemu* SBR.

**Problem 1.7** ($k$-MCSP)

**Dane:** *Dwa zgodne słowa $A$, $B$ i takie, że $\#_c(A) = \#_c(B) \leq k$, dla każdego $c \in \Sigma$.*

**Wynik:** *Analogiczny podział jak w przypadku problemu* MCSP.

Tabela z rysunku 1.1 zawiera podsumowanie wyników dotyczących różnych wariantów problemu SBR.

Problem SMCSP został wprowadzony przez Chen'a [8] jako narzędzie pomagające rozwiązywać problem SBR. Autorzy zauważyli, że dla zgodnych słów (ze znakami) $A$ i $B$ rozwiązania SMCSP$(A, B)$ i SBR$(A, B)$ różnią się o stały współczynnik multiplikatywny.

**Lemat 1.2.1 ([8])** *Niech $A$ i $B$ będą zgodnymi słowami (ze znakami) i niech $\alpha$ będzie rozmiarem optymalnego rozwiązania problemu* SMCSP$(A, B)$*, a $\beta$ rozmiarem optymalnego rozwiązania problemu* SSBR$(A, B)$*. Wówczas*

$$\lceil(\alpha - 1)/2\rceil \leq \beta \leq \alpha - 1$$

Z dowodu lematu 1.2.1 wynika następująca obserwacja:

| **Problem** | **Współczynnik aproksymacji** | **Złożoność czasowa** |
|---|---|---|
| SBR | $\mathcal{O}(\log n \log^* n)$ | $\mathcal{O}(n \log^* n)$ [13, 14] |
| 2-SBR | 2.2074 | P [21] |
| 3-SBR | 8 | $\mathcal{O}(n)$ [21] |
| $k$-SBR | $\mathcal{O}(k^2)$ | $\mathcal{O}(kn)$ [27] |
| (nasze wyniki) | $\mathcal{O}(k^2)$ | $\mathcal{O}(n)$ [29] |
| | $\mathcal{O}(k)$ | $\mathcal{O}(n)$ [28, 30] |

Rysunek 1.1: Podsumowanie znanych wyników dla problemu SBR.

**Lemat 1.2.2** *Przy pomocy $\delta$–aproksymacyjnego algorytm dla problemu* SMCSP *można skonstruować $2\delta$–aproksymacyjny algorytm dla problemu* SSBR*, działający w tym samym czasie (asymptotycznie), co algorytm dla problemu* SMCSP.

**Dowód:** Dla danych słów $A$ i $B$, niech $\alpha$ oznacza rozmiar optymalnego rozwiązania dla problemu SMCSP$(A, B)$.

Optymalne rozwiązanie problemu SMCSP definiuje pewne podziały słów $A$ i $B$, oraz bijekcję pomiędzy blokami podziałów, którą możemy utożsamić z permutacjami ze znakiem $\pi^A$ i $\pi^B$ (długości $\alpha$). Ponieważ dla każdych dwóch permutacji ze znakiem SSBR$(\pi_1, \pi_2) \leq |\pi_1|$, to istnieje sekwencja operacji $\rho$, która przekształca permutację $\pi^A$ w $\pi^B$ długości $\alpha^* \leq \alpha$. Tą samą sekwencję operacji $\rho$ (przy czym operujemy już w tym przypadku na całych blokach) można zastosować do przekształcenia napisu $A$ w $B$. Czyli otrzymujemy rozwiązanie dla problemu SSBR o rozmiarze $\alpha^* \leq \alpha$.

Korzystając z lematu 1.2.1 dostajemy, że otrzymane rozwiązanie dla problemu SSBR jest co najwyżej dwa razy gorsze od optymalnego. ■

W analogiczny sposób możemy pokazać związek problemów RMCSP i SBR.

**Lemat 1.2.3** *Niech $A$ i $B$ będą zgodnymi słowami $A$ i $B$ i niech $\alpha$ będzie rozmiarem optymalnego rozwiązania problemu* RMCSP$(A, B)$*, a $\beta$ rozmiarem optymalnego rozwiązania problemu* SBR$(A, B)$*. Wówczas*

$$\lceil(\alpha - 1)/2\rceil \leq \beta \leq 2(\alpha - 1)$$

**Wniosek 1.2.4** *Dla dowolnego $\alpha$–aproksymacyjnego algorytmu rozwiązującego problem* RMCSP *można skonstruować $4\alpha$–aproksymacyjny algorytm dla problemu* SBR*, działający w tym samym czasie (asymptotycznie), co algorytm dla problemu* RMCSP.

## 1.3 Algorytm $\mathcal{O}(k^2)$–aproksymacyjny

W tym podrozdziale opisujemy algorytm $\mathcal{O}(k^2)$–aproksymacyjny dla problemu MCSP. Jest on sprytną modyfikacją algorytmu zachłannego.

W podrozdziale 1.3.1 krótko przedstawiamy algorytm zachłanny. Następnie w podrozdziale 1.3.2 opisujemy modyfikację algorytmu zachłannego, dzięki której otrzymujemy wspomniany współczynnik aproksymacji. W podrozdziale 1.3.3 proponujemy liniową implementację algorytmu aproksymacyjnego.

### 1.3.1 Algorytm zachłanny

Dla zadanych dwóch zgodnych słów $A$ i $B$ algorytm zachłanny rozpoczyna obliczenia od podziałów $\mathcal{A} = (A)$ i $\mathcal{B} = (B)$. Następnie tak długo, aż nie znajdzie żądanej bijekcji pomiędzy aktualnymi podziałami $(\mathcal{A}, \mathcal{B})$, stara się je dalej podzielić. W trakcie działania algorytmu każdy blok ma status niezaznaczony lub zaznaczony. Zaznaczone bloki to słowa, które mają już swoje odpowiedniki w drugim podziale. Algorytm zachłannie wybiera najdłuższe słowo $S$, które jest podsłowem pewnych niezaznaczonych bloków $A_i \in \mathcal{A}$ i $B_j \in \mathcal{B}$, dzieli bloki $A_i$ i $B_j$ tak, by słowo $S$ było osobnym blokiem w każdym podziale i następnie zaznacza $S$ w obu podziałach. Formalny zapis algorytm znajduje się na rysunku 1.2.

**Algorytm 1.3.1**: GREEDY

**Dane**: zgodne słowa $A$ i $B$
**Wynik**: zgodne podziały $(\mathcal{A}, \mathcal{B})$ słów $A$ i $B$

$\mathcal{A} \leftarrow (A)$; $\mathcal{B} \leftarrow (B)$
bloki podziałów $\mathcal{A}$ i $\mathcal{B}$ otrzymują status niezaznaczone
**while** *w $\mathcal{A}$ istnieje niezaznaczony blok* **do**
    niech $S$ najdłuższe słowo takie, że $S$ jest podsłowem pewnego niezaznaczonego bloku $A_i \in \mathcal{A}$, oraz pewnego niezaznaczonego bloku $B_j \in \mathcal{B}$
    niech $A_j = A'A''A'''$ oraz $B_j = B'B''B'''$, gdzie $A'' = B'' = S$
    $\mathcal{A} =$ zastąp blok $A_i$ w $\mathcal{A}$ przez $(A', A'', A''')$
    $\mathcal{B} =$ zastąp blok $B_j$ w $\mathcal{B}$ przez $(B', B'', B''')$
    zaznacz bloki $A''$ i $B''$
    usuń puste bloki z podziałów
**end**
**return** $(\mathcal{A}, \mathcal{B})$

Rysunek 1.2: Algorytm zachłanny dla problemu MCSP.

Niestety dla $k = 4$ współczynnik aproksymacji algorytmu GREEDY wynosi $\Omega(\log n)$ [10]. Natomiast dla dowolnego $k$ ma on wartość co najwyżej $\mathcal{O}(n^{0.69})$, ale nie mniejszą niż $\Omega(n^{0.43})$ [10].

### 1.3.2 Algorytm REFINED GREEDY

W tym podrozdziale opisujemy prostą modyfikację algorytmu GREEDY. Nowy algorytm, o nazwie REFINED GREEDY, rozwiązuje problem $k$-MCSP ze współczynnikiem aproksymacji $\mathcal{O}(k^2)$. Opisujemy również sposób rozszerzenia algorytmu dla problemu $k$-MCSP, tak by otrzymać algorytmy dla problemów $k$-RMCSP i $k$-SMCSP. Dzięki temu, zgodnie z lematem 1.2.3 otrzymujemy algorytm $\mathcal{O}(k^2)$-aproksymacyjny dla problemu $k$-SBR.

Niech $S = a_i \dots a_j$ będzie podsłowem $A = a_1 \dots a_n$. Jeśli $i > 1$, to podsłowo $a_{i-1}a_i$ nazywamy *(lewym) granicznym* duetem słowa $S$. Dla $i = 1$, lewy graniczny duet słowa $S$ nie istnieje. Podobnie, jeśli $j < n$, to $a_j a_{j+1}$ nazywamy *(prawym) granicznym* duetem słowa $S$. Analogicznie, jeśli $j = n$, to prawy graniczny duet słowa $S$ nie istnieje. Pojęcie granicznych duetów rozszerzamy również na podziały. Dla słowa $S$ będącego podsłowem bloku $A_i$ z podziału $\mathcal{A} = (A_1, \dots, A_k)$, zbiorem granicznych duetów wystąpienia $S$ w $\mathcal{A}$ nazywamy zbiór granicznych duetów $S$ w $A_i$.

W pracy [27] Petr Kolman zaproponował następujący algorytm dla problemu $k$-MCSP.

**Algorytm 1.3.2**: REFINED GREEDY

**Dane**: zgodne słowa $A$ i $B$
**Wynik**: zgodne podziały $(\mathcal{A}, \mathcal{B})$ słów $A$ i $B$

$\mathcal{A} \leftarrow (A)$, $\mathcal{B} \leftarrow (B)$
słowa w podziałach otrzymują status niezaznaczonych
**while** *istnieją niezaznaczone bloki w* $\mathcal{A}$ *i* $\mathcal{B}$ **do**
    $S \leftarrow$ najdłuższe wspólne podsłowo dwóch niezaznaczonych bloków z $\mathcal{A}$ i $\mathcal{B}$
    niech $S^A$ będzie dowolnym wystąpieniem $S$ w $\mathcal{A}$, a $S^B$ dowolnym wystąpieniem $S$ w $\mathcal{B}$
    zaznacz $S^A$ w $\mathcal{A}$ oraz $S^B$ w $\mathcal{B}$
    niech $\Phi$ będzie zbiorem zawierającym graniczne duety $S^A$ w $\mathcal{A}$ oraz graniczne duety $S^B$ w $\mathcal{B}$
    $\mathcal{A} \leftarrow \text{CUT}(\mathcal{A}, \Phi)$
    $\mathcal{B} \leftarrow \text{CUT}(\mathcal{B}, \Phi)$
**end**
**return** $(\mathcal{A}, \mathcal{B})$

Rozważmy przykład działania algorytmu dla danych:

$$A = abxyuvafxyuvddddhefxyuvebxyuvgggg$$
$$B = abxyuvddddafxyuvhefxyuvggggebxyuv$$

Algorytm REFINED GREEDY w pierwszym kroku zaznaczy podsłowo $S_1 = xyuvdddd$ oraz dokona przecięć duetów ze zbioru $\Phi_1 = \{fx, dh, bx, da\}$. Otrzymujemy w ten sposób podziały (zaznaczone bloki są podkreślone):

$$\mathcal{A} = (ab,\ xyuvaf,\ \overline{xyuvdddd},\ hef,\ xyuveb,\ xyuvgggg)$$
$$\mathcal{B} = (ab,\ \overline{xyuvdddd},\ af,\ xyuvhef,\ xyuvggggeb,\ xyuv)$$

W następnym kroku najdłuższym wspólnym podsłowem podziałów $\mathcal{A}, \mathcal{B}$ jest $S_2 = xyuvgggg$, a cięcia dokonane zostaną dla duetów ze zbioru $\Phi_2 = \{ge\}$. Nowe podziały to:

$$\mathcal{A} = (ab,\ xyuvaf,\ \overline{xyuvdddd},\ hef,\ xyuveb,\ \overline{xyuvgggg})$$
$$\mathcal{B} = (ab,\ \overline{xyuvdddd},\ af,\ xyuvhef,\ \overline{xyuvgggg},\ eb,\ xyuv)$$

W trzecim kroku najdłuższym wspólnym podsłowem podziałów $\mathcal{A}, \mathcal{B}$ jest $S_3 = xyuv$, a zbiór granicznych duetów to $\Phi_3 = \{va, vh\}$.

$$\mathcal{A} = (ab,\ \overline{xyuv},\ af,\ \overline{xyuvdddd},\ hef,\ xyuveb,\ \overline{xyuvgggg})$$
$$\mathcal{B} = (ab,\ \overline{xyuvdddd},\ af,\ \overline{xyuv},\ hef,\ \overline{xyuvgggg},\ eb,\ xyuv)$$

W ostatnim kroku, algorytm REFINED GREEDY wybiera $S_4 = xyuv$, oraz zbiór $\Phi_4 = ve$, wyznaczając w ten sposób następujący wspólny podział:

$$\mathcal{A} = (ab,\ xyuv,\ af,\ xyuvdddd,\ hef,\ xyuv,\ eb,\ xyuvgggg)$$
$$\mathcal{B} = (ab,\ xyuvdddd,\ af,\ xyuv,\ hef,\ xyuvgggg,\ eb,\ xyuv)$$

Optymalne rozwiązanie składa się z 6 bloków:

$$\mathcal{A}_{\text{OPT}} = (abxyuv,\ afxyuv,\ dddd,\ hefxyuv,\ ebxyuv,\ gggg)$$
$$\mathcal{B}_{\text{OPT}} = (abxyuv,\ dddd,\ afxyuv,\ hefxyuv,\ gggg,\ ebxyuv)$$

Porównując algorytmy GREEDY i REFINED GREEDY zauważamy, że dzięki dodatkowym cięciom unikamy propagacji "błędnych" cięć spowodowanych przez zachłanny wybór podsłów. Algorytm GREEDY, nawet w przypadku gdy $k = 4$, można za pomocą specjalnie dobranych danych wejściowych zmusić do wyboru nieoptymalnych cięć i spowodować lawinowy wzrost rozmiaru rozwiązania [10]. Wykażemy, że w przypadku algorytmu REFINED GREEDY, liczba błędnych cięć jest ograniczona i dzięki dodatkowym cięciom takie kaskadowe pogorszenie rozwiązania nie jest możliwe.

**Lemat 1.3.1** *Algorytm* REFINED GREEDY *można zmodyfikować w taki sposób, by rozwiązywał problemy* $k$-RMCSP *oraz* $k$-SMCSP.

**Dowód:** Pierwsza modyfikacja polega na zmianie definicji zbioru $\Phi$ na $\Phi' = \Phi \cup \{-\delta : \delta \in \Phi\}$. Kolejną zmianą jest wybór słowa $S$. Musimy również rozpatrywać słowa $S$ takie, że $S$ jest podsłowem pewnego niezaznaczonego bloku z $\mathcal{A}$, a $-S$ jest podsłowem pewnego niezaznaczonego bloku z $\mathcal{B}$. ■

**Twierdzenie 1.3.2** *Algorytm* REFINED GREEDY *oblicza* $2k^2$*-aproksymacyjne rozwiązanie dla problemu* $k$-MCSP *oraz* $2(2k-1)^2$*-aproksymacyjne rozwiązanie dla problemu* $k$-RMCSP.

**Dowód:** Łatwo zauważyć, że podział obliczony przez algorytm jest wspólnym podziałem. Musimy jedynie udowodnić, że rozmiar wyznaczonego podziału nie jest zbyt odległy od rozmiaru rozwiązania optymalnego.

Dla czytelności opisu skupimy się na udowodnieniu twierdzenia dla problemu $k$-MCSP, a następnie krótko opiszemy niezbędne zmiany potrzebne w dowodzie dla problemu $k$-RMCSP.

Definicję wspólnego podziału dla słów można łatwo uogólnić na wspólny podział dwóch ciągów słów — dzielimy poszczególne słowa, a dla otrzymanych (całościowych) podziałów musi istnieć stosowna bijekcja. Przypomnijmy, że podział słowa też jest ciągiem słów.

**Obserwacja 1.3.3** *Niech* $(\mathcal{Q}, \mathcal{R})$ *będzie wspólnym podziałem podziałów* $\mathcal{A}$*,* $\mathcal{B}$ *i niech* $\delta$ *będzie dowolnym duetem występującym w* $\mathcal{Q}$ *oraz* $\mathcal{R}$*. Niech* $\mathcal{Q}'$ *będzie podziałem* $\mathcal{A}$ *otrzymanym z* $\mathcal{Q}$ *przez przecięcie wszystkich wystąpień duetu* $\delta$ *w słowach z* $\mathcal{Q}$*, a* $\mathcal{R}'$ *podziałem* $\mathcal{B}$ *otrzymanym z* $\mathcal{R}$ *przez przecięcie wszystkich wystąpień duetu* $\delta$ *w słowach z* $\mathcal{R}$*. Wówczas* $(\mathcal{Q}', \mathcal{R}')$ *jest wspólnym podziałem* $\mathcal{A}$ *i* $\mathcal{B}$*.*

**Dowód:** Ponieważ $\mathcal{Q}$ i $\mathcal{R}$ składają się z tych samych bloków (każdy w tej samej liczbie egzemplarzy), to każdy blok $P$ z $\mathcal{Q}$ zawierający $\delta$ występuje również w $\mathcal{R}$ (i na odwrót). Stąd, jeśli dokonamy cięć we wszystkich wystąpieniach $\delta$ w $\mathcal{Q}$ i $\mathcal{R}$, otrzymane podziały będą zawierały te same multizbiory bloków. ■

Niech $\pi = (\mathcal{P}, \mathcal{Q})$ będzie wspólnym podziałem słów $A$ i $B$ o najmniejszym rozmiarze, $m$ jego rozmiarem, zaś $\Delta$ zbiorem wszystkich granicznych duetów bloków w podziałach $\mathcal{P}$ i $\mathcal{Q}$. Niech $T$ będzie liczbą kroków wykonanych przez algorytm REFINED GREEDY dla słów $A$ i $B$. Iteracyjnie skonstruujemy ciąg wspólnych podziałów $\pi_i$ słów $A$ i $B$, $i = 1, \ldots, T$, który pozwoli nam oszacować rozmiar podziału obliczonego przez algorytm REFINED GREEDY.

Niech $\pi_1$ będzie wspólnym podziałem otrzymanym z $\pi$ przez przecięcie **wszystkich** wystąpień duetów z $\Delta$. Fakt, że $\pi_1$ jest wspólnym podziałem wynika z obserwacji 1.3.3. Cięcia (wystąpień) duetów z $\Delta$ nazywamy *początkowymi cięciami*.

Dla instancji problemu $k$-MCSP, liczba bloków w $\pi_1$ jest co najwyżej $k$ razy większa od liczby bloków w $\pi$ – jeśli symbol $a$ występuje jako pierwszy w pewnym bloku z $\pi$, to podział $\pi_1$ zawiera co najwyżej $k$ bloków rozpoczynających się od $a$. Zatem liczba początkowych cięć jest nie większa niż $km - 1$.

Niech $S_i$ będzie podsłowem wybranym przez algorytm REFINED GREEDY w $i$-tej iteracji algorytmu, a $\Phi_i$ zbiorem granicznych duetów wystąpień bloków $S_i^A$ i $S_i^B$ odpowiednio w słowach $A$ i $B$. Dla kroków algorytmu REFINED GREEDY o numerach $i \geq 1$ definiujemy $\pi_{i+1}$ jako podział otrzymany z $\pi_i$ przez przecięcie wszystkich wystąpień duetów z $\Phi_i$. Oznaczmy podziały $\mathcal{A}$ i $\mathcal{B}$ na początku $i$-tej iteracji przez $\mathcal{A}_i$ i $\mathcal{B}_i$, indeks pierwszego symbolu $S_i^A$ w słowie $A$ przez $s_i$, indeks ostatniego symbolu $S_i^A$ w $A$ przez $t_i$. Analogicznie oznaczamy przez $s'_i$ indeks pierwszego symbolu $S_i^B$ w $B$ i przez $t'_i$ indeks ostatniego symbolu $S_i^B$ w $B$.

**Obserwacja 1.3.4** *Dla każdego $i = 1, 2, \ldots, T$ i każdego $l = 0, 1, \ldots, |S_i| - 1$, pozycja $s_i + l$ jest początkowym cięciem w $A$ wtedy i tylko wtedy, gdy pozycja $s'_i + l$ jest początkowym cięciem w $B$.*

**Dowód:** Obserwacja wynika z definicji $\pi_1$ i początkowych cięć: jeśli jedno wystąpienie duetu jest przecięte, to wszystkie jego wystąpienia są też przecięte. ■

Powyższa obserwacja mówi, że jeśli blok $S_i^A$ zawiera jedno lub więcej początkowych cięć, to blok $S_i^B$ zawiera taką samą liczbę początkowych cięć, a ich pozycje względem początków bloków są takie same.

Niech $\mathcal{A}'_i \subseteq \mathcal{A}_i$ i $\mathcal{B}'_i \subseteq \mathcal{B}_i$ będą podpodziałami $\mathcal{A}_i$ i $\mathcal{B}_i$ złożonymi ze wszystkich niezaznaczonych bloków na początku $i$-tej iteracji, a $\pi'_i$ obcięciem $\pi_i$ do $\mathcal{A}'_i$ i $\mathcal{B}'_i$. Obserwacja 1.3.4 implikuje następujący ważny fakt.

**Obserwacja 1.3.5** *Dla każdego $i = 1, \ldots, T$, $\pi'_i$ jest wspólnym podziałem $\mathcal{A}'_i$ i $\mathcal{B}'_i$.*

**Dowód:** Powyższe stwierdzenie można udowodnić przez indukcję. Dla $i = 1$ nic nie jest zaznaczone, więc $\mathcal{A}'_1 = (A)$, $\mathcal{B}'_1 = (B)$, $\pi'_1 = \pi_1$ — stwierdzenie jest oczywiste. Weźmy $i > 1$. Z obserwacji 1.3.3 i 1.3.4 wynika, że bloki z $\pi_i$ pokrywane przez $S_{i-1}^A$ są takie same, jak bloki z $\pi_i$ pokrywane przez $S_{i-1}^B$. Zauważmy, że cięcia na zewnątrz $S_{i-1}^A$ i $S_{i-1}^B$, w wystąpieniach duetów z $\Phi_{i-1}$, są używane do otrzymania $\pi'_i$ z $\pi'_{i-1}$ i $(\mathcal{A}'_i, \mathcal{B}'_i)$ z $(\mathcal{A}'_{i-1}, \mathcal{B}'_{i-1})$, co kończy dowód. ■

**Lemat 1.3.6** *Dla każdego $i = 1, \ldots, T$,*

- *podsłowo $S_i = a_{s_i} \ldots a_{t_i}$ jest blokiem w $\pi_i$, lub*
- *podsłowo $S_i$ zawiera początkowe cięcie.*

**Dowód:** Lemat wynika z Obserwacji 1.3.5 oraz zachłannej natury algorytmu REFINED GREEDY. Dla każdego wspólnego podsłowa $S$ podziałów $\mathcal{A}'_i$ i $\mathcal{B}'_i$, które nie spełnia warunków lematów, istnieje dłuższe wspólne podsłowo $S'$ podziałów $\mathcal{A}'_i$ i $\mathcal{B}'_i$ takie, że $S \sqsubset S'$. ■

Lemat 1.3.6 pozwala oszacować z góry liczbę bloków we wspólnym podziale $\pi_T$ (czyli, po zakończeniu wykonywania algorytmu). Jeśli REFINED GREEDY wybiera jako $S_i$ cały blok w $\pi_i$, to $\pi_{i+1} = \pi_i$ — liczba bloków pozostaje bez zmian. Jeśli REFINED GREEDY wybiera jako $S_i$ podsłowo, które zawiera *początkowe cięcie*, to liczba bloków $\pi_{i+1}$ może wzrosnąć w stosunku do $\pi_i$ o co najwyżej $2k$ — co najwyżej tyle wykonamy nowych przecięć. Cięciami przeprowadzającymi $\pi_i$ na $\pi_{i+1}$ obciążamy jedno *początkowe cięcie*, które jest zawarte w $S_i^A$. Podział $\pi_T$ składa się z co najwyżej $(2k+1)(km-1)+1$ bloków. Zauważmy, że liczba początkowych bloków jest równa co najwyżej $km$.

Z konstrukcji $\pi_T$ wynika, że jest to wspólny podział podziału $(\mathcal{A}_T, \mathcal{B}_T)$ obliczanego przez REFINED GREEDY. Stąd liczba bloków $\pi_T$ stanowi górne ograniczenie na liczbę bloków w $(\mathcal{A}_T, \mathcal{B}_T)$. Zatem współczynnik aproksymacji algorytmu wynosi co najwyżej $k(2k+1)$. Aby nieznacznie poprawić ten współczynnik zauważmy, że jeśli algorytm REFINED GREEDY w $L$ krokach wybierał podsłowa, które zawierają początkowe cięcia (zauważmy, że $L \leq km-1$), to istnieje co najwyżej $2kL + (km-1-L) \leq 2k^2m - 1$ cięć wyznaczonych w $A$ (i $B$) przez podział $(\mathcal{A}_T, \mathcal{B}_T)$, co daje żądany współczynnik aproksymacji. ■

Dla problemu $k$-SMCSP i $k$-RMCSP musimy jedynie uwzględnić fakt, że $S$ ze słowa $A$ może zostać dopasowane do $R$ ze słowa $B$ nawet gdy $S \neq R$, ale $S = -R$.

Zatem w obserwacji 1.3.3 należy brać pod uwagę cięcia nie tylko używające duetu $\delta$, ale również duetu $-\delta$. Analogicznie postępujemy w przypadku konstrukcji $\pi_1$ z $\pi$, dla każdego $\delta \in \Delta$ przecinamy wszystkie wystąpienia $\delta$ i $-\delta$; dla przypadku $k$-SMCSP liczba cięć w $\pi_1$ wzrasta, tak jak w przypadku $k$-MCSP, co najwyżej $k$ razy, natomiast dla $k$-RMCSP wzrasta co najwyżej $2k-1$ razy.

W obserwacji 1.3.4 musimy uwzględnić przypadki $S_i^A = S_i^B$ lub $S_i^A = -S_i^B$. W tym drugim przypadku liczymy względne pozycje początkowych cięć w $S_i^B$ od tyłu (to jest, twierdzimy, że $i+l$ jest początkowym cięciem w $A$ wtedy i tylko wtedy, gdy $i' - l - 1$ jest początkowym cięciem w $B$).

Dla problemu $k$-SMCSP, liczba bloków w $\mathcal{A}$ w pojedynczej iteracji wzrasta co najwyżej o $2k$, natomiast dla $k$-RMCSP co najwyżej o $2(2k-1)$.

Ze względu na zależności pomiędzy problemami SMCSP i problemem SSBR, oraz pomiędzy RMCSP i problemem SBR (bez znaku) otrzymujemy następujące twierdzenie.

**Twierdzenie 1.3.7** *Dla problemów $k$-SSBR i $k$-SBR istnieją algorytmy o złożoności wielomianowej i współczynnikach aproksymacji równych odpowiednio $4k^2$ i $8(2k-1)^2$.*

**Dowód:** Twierdzenie to jest konsekwencją twierdzenia 1.3.2 oraz lematów 1.2.2 i 1.2.4. ■

Zauważmy, że algorytm REFINED GREEDY można w prosty sposób zaimplementować tak, by jego złożoność czasowa wynosiła $\mathcal{O}(n^3)$.

### 1.3.3 Algorytm EDUCATED GREEDY

W przeprowadzonej analizie algorytmu nie korzystaliśmy z faktu iż $S_i$ jest *najdłuższym* wspólnym podsłowem. Istotny był jedynie fakt, że $S_i$ nigdy nie jest właściwym podsłowem niezaznaczonego bloku podziału $\pi_i$ (dowód lematu 1.3.6).

Korzystając z tej obserwacji przedstawimy dwie implementacje szybszego algorytmu, który nazwaliśmy EDUCATED GREEDY. Tak jak w przypadku algorytmu REFINED GREEDY, szczegółowo opisujemy algorytm dla problemu $k$-MCSP (bez znaku).

Dla problemów $k$-SMCSP i $k$-RMCSP niezbędne są modyfikacje analogiczne jak w poprzednim algorytmie.

Rozpoczniemy od najprostszej implementacji algorytmu EDUCATED GREEDY o złożoności czasowej $\mathcal{O}(k^2n)$. W następnej części pracy opiszemy liniową implementację algorytmu EDUCATED GREEDY. Kluczowymi składnikami implementacji o liniowej złożoności czasowej jest liniowy algorytm konstrukcji drzew sufiksowych [18] oraz struktura danych umożliwiająca reprezentację zbiorów rozłączonych [19].

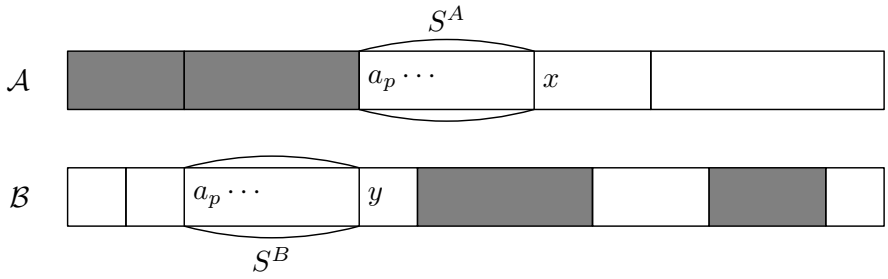
Obydwie implementacje algorytmu EDUCATED GREEDY mają bardzo zbliżoną strukturę i różnią się jedynie sposobem wyboru w każdej iteracji wspólnego podsłowa $S$ (realizowanego za pomocą funkcji FIND–UNEXTENDABLE). Opisywane implementacje algorytmu EDUCATED GREEDY, wykorzystują następujący schemat:

**Algorytm 1.3.3**: EDUCATED GREEDY′

**Dane**: dwa zgodne słowa $A = a_1 \dots a_n$ i $B = b_1 \dots b_n$
**Wynik**: zgodne podziały $(\mathcal{A}, \mathcal{B})$ słów $A$ i $B$

```
1  A ← (A), B ← (B)
2  p = 1
3  while p ≤ n do
4      S ← FIND-UNEXTENDABLE(a_p)
5      niech S^A będzie dowolnym wystąpieniem S w A, a S^B dowolnym
       wystąpieniem S w B
6      zaznacz S^A w A oraz S^B w B
7      niech Φ będzie zbiorem granicznych duetów S^A w A oraz granicznych
       duetów S^B w B
8      przetnij wszystkie wystąpienia duetów z Φ w niezaznaczonych blokach
       A i B
9      p ← min{j : j ≥ p i a_j nie należy do zaznaczonego bloku}
10 end
11 return (A, B)
```

Dla zadanego $c \in \Sigma$, funkcja FIND–UNEXTENDABLE$(c)$ oblicza wspólne podsłowo $S$ podziałów $\mathcal{A}$ i $\mathcal{B}$ i takie, że $c$ należy do $S$ oraz nie istnieje właściwe nadsłowo $S$, które jest wspólnym, niezaznaczonym podsłowem tych podziałów.




Rysunek 1.3: Przykład działania funkcji FIND–UNEXTENDABLE.

## Prosta implementacja algorytmu EDUCATED GREEDY

Najprostsza implementacja funkcji FIND–UNEXTENDABLE$(c)$ polega na sprawdzeniu wszystkich $k^2$ potencjalnych pozycji podsłów $S^A$ i $S^B$. Takie postępowanie wymaga $\mathcal{O}(k^2|S|)$ operacji. Ponieważ wszystkie pozostałe operacje wykonywane przez algorytm wymagają $\mathcal{O}(n)$ operacji, stąd całkowity czas wykonywania algorytmu wynosi $\sum_i O(k^2|S_i|) = O(k^2 n)$. Formalny zapis takiej implementacji funkcji FIND–UNEXTENDABLE został przedstawiony na rysunku 1.4.

**Function** SIMPLE-FIND-UNEXTENDABLE ($c$)

```
1  S ← ϵ
2  foreach i : A[i] = c oraz znak A[i] nie jest zaznaczony do
3      foreach j : B[j] = c oraz znak B[j] nie jest zaznaczony do
4          niech l, r ≥ 0 największe liczby całkowite, takie, że:
5          - A[i − l..i + r] = B[j − l..i + r],
6          - słowa A[i − l..i + r], B[j − l..i + r] nie zawierają cięć
7            oraz zaznaczonych znaków
8          if l + r + 1 > |S| then S = A[i − l..i + r]
9      end
10 end
11 return S
```

Rysunek 1.4: Prosta implementacja funkcji FIND-UNEXTENDABLE.

Tak jak już wcześniej wspomnieliśmy, dowód lematu 1.3.6 jest jedynym miejscem w dowodzie twierdzenia 1.3.2, które odnosi się do wyboru wspólnego podsłowa $S_i$ w algorytmie REFINED GREEDY. W dowodzie korzystamy tylko z tego, że $S_i$ nie może być rozszerzone (w którąkolwiek ze stron). Zatem lemat 1.3.6 jest również prawdziwy dla algorytmu EDUCATED GREEDY, a współczynnik aproksymacji $\mathcal{O}(k^2)$ wynika z tego samego rozumowania, co w przypadku algorytmu REFINED GREEDY. W wyniku otrzymujemy następujące twierdzenie.

**Twierdzenie 1.3.8** *Prosta implementacja algorytmu* EDUCATED GREEDY *działa w czasie $\mathcal{O}(k^2 n)$ i oblicza $\mathcal{O}(k^2)$-aproksymacyjne rozwiązanie dla problemów $k$-*MCSP*, $k$-*RMCSP *i $k$-*SBR.

## Efektywna implementacja algorytmu EDUCATED GREEDY

Najbardziej czasochłonną częścią prostej implementacji jest wyszukiwanie nierozszerzalnego wspólnego (niezaznaczonego) podsłowa podziałów $\mathcal{A}$, $\mathcal{B}$ zawierającego zadany symbol. W pesymistycznym przypadku wyznaczenie podsłowa $S$ wymaga $\mathcal{O}(k^2|S|)$ operacji. Pokażemy, w jaki sposób przy użyciu dodatkowych struktur danych zaimplementować ten krok w (zamortyzowanym) czasie $\mathcal{O}(|S|)$, co pozwoli na opracowanie liniowego algorytmu $\mathcal{O}(k^2)$-aproksymacyjnego dla problemu $k$-MCSP.

Niech $X$ będzie złączeniem słowa $A$, znaku \$ oraz słowa $B$ (gdzie \$ jest symbolem, który nie występuje ani w $A$, ani w $B$), czyli $X = A\$B$. Zakładamy, że symbole występujące w słowie $X$ są reprezentowane przez dodatnie liczby całkowite zapisane na co najwyżej $\mathcal{O}(\log n)$ bitach.

Idea nowego algorytmu jest bardzo prosta. Zauważmy, że mając dane słowa $A$ i $B$, wspólne podsłowo $Z$ słów $A$ i $B$, oraz drzewo sufiksowe dla $X = A\$B$, możemy łatwo obliczyć podsłowo $Z'$, takie, że:

- $Z$ jest prefiksem $Z'$,
- $Z'$ nie jest właściwym prefiksem żadnego innego wspólnego podsłowa $A$ i $B$ (czyli nie jest możliwe przedłużenie $Z'$ "w prawo").

W celu obliczenia $Z'$ algorytm rozpoczyna poszukiwania w korzeniu drzewa sufiksowego. Następnie porusza się krawędziami drzewa zgodnie z kolejnymi symbolami słowa $Z$. W chwili gdy wszystkie symbole $Z$ zostaną wykorzystane, algorytm ma (prawie) pełną dowolność wyboru kolejnych krawędzi (czyli kolejnych symboli słowa $Z'$). Jedyne ograniczenie to konieczność zapewnienia, że słowo $Z'$ jest wspólnym podsłowem $A$ i $B$. W tym celu algorytm wybiera te krawędzie, które prowadzą do poddrzew posiadających co najmniej jeden liść odpowiadający sufiksowi $X$ rozpoczynającemu się w $A$ oraz jeden liść odpowiadający sufiksowi z $B$. Algorytm kontynuuje takie postępowanie aż dojdzie do węzła, w którego poddrzewach są wyłącznie liście odpowiadające sufiksom o początkach w $A$, albo wyłącznie odpowiadające sufiksom o początkach w $B$.

W przypadku funkcji FIND-UNEXTENDABLE początkowym słowem $Z$ od którego rozpoczynane są poszukiwania jest symbol $c$.

Nie kończy to jednak obliczeń, gdyż może się zdarzyć, że słowo $Z'$ można rozszerzyć "w lewo". Stąd, musimy również wyznaczyć wspólne podsłowo $S'$ słów $-A$ i $-B$, którego prefiksem jest $-Z'$ oraz $S'$ nie jest właściwym prefiksem innych wspólnych podsłów $-A$ i $-B$. Jedyna różnica w stosunku do poprzedniego kroku, to fakt, że tym razem do poszukiwań konieczne jest drzewo sufiksowe słowa $(-A)\$(-B)$.

Podsłowo $-S'$ jest wspólnym podsłowem, którego poszukiwaliśmy. Ilustracja tego podstępowania została przedstawiona na rysunku 1.5.

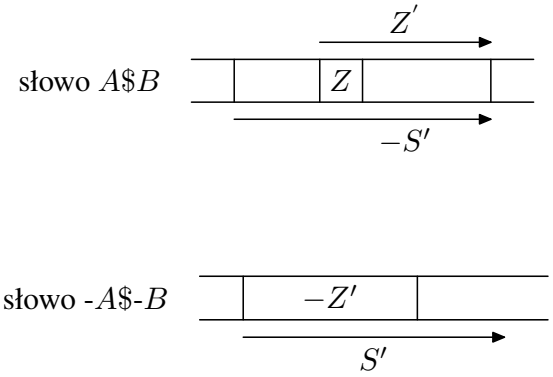
Dodatkową trudność implementacyjną dla algorytmu EDUCATED GREEDY stanowi fakt, że mamy do czynienia z przypadkiem dynamicznym, kiedy to w kolejnych iteracjach niektóre fragmenty słów zostają zamarkowane jako już zużyte, a pewne duety w $A$ i $B$ zostają przecięte. W dalszej części rozdziału pokazujemy, w jaki sposób poradzić sobie z tymi problemami.

Opiszemy funkcję EXTEND$(\mathcal{A}, \mathcal{B}, Z)$, która dla zadanych podziałów $\mathcal{A}$ i $\mathcal{B}$ oraz słowa $Z$, znajduje niezaznaczone wspólne podsłowo $Z'$ i takie, że $Z$ jest prefiksem $Z'$, oraz $Z'$ nie jest właściwym prefiksem żadnego innego nieznaczonego wspólnego podsłowa podziałów $\mathcal{A}$ i $\mathcal{B}$.

Szybka implementacja funkcji EXTEND jest kluczowym elementem funkcji FIND-UNEXTENDABLE:

**Function** FIND-UNEXTENDABLE ($Z$)

```
1 Z' ← EXTEND(A, B, Z)
2 S' ← EXTEND(−A, −B, −Z')
3 return (−S')
```




Rysunek 1.5: Konstrukcja nierozszerzalnego wspólnego podsłowa $S$ z pomocą funkcji FIND-UNEXTENDABLE.

Dla słowa $Z$, niech $Z[i \rightarrow]$ będzie sufiksem $Z$ rozpoczynającym się na pozycji $i$. Ranga sufiksu $T$ słowa $Z$, oznaczana przed $\text{RANK}_Z(T)$, to jego numer porządkowy wśród wszystkich sufiksów $Z$ uporządkowanych leksykograficznie. Rangi wszystkich sufiksów $Z$ mogą zostać obliczone w czasie liniowym podczas konstrukcji drzewa sufiksowego dla $Z$ [18]. Zauważmy, że dla dowolnego poddrzewa drzewa sufiksowego rangi jego liści (rangi odpowiadających im sufiksów) tworzą spójny przedział liczb naturalnych.
Podczas działania algorytmu EDUCATED GREEDY będziemy utrzymywać drzewo sufiksowe $\mathcal{T}$. Na początku działania algorytmu $\mathcal{T}$ jest drzewem sufiksowym słowa $X = A\$B$. Podczas kolejnych iteracji $\mathcal{T}$ będzie poddrzewem oryginalnego drzewa. Dodatkowo, dla każdego węzła $v$ pamiętamy końce przedziału $[i_v, j_v]$ zawierającego rangi sufiksów $X$ odpowiadających liściom poddrzewa $\mathcal{T}$ o korzeniu w $v$.

W poniższym opisie funkcji EXTEND przyjmujemy, że początkowo $\mathcal{T}$ jest drzewem sufiksowym $X$. Jeśli odwołujemy się do (etykiety) krawędzi lub węzła, to zawsze jest to (etykieta) krawędź lub węzeł z $\mathcal{T}$. Funkcja EXTEND rozpoczyna obliczenia z zadanym wspólnym podsłowem $Z$ podziałów $\mathcal{A}$ oraz $\mathcal{B}$ i w kolejnych iteracjach stara się rozszerzyć to słowo o kolejne znaki. W tym celu przeszukiwana jest ścieżka w drzewie sufiksowym od korzenia drzewa etykietowana słowem $Z$. Gdy słowo $Z$ kończy się wewnątrz krawędzi drzewa, wystarczy jedy-

nie sprawdzić czy możliwe jest rozszerzenie słowa o kolejny znak. Gdy słowo $Z$ kończy się w węźle wewnętrznym $v$, algorytm stara się tak dobrać kolejny znak, aby nie napotkać wcześniej wykonanego cięcia, oraz żeby tak otrzymane słowo było zawarte jednocześnie w $\mathcal{A}$ i $\mathcal{B}$.

**Function** EXTEND ($\mathcal{A}, \mathcal{B}, Z$)

```
1  (u, v) ← krawędź, dla której L_u ⊏ Z i Z ⊑ L_v
2  p ← ostatni znak słowa Z
3  if Z = L_v then
4      foreach w ∈ children(v) do
5          y_1 ... y_r ← etykieta krawędzi (v, w)
6          if p y_1 nie jest przeciętym duetem and EXISTS(A, w) and
           EXISTS(B, w) then
7              return EXTEND(A, B, Z y_1)
8          else
9              usuń krawędź (v, w) z T wraz z całym poddrzewem
10         end
11     end
12     return Z
13 else
14     y_1 ... y_r ← etykieta krawędzi (u, v)
15     i ← indeks taki, że L_u y_1 ... y_i = Z
16     if y_i y_{i+1} jest przeciętym duetem then
17         return Z
18     else
19         return EXTEND(A, B, Z y_{i+1})
20     end
21 end
```

Do pełnego opisu funkcji EXTEND brakuje opisu funkcji EXISTS$(Y, v)$, która odpowiada na pytanie, czy w poddrzewie o korzeniu w $v$ istnieje liść reprezentujący sufiks słowa $X$ rozpoczynający się w części $Y$ słowa $X$ ($Y = A$ lub $Y = B$) i taki, że jego pierwszy znak nie jest zaznaczony. W implementacji funkcji EXISTS używamy rozwiązania specjalnego przypadku problemu złączania zbiorów rozłącznych [19], w którym zbiory są rozłącznymi przedziałami liczb całkowitych. Problem złączania zbiorów rozłącznych polega na wykonywaniu ciągu następujących dwóch operacji, które tutaj odnoszą się do zbioru rozłącznych przedziałów:

- FIND$(h)$ - podaje największy element należący do zbioru zawierającego $h$;
- UNION$(h)$ - tworzy nowy przedział stanowiący złączenie zbioru zawierającego $h$ i kolejnego przedziału (przedziału zawierającego FIND$(h) + 1$)

oraz niszczy dwa stare przedziały; jeśli następny przedział nie istnieje, to operacja nie powoduje żadnych akcji.

Gabow i Tarjan [19] opisali implementację, która wykonuje ciąg $m$ operacji UNION i FIND w czasie $\mathcal{O}(m)$ — amortyzowany czas pojedynczej operacji wynosi $\mathcal{O}(1)$.

W naszym przypadku ustalamy $m = 2n + 1$ i będziemy pracować z dwoma zbiorami przedziałów: $\mathcal{S}_A$ dla słowa $A$, oraz $\mathcal{S}_B$ dla słowa $B$. Dla $\mathcal{S}_A$ (odpowiednio $\mathcal{S}_B$) przedział $\{l, l+1, \ldots, r\}$ reprezentuje sytuację, w której pierwszy symbol sufiksu o randze $r$ nie jest zaznaczony oraz nie istnieje sufiks słowa $X$ rozpoczynający się w części odpowiadającej $A$ (odpowiednio $B$) o randze z przedziału $\{l, \ldots, r-1\}$. Inicjalizacja $\mathcal{S}_A$ i $\mathcal{S}_B$ wymaga następujących kroków: niech $\chi_A$ i $\chi_B$ będą dwiema tablicami binarnymi o długościach równych $m$ i takimi, że $\chi_A[i] = 1$ wtedy i tylko wtedy, gdy sufiks $X$ o randze $i$ rozpoczyna się w części odpowiadającej $A$, natomiast $\chi_B[i] = 1$ wtedy i tylko wtedy, gdy sufiks $X$ o randze $i$ rozpoczyna się w części odpowiadającej $B$. Bierzemy $\mathcal{S}_A = \{\{l, l+1, \ldots, r\} \mid \chi_A[r] = 1, \chi_A[i] = 0, \text{ dla } i = l, \ldots, r-1, \text{ oraz albo } l = 1, \text{ albo } \chi_A[l-1] = 1\}$. Zbiór $\mathcal{S}_B$ jest zdefiniowany w analogiczny sposób. Powyższa inicjalizacja może zostać wykonane w czasie liniowym, jako produkt uboczny konstrukcji drzewa sufiksowego $X$. Mając dane zbiory $\mathcal{S}_A$ i $\mathcal{S}_B$ możemy zaimplementować funkcję EXISTS w następujący sposób ($Y = A, B$):

**Function** EXISTS $(Y, u)$

```
1 (i_u, j_u) ← rangi, minimalna i maksymalna, sufiksów zawartych w
  poddrzewie węzła u
2 if S_Y.FIND(i_u) > j_u then
3     return False
4 else
5     return True
6 end
```

Aby utrzymywać aktualny stan zbiorów $\mathcal{S}_A$ i $\mathcal{S}_B$ wykonujemy operację

$$\text{UNION}(\text{RANK}(X[i \rightarrow]))$$

gdy tylko zaznaczamy symbol $a_i$ w $A$. Analogicznie wykonujemy operację

$$\text{UNION}(\text{RANK}(X[n+1+i \rightarrow]))$$

gdy tylko zaznaczamy symbol $b_i$ w $B$. Daje to już pełen opis struktur danych używanych przez funkcję EXTEND$(\mathcal{A}, \mathcal{B}, Z)$.

Żeby móc wykonać funkcję EXTEND$(-\mathcal{A}, -\mathcal{B}, -Z')$ musimy dodatkowo utrzymywać analogiczne struktury dla słów $-A, -B$ i $-X$.

**Twierdzenie 1.3.9** *Szybka implementacja* EDUCATED GREEDY *wymaga czasu działania $\mathcal{O}(n)$ i oblicza $\mathcal{O}(k^2)$-aproksymacyjne rozwiązanie dla problemów* $k$-MCSP*,* $k$-RMCSP *i* $k$-SBR.

**Dowód:** Z pomocą struktur danych dla problemów zbiorów rozłącznych, każda iteracja wymaga czasu $\mathcal{O}(|S_i|)$ powiększonego o czas poświęcony na usuwanie krawędzi z $\mathcal{T}$ (linia 9). Ponieważ każda krawędź może zostać usunięta jedynie raz, to całkowity czas poświęcony na usuwanie krawędzi z oryginalnego drzewa sufiksowego dla słowa $X$ wynosi $\mathcal{O}(n)$. Zatem całkowity czas działania algorytmu wynosi $\mathcal{O}(n)$. Poprawność implementacji wynika z poprawności algorytmu REFINED GREEDY. ■

## 1.4 Algorytm $\mathcal{O}(k)$–aproksymacyjny

Zanim przejdziemy do opisu algorytmu $\mathcal{O}(k)$–aproksymacyjnego zdefiniujmy problem *minimalnego zbioru przecinającego* (ang. *Minimum Hitting Set*, w skrócie MHS), który wykorzystamy jako narzędzie pomagające rozwiązać problem MCSP.

**Problem 1.8 (Minimalnego Zbioru Przecinającego** MHS**)**

**Dane:** *Skończone uniwersum elementów $U$ oraz rodzina $\mathcal{S} = \{S_1, \ldots, S_k\}$ niepustych podzbiorów zbioru $U$.*

**Wynik:** *Najmniej liczny zbiór $H \subseteq U$ taki, że $H \cap S_i \neq \emptyset$, dla każdego $i = 1..k$.*

Problem MHS jest równoważny problemowi *minimalnego pokrycia zbiorami* (ang. *Minimum Set Cover*) [1].

### 1.4.1 Zastosowanie problemu MHS do rozwiązania problemu MCSP

W tym rozdziale opiszemy, w jaki sposób dowolny algorytm dla problemu MHS można wykorzystać do rozwiązania problemu MCSP. Następnie pokażemy, że specjalne właściwości instancji problemu MHS napotykane w trakcie rozwiązywania problemu MCSP pozwalają otrzymać algorytm $\mathcal{O}(k)$–aproksymacyjny.

Niech $A$, $B$ będą wejściowymi słowami w problemie MCSP i niech $P$ będzie słowem, które występuje więcej (odpowiednio mniej) razy w $A$ niż w $B$. Wówczas dowolny wspólny podział $A$ i $B$ powstaje w wyniku przecięcia co najmniej jednego wystąpienia słowa $P$ w $A$ (odpowiednio w $B$). Idea nowego algorytm polega na próbach "trafienia" (tzn. przecinaniu) wszystkich podsłów $A$ i $B$, które mają różne liczby wystąpień w słowach $A$ i $B$.

Poniższy algorytm wykorzystuje rozwiązanie problemu MHS do znalezienia rozwiązania problemu MCSP.

**Algorytm 1.4.1**: HS

**Dane**: zgodne słowa $A$ i $B$
**Wynik**: zgodne podziały $(\mathcal{A}, \mathcal{B})$ słów $A$ i $B$

1 zbuduj instancję $(U, \mathcal{S})$ problemu MHS:
  $U \leftarrow duos(A) \cup duos(B)$
  $T \leftarrow \{X \in \Sigma^+ \mid \#_X(A) \neq \#_X(B)\}$
  $\mathcal{S} \leftarrow \{duos(X) \mid X \in T\}$
2 rozwiąż (aproksymacyjnie) problem Minimum Hitting Set
  $\Phi \leftarrow$ rozwiązanie dla $(U, \mathcal{S})$
3 użyj zbioru duetów $\Phi$ aby otrzymać wspólny podział:
  $(\mathcal{A}, \mathcal{B}) \leftarrow (\textsc{Cut}(A, \Phi), \textsc{Cut}(B, \Phi))$
4 **return** $(\mathcal{A}, \mathcal{B})$

Udowodnimy teraz, że $(\mathcal{A}, \mathcal{B})$ jest wspólnym podziałem słów $A$ i $B$.

W dowodzie poprawności algorytmu HS wykorzystamy następujące oznaczenie. Dla podziału (słowa) $\mathcal{P} = (P_1, P_2, \ldots, P_m)$ i dowolnego słowa $K$, $\#\textsc{bl}(\mathcal{P}, K)$ jest liczbą bloków w podziale $\mathcal{P}$ równych $K$.

**Lemat 1.4.1** *Podział $(\mathcal{A}, \mathcal{B})$ obliczany przez algorytm* HS *jest wspólnym podziałem słów $A$ i $B$.*

**Dowód:** Załóżmy przeciwnie, że istnieje blok $X \in \mathcal{A}$ taki, że $\#\textsc{bl}(\mathcal{A}, X) \neq \#\textsc{bl}(\mathcal{B}, X)$. Jeśli jest wiele takich bloków, to wybieramy (dowolny) najdłuższy z nich. Ponieważ blok $X$ nie został przecięty przez żaden duet ze zbioru $\Phi$, stąd mamy $duos(X) \cap \Phi = \emptyset$. Jednakże $\Phi$ jest poprawnym rozwiązaniem problemu MHS, co pociąga za sobą, że $duos(X) \notin \mathcal{S}$. Zatem $\#_X(A) = \#_X(B)$. Aby otrzymać sprzeczność musimy wykazać, że $\#\textsc{bl}(\mathcal{A}, X) = \#\textsc{bl}(\mathcal{B}, X)$.

Ponieważ żadne z wystąpień słowa $X$ w słowach $A$ i $B$ nie zostało przecięte, to

$$\#\textsc{bl}(\mathcal{A}, X) = \#_X(A) - \sum_{Y \sqsubseteq A, X \sqsubset Y} \#_X(Y) \cdot \#\textsc{bl}(\mathcal{A}, Y)$$

$$\#\textsc{bl}(\mathcal{B}, X) = \#_X(B) - \sum_{Y \sqsubseteq B, X \sqsubset Y} \#_X(Y) \cdot \#\textsc{bl}(\mathcal{B}, Y)$$

Ponieważ $X$ jest najdłuższym blokiem dla którego $\#\textsc{bl}(\mathcal{A}, X) \neq \#\textsc{bl}(\mathcal{B}, X)$, to dla wszystkich $Y$ spełniających $X \sqsubset Y$ (czyli $X$ jest podsłowem $Y$) mamy $\#\textsc{bl}(\mathcal{A}, Y) = \#\textsc{bl}(\mathcal{B}, Y)$. W takim razie otrzymujemy sprzeczność: $\#\textsc{bl}(\mathcal{A}, X) = \#\textsc{bl}(\mathcal{B}, X)$. ■

**Lemat 1.4.2** *Algorytm* HS *znajduje $2k$-aproksymację minimalnego wspólnego podziału przy założeniu, że problem MHS jest rozwiązywany dokładnie.*

**Dowód:** Rozważmy dowolny wspólny podział $(\mathcal{A}', \mathcal{B}')$ słów $A$ i $B$ i niech $\Phi'$ będzie zbiorem duetów, których przecięcia (a dokładniej ich wystąpienia) dały podział $(\mathcal{A}', \mathcal{B}')$. Ponieważ każdy blok podziału $\mathcal{A}'$ występuje tyle samo razy w $\mathcal{A}'$, co w $\mathcal{B}'$ (i na odwrót), to $\Phi$ jest podzbiorem $\Phi'$. Ponieważ w algorytmie HS przecinane są wszystkie wystąpienia duetów z $\Phi$, to

$$|\mathcal{A}| \leq k \cdot (|\mathcal{A}'| + |\mathcal{B}'| - 2) + 1 \leq k \cdot (|\mathcal{A}'| + |\mathcal{B}'|).$$

■

Zauważmy, że jeśli zastąpimy znajdowanie optymalnego rozwiązania problemu MHS przez $\alpha$-aproksymację, to algorytm HS znajdzie rozwiązanie $2k\alpha$-aproksymacyjne dla problemu minimalnego wspólnego podziału.

Niestety problem Minimum Hitting Set jest trudny do aproksymacji. W pracy [39] autorzy wykazali, że o ile P≠NP, to rozwiązania problemu MHS nie da się aproksymować ze współczynnikiem $c \log n$ (dla pewnej stałej $c > 0$). Nadzieje na lepsze rozwiązanie dają specjalne własności instancji $(U, \mathcal{S})$.

Niech $(\mathcal{A}_o, \mathcal{B}_o)$ będzie minimalnym wspólnym podziałem słów $A$ i $B$. Jeśli jest wiele takich podziałów, to wybieramy dowolny z nich. Cięcia występujące w $\mathcal{A}_o$ i $\mathcal{B}_o$ nazywamy *optymalnymi cięciami*. Wszystkich optymalnych cięć jest $2|\mathcal{A}_o| - 2$. Mówimy, że słowo $X = a_i \dots a_j$ (lub odpowiednio $X = b_i \dots b_j$) *przechodzi przez* optymalne cięcie, jeśli istnieje cięcie $l$ w $\mathcal{A}_o$ (lub odpowiednio w $\mathcal{B}_o$) takie, że $i \leq l < j$.

Niech $T$ będzie zbiorem wszystkich podsłów, które należy "przeciąć". Bardziej formalnie, $T = \{X \in \Sigma^* \mid \#_X(A) \neq \#_X(B)\}$. Zauważmy, że w instancji problemu Minimum Hitting Set większość słów z $T$ jest zbędna. Dokładniej, jeśli $X, Y \in T$ i $X$ jest podsłowem $Y$, to możemy bezpiecznie usunąć $Y$ ze zbioru $T$, a rozwiązanie dla $\mathcal{S}' = \{duos(Q) \mid Q \in T \setminus \{Y\}\}$ będzie również dobrym rozwiązaniem dla $\mathcal{S}$. Dzięki tej obserwacji możemy znacznie zredukować rozmiar rodziny $\mathcal{S}$. Relacja $\sqsubseteq$ generuje częściowy porządek na zbiorze $T$. Niech $T_{min} \subseteq T$ będzie zbiorem minimalnych elementów w $T$ względem relacji $\sqsubseteq$.
Zbiór $T_{\min}$ ma następujące własności:

**(P)** Jeśli $X, Y \in T$ i $X$ jest właściwym podsłowem $Y$, to $Y \notin T_{\min}$.

**(Q)** Rozwiązanie problemu MHS dla rodziny $\mathcal{S}' = \{duos(X) \mid X \in T_{\min}\}$ jest również rozwiązaniem dla rodziny $\mathcal{S}$.

**Lemat 1.4.3** *Jeśli $X \in T_{\min}$, to istnieje wystąpienie $X$ w $A$ lub w $B$, które przechodzi nad optymalnym cięciem.*

**Dowód:** Załóżmy, że żadne wystąpienie $X$ w $A$ i żadne wystąpienie $X$ w $B$ nie przechodzi nad optymalnym cięciem.

W takim wypadku każde wystąpienie $X$ jest podsłowem pewnego bloku w optymalnym wspólnym podziale $(\mathcal{A}_o, \mathcal{B}_o)$.

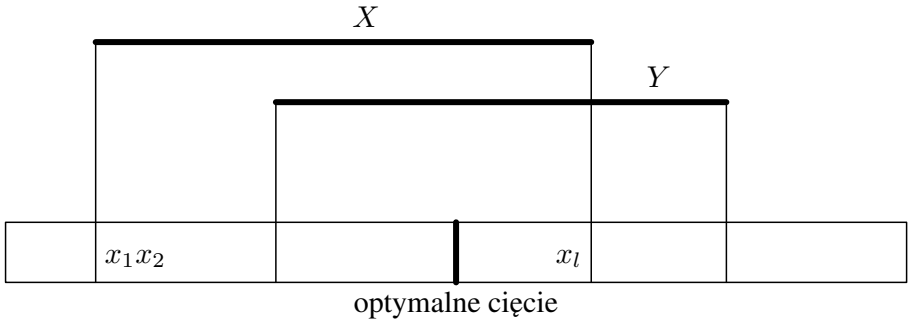
Jednak podziały $(\mathcal{A}_o, \mathcal{B}_o)$ są poprawnym rozwiązaniem problemu MCSP, stąd każdy blok tych podziałów, który zawiera $X$ jako podsłowo, występuje w tej samej liczbie w $\mathcal{A}_o$, co w $\mathcal{B}_o$. Stąd otrzymujemy $\#_X(A) = \#_X(B)$. Zatem $X \notin T$ — sprzeczność. ■

Niech $f$ będzie funkcją, która każdemu słowu z $X \in T_{min}$ przypisuje dowolne optymalne cięcie, przez które przechodzi $X$.

**Przykład:**
Dla słów $A = abaab$, $B = ababa$, minimalny wspólny podział to $(aba, ab), (ab, aba)$, $ba \in T_{min}$ i za $f(ba)$ można wybrać cięcie słowa $B$ pomiędzy jego 2 i 3 znakiem.

**Lemat 1.4.4** *Jeśli* $X, Y \in T_{\min}$, $X = x_1, \ldots, x_l$ *i* $f(X) = f(Y)$, *to* $duos(Y) \cap \{x_1x_2, x_{l-1}x_l\} \neq \emptyset$.

**Dowód:** Ponieważ $X$ i $Y$ przechodzą nad tym samym optymalnym cięciem, to ich wspólny fragment ma rozmiar co najmniej 2. Z własności (P) wiemy, że $X$ nie jest podsłowem $Y$ (i na odwrót). Stąd musi zachodzić $duos(Y) \cap x_1x_2 \neq \emptyset$ lub $duos(Y) \cap x_{l-1}x_l \neq \emptyset$ (zobacz rysunek 1.6). ■




Rysunek 1.6: Ilustracja do lematu 1.4.4.

Zauważmy teraz, że dla podziałów $\mathcal{A}$ słowa $A$ i $\mathcal{B}$ słowa $B$ oraz słowa $X \in T_{\min}$ (niech $X = x_1 \ldots x_l$), jeśli przetniemy wszystkie wystąpienia duetów $x_1x_2$ i $x_{l-1}x_l$ w $\mathcal{A}$ i $\mathcal{B}$, to przetniemy również wszystkie słowa z $T_{\min}$, które przechodzą

przez optymalne cięcie $f(X)$. Pozwala to sformułować następujący algorytm.

**Algorytm 1.4.2**: FASTHS

**Dane**: zgodne słowa $A$ i $B$
oblicz zbiór $T'$ (małego rozmiaru) i taki, że $T_{\min} \subseteq T'$
$\Phi \leftarrow \emptyset$
**for** $X \in T'$ *o niemalejących długościach* **do**
    **if** $duos(X) \cap \Phi = \emptyset$ **then**
        dodaj pierwszy i ostatni duet słowa $X$ do $\Phi$
    **end**
**end**
$\mathcal{A} \leftarrow \text{CUT}(A, \Phi)$, $\mathcal{B} \leftarrow \text{CUT}(B, \Phi)$
**return** $(\mathcal{A}, \mathcal{B})$

**Lemat 1.4.5** *Jeśli podczas wykonywania algorytmu* FASTHS *słowo $X$ pozytywnie przejdzie test $duos(X) \cap \Phi = \emptyset$, to $X \in T_{\min}$.*

**Dowód:** Załóżmy, że słowo $X$ przeszło test, a mimo to $X \notin T_{min}$. Niech $\Phi'$ będzie zbiorem $\Phi$ tuż przed przetworzeniem słowa $X$. Z założenia $X \notin T_{\min}$ wynika, że istnieje słowo $X' \in T_{\min}$ i takie, że $X'$ jest właściwym podsłowem $X$. Ponieważ $|X'| < |X|$, to $X'$ był rozważany przed $X$, a stąd $duos(X') \cap \Phi' \neq \emptyset$. Dodatkowo mamy $duos(X') \subseteq duos(X)$, więc $duos(X) \cap \Phi' \neq \emptyset$, czyli $X$ nie mogło pozytywnie przejść testu — sprzeczność. ■

**Twierdzenie 1.4.6** *Algorytm* FASTHS *oblicza $4k$-aproksymację problemu minimalnego wspólnego podziału słów $A$ i $B$.*

**Dowód:** Niech $X_1$, $X_2$ będą różnymi słowami, dla których zbiór $\Phi$ zostaje powiększony. Z lematu 1.4.4 mamy, że $f(X_1) \neq f(X_2)$. Zatem zbiór $\Phi$ mógł być powiększany co najwyżej $|\mathcal{A}_o| + |\mathcal{B}_o| - 2$ razy, a co za tym idzie, jego rozmiar po zakończeniu wykonywania algorytmu wynosi co najwyżej $2 \cdot (|\mathcal{A}_o| + |\mathcal{B}_o| - 2)$.

Ponieważ rozwiązujemy problem $k$-MCSP, to każdy duet z $\Phi$ wprowadza co najwyżej $k$ cięć.

Tak więc:

$$|\mathcal{A}| \leq k \cdot 2 \cdot (|\mathcal{A}_o| + |\mathcal{B}_o| - 2) + 1 \leq 4k \cdot |\mathcal{A}_o| \ .$$

■

Powyższe oszacowanie współczynnika aproksymacji jest asymptotycznie dokładne.

**Lemat 1.4.7** *Współczynnik aproksymacji algorytmu* FASTHS *wynosi $\Omega(k)$.*

**Dowód:** Dla $A = ba\{ab\}^{k-1}$ i $B = \{ab\}^k$ algorytm FASTHS wyznacza zbiór $\Phi$ składający się z dwóch duetów $\{aa, ab\}$ i wspólny podział o rozmiarze $k+1$, natomiast optymalne rozwiązanie ma rozmiar 3. ■

Tak jak w przypadku algorytmu EDUCATED GREEDY, również i algorytm FASTHS możemy prostymi modyfikacjami, tak przekształcić aby rozwiązywał problemy $k$-SMCSP i $k$-RMCSP.

### 1.4.2 Efektywna implementacja algorytmu FASTHS

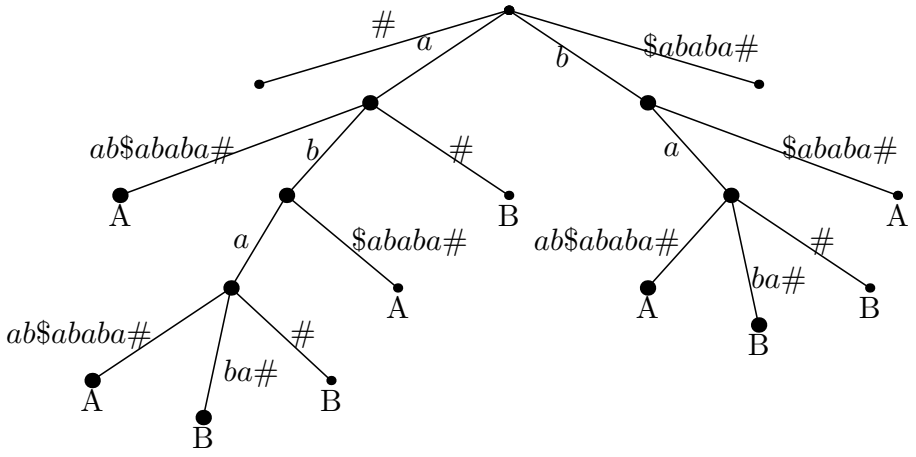
W tym rozdziale opiszemy w jaki sposób zaimplementować algorytm FASTHS, aby jego czas działania był liniowy ze względu na rozmiar danych wejściowych. Podstawą proponowanej implementacji są *drzewa sufiksowe*. Wykorzystamy znany fakt, że takie drzewa można skonstruować w czasie liniowym [44, 18].

Rozpoczynamy od skonstruowania zbioru $T'$ o rozmiarze linowym ze względu na długości słów $A$ i $B$. Niech \$ i $\#$ będą różnymi znakami, które nie występują w słowach $A$ i $B$. Obliczamy drzewo sufiksowe $\mathcal{T}$ dla słowa $C = A\$B\#$. Każdy liść drzewa $\mathcal{T}$ reprezentuje dokładnie jeden sufiks $C$. Liście drzewa $\mathcal{T}$, które odpowiadają sufiksom rozpoczynającym się w podsłowie $A$ słowa $C$ nazywamy $A$-liśćmi, natomiast liście odpowiadające sufiksom o początkach w $B$ nazywamy $B$-liśćmi. Dla każdego węzła $v$ z drzewa $\mathcal{T}$ obliczamy wartości $numA(v)$ i $numB(v)$, odpowiednio liczby $A$-liści i $B$-liści w poddrzewie $\mathcal{T}'$ o korzeniu w węźle $v$. Wartości $numA$ oraz $numB$ można obliczyć w czasie $\mathcal{O}(n)$ przechodząc drzewo metodą "postorder".

Przypomnijmy, że dla węzła $v$ drzewa $\mathcal{T}$, $L_v$ jest słowem powstającym przez sklejenie etykiet krawędzi na ścieżce od korzenia drzewa do węzła $v$. Dodatkowo niech $L'_v$, dla $v$ różnego od korzenia, będzie słowem $L_{parent(v)}$ powiększonym o pierwszy znak etykiety krawędzi $(parent(v), v)$, gdzie $parent(v)$, to węzeł-ojciec węzła $v$ w drzewie. Jeśli słowo $L'_v$ nie zawiera znaków \$ i #, to mówimy, że $v$ jest *węzłem właściwym*. Zauważmy, że dla każdego węzła właściwego $v$, mamy $numA(v) = \#_{L'_v}(A)$ i $numB(v) = \#_{L'_v}(B)$. Tak więc, jeśli $numA(v) \neq numB(v)$, to wiemy, że $L'_v \in T$. Powyższe pozwala za $T'$ wziąć następujący zbiór:

$$T' = \{L'_v \mid v \text{ jest węzłem właściwym i } numA(v) \neq numB(v)\}$$

Ponieważ drzewo sufiksowe zawiera $\mathcal{O}(n)$ węzłów, to zbiór $T'$ posiada co najwyżej $\mathcal{O}(n)$ elementów. Zauważmy także, że dla każdego słowa $X \in T_{\min}$ istnieje taki właściwy węzeł $v$, że $L'_v = X$ i $numA(v) \neq numB(v)$, co gwarantuje, że $T_{\min} \subseteq T'$.




Rysunek 1.7: Drzewo sufiksowe $\mathcal{T}$ dla słowa $C = abaab\$ababa\#$. Czarne kropy oznaczają węzły właściwe.

Rozważmy dla przykładu słowa $A = abaab$ i $B = ababa$. Drzewo sufiksowe słowa $C = A\$B\#$ przedstawione jest na rysunku 1.7, a zbiory, o których mowa w algorytmie, są następujące:

$$\begin{aligned} T' &= \{aa, aba, abaa, abab, ba, baa, bab\} \\ T_{\min} &= \{aa, ba\} \\ \Phi &= \{aa, ba\} \\ \mathcal{A} &= (ab, a, ab) \\ \mathcal{B} &= (ab, ab, a) \end{aligned}$$

Musimy jeszcze pokazać sposób reprezentowania (dynamicznego) zbioru $\Phi$, który umożliwi efektywne testowanie warunku $duos(X) \cap \Phi \neq \emptyset$ oraz wykonywanie cięć. W tym celu wykorzystamy strukturę danych używaną w rozwiązaniu problemu *podziału zbiorów* [19].

W problemie podziału zbiorów dany jest zbiór liczb całkowitych $\{1, \ldots, m\}$, na którym należy wykonać ciąg następujących operacji:

- $split(i)$ – podziel zbiór zawierający element $i$ na dwa podzbiory, jeden ze wszystkimi elementami mniejszymi od $i$, drugi ze wszystkimi elementami nie mniejszymi od $i$,
- $find(i)$ – podaj najmniejszej element w zbiorze zawierającym $i$.

Gabow i Tarjan [19] opisali strukturę danych, która pozwala na wykonanie każdej z powyższych operacji w zamortyzowanym czasie $\mathcal{O}(1)$.

W naszym przypadku, dla każdego podziału $\mathcal{A}$ i $\mathcal{B}$ utrzymujemy osobną strukturę danych, która przetrzymuje informację o cięciach w tych podziałach.

Początkowo każda z tych struktur zawiera tylko jeden zbiór $\{1, \ldots, n\}$. Za każdym razem, gdy dodajemy duet $cd$ do zbioru $\Phi$, wykonujemy cięcia na podziałach $\mathcal{A}$ i $\mathcal{B}$ w następujący sposób:

**Function** ADD-DUO ($cd$, $\mathcal{A}$, $\mathcal{B}$)

1 **for** *dla każdego wystąpienia duetu* $cd$ *w* $\mathcal{A}$ **do**
2 $A.split(j+1)$, gdzie $j$ jest pozycją wystąpienia duetu $cd$ w $A$ (to jest, $a_j a_{j+1} = cd$)
3 **end**
4 **for** *dla każdego wystąpienia duetu* $cd$ *w* $\mathcal{B}$ **do**
5 $B.split(j+1)$, gdzie $j$ jest pozycją wystąpienia duetu $cd$ w $B$ (to jest, $b_j b_{j+1} = cd$)
6 **end**

Ponieważ każdy duet występujący w $A$ i $B$ jest przeglądany co najwyżej raz, stąd całkowita liczba operacji $split$ wynosi $\mathcal{O}(n)$.

Dla słowa $a_i \ldots a_j = X$ (lub odpowiednio $b_i, \ldots, b_j = X$) zachodzi następujący fakt: $duos(X) \cap \Phi = \emptyset$ wtedy i tylko wtedy, gdy $A.find(i) = A.find(j)$ (lub odpowiednio $B.find(i) = B.find(j)$). Powyższy fakt umożliwia sprawdzanie warunku $duos(X) \cap \Phi \neq \emptyset$ w (zamortyzowanym) czasie stałym.

**Twierdzenie 1.4.8** *Algorytmu* FASTHS *można zaimplementować tak, żeby działał w czasie liniowym.*

W ten sposób otrzymujemy główny wynik tego rozdziału.

**Twierdzenie 1.4.9** *Istnieje $\mathcal{O}(k)$-aproksymacyjny algorytm rozwiązujący problem $k$-*SBR*, którego czas działania jest liniowy względem rozmiaru danych wejściowych.*

# Rozdział 2

# Problem MAX-NLS

## 2.1 Wprowadzenie

W tym rozdziale zajmujemy się problemami algorytmicznymi inspirowanymi zagadnieniami związanymi z porównywaniem podobieństw przestrzennych struktur cząsteczek ncRNA. Jest to motywowane faktem, iż bardzo często funkcję cząsteczki ncRNA można odgadnąć na podstawie jej struktury drugorzędowej. W książce [43] można znaleźć dokładne informacje na temat ncRNA.

Problemowi porównywania struktur ncRNA poświęcono bardzo wiele badań: [4, 5, 6, 16, 32, 33]. W najprostszym modelu matematycznym przyjmuje się, że struktura przestrzenna cząsteczek RNA jest zdeterminowana przez wiązania pomiędzy nukleotydami. Najczęściej występujące wiązania to: A–U, G–C and U–G (gdzie $\mathrm{A}$ to adenina, $\mathrm{C}$ – cytozyna, $\mathrm{G}$ – guanina, $\mathrm{U}$ – uracyl). Davydov i Batzoglou zaproponowali nowy model strukturalnego uliniowienia wielu sekwencji ncRNA [16, 17]. W tym modelu największą wspólną strukturę drugorzędową dla danych $k$ sekwencji ncRNA można wyliczyć w czasie $\mathcal{O}(n^{3k})$ [26]. Niestety, jeśli $k$ nie jest ograniczone, problem jest NP–trudny [16, 17], stąd poszukiwanie rozwiązań aproksymacyjnych. W pracy [16, 17] autorzy zaproponowali algorytm $\mathcal{O}(\log^2 m_{opt})$-aproksymacyjny, działający w czasie $\mathcal{O}(k \cdot n^5)$, gdzie $m_{opt}$ to rozmiar optymalnego rozwiązania.

W tym rozdziale znacząco poprawiamy wyniki z [16, 17] i podajemy algorytm $\mathcal{O}(\log m_{opt})$-aproksymacyjny, działający w czasie $\mathcal{O}(k \cdot n^2)$. Opisywane wyniki zostały opublikowane w pracy [31].

## 2.2 Definicje

Powiemy, że graf $G$ jest *uliniowiony*, gdy jego wierzchołki są jednoznacznie poetykietowane liczbami ze zbioru $\{1, 2, \ldots, n\}$, gdzie $n = |\mathbf{V}(G)|$. Tak poetykie-

towane grafy będziemy nazywać *grafami uliniowionymi*. W grafie uliniowionym wierzchołki utożsamiamy z ich numerami. Krawędź między wierzchołkami $i$ i $j$, dla $i < j$, oznaczamy jako parę $(i, j)$.

W interpretacji biologicznej wierzchołki grafu odpowiadają poszczególnym nukleotydom, natomiast krawędź pomiędzy wierzchołkami oznacza możliwość istnienia wiązania pomiędzy odpowiadającymi im nukleotydami. Dwie krawędzie grafu nazywamy *niezależnymi*, jeśli nie mają wspólnych końców. Uliniowiony graf $G$ nazywamy *krawędziowo niezależnym*, jeśli każde dwie jego krawędzie są niezależne (czyli graf $G$ jest skojarzeniem).

W grafie uliniowionym definiujemy następujące relacje na zbiorze krawędzi ([42]).

Niech $e = (i, j)$ i $e' = (i', j')$ będą dwiema niezależnymi krawędziami w uliniowionym grafie $G$. Piszemy, że:

- $e$ *poprzedza* $e'$ ($e < e'$) wtedy i tylko wtedy, gdy $i < j < i' < j'$,
- $e'$ *zawiera* $e$ ($e \sqsubset e'$) wtedy i tylko wtedy, gdy $i' < i < j < j'$,
- $e$ *przecina* $e'$ ($e \between e'$) wtedy i tylko wtedy, gdy $i < i' < j < j'$.

Dla danej relacji $\mathrm{R} \in \{<, \sqsubset, \between\}$, dwie krawędzie $e$ i $e'$ są $\mathrm{R}$-*porównywalne* jeśli $e \,\mathrm{R}\, e'$ lub $e' \,\mathrm{R}\, e$.
Zauważmy, że dowolne dwie niezależne krawędzie są $\mathrm{R}$-porównywalne, dla pewnej relacji $\mathrm{R} \in \{<, \sqsubset, \between\}$. Zauważmy, że relacje $<$ oraz $\sqsubset$ uzupełnione o zwrotność, to częściowe porządki.

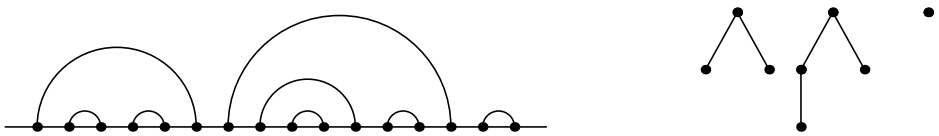
Niezależny krawędziowo, uliniowiony graf $G$ nazywamy $\mathcal{R}$-*porównywalnym* dla niepustego zbioru relacji $\mathcal{R} \subseteq \{<, \sqsubset, \between\}$, jeśli dowolne dwie krawędzie $e_1, e_2 \in \mathbf{E}(G)$, są $\mathrm{R}$-porównywalne, dla pewnej relacji $\mathrm{R} \in \mathcal{R}$.

Niech $G$ będzie grafem uliniowionym. *Szerokością* (odpowiednio *wysokością*) grafu $G$ nazywamy rozmiar najliczniejszego $\{<\}$–porównywalnego (odpowiednio $\{\sqsubset\}$- porównywalnego) podzbioru $\mathbf{E}(G)$.

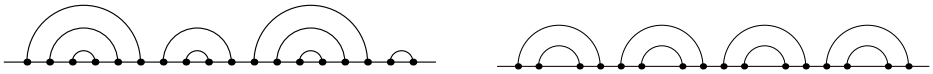
*Usunięciem wierzchołka* $i$ z grafu uliniowionego $G$ nazywamy operację polegającą na:

- usunięciu wszystkich krawędzi incydentnych z wierzchołkiem $i$,
- usunięcie wierzchołka $i$,
- zmniejszeniu o 1 numerów wszystkich wierzchołków o numerach większych od $i$.

Możemy już teraz zdefiniować pojęcie *występowania* grafu uliniowionego w innym grafie.



Rysunek 2.1: Przykład zagnieżdżonego grafu uliniowionego i jego drzewowa reprezentacja.




Rysunek 2.2: Przykłady grafów typów FLG i LLG. Oba grafy mają szerokość 4. Graf typu LLG ma wysokość 2.

Niech $G_1$ i $G_2$ będą grafami uliniowionymi. Piszemy, że graf $G_1$ *występuje* w $G_2$ (lub $G_1$ jest *podgrafem* $G_2$) jeśli $G_1$ można otrzymać z $G_2$ poprzez usunięcie pewnych krawędzi i wierzchołków.

Powiemy, że graf uliniowiony $G$ *nie zawiera przecięć*, jeśli nie zawiera krawędzi $e$ i $e'$ takich, że $e \between e'$.
Grafy uliniowione, które są $\{<, \sqsubset\}$–porównywalne, nazywamy zagnieżdżonymi grafami uliniowionymi i mówimy, że są typu NLG (ang. Nested Linear Graph).
$\{\sqsubset\}$-porównywalny podgraf grafu uliniowionego $G$ nazywamy *zagnieżdżoną pętlą*. Krawędź $(i, j)$ w zagnieżdżonej pętli o największej różnicy $j - i$ nazywamy *zewnętrzną krawędzią pętli*, a różnicę $j - i$ nazywamy *średnicą* tej pętli.
Graf typu NLG może być utożsamiany z ukorzenioną kolekcją drzew (zobacz rysunek 2.1). Każda krawędź w grafie odpowiada jednemu węzłowi drzewa. Krawędź $e$ jest ojcem krawędzi $e'$ w drzewie wtedy i tylko wtedy, gdy $e' \sqsubset e$ i nie istnieje krawędź $e''$ taka, że $e' \sqsubset e'' \sqsubset e$. Każda krawędź nie zawarta w innej jest korzeniem pewnego drzewa.
Graf $G$ typu NLG o $n$ wierzchołkach można również reprezentować przez słowo Dyck'a o długości $2n$ nad alfabetem $\{a, b\}$ — symbol $a$ odpowiada nawiasowi otwierającemu, symbol $b$ nawiasowi zamykającemu, a para odpowiadających sobie nawiasów reprezentuje krawędź grafu. W następnych paragrafach będziemy często utożsamiać grafy typu NLG z odpowiadającymi im słowami Dyck'a.

Graf $G$ typu NLG jest *płaski* (typu FLG, z ang. Flat Linear Graph), jeśli można go zapisać jako słowo $a^{h_1}b^{h_1}\ a^{h_2}b^{h_2}\ \ldots\ a^{h_k}b^{h_k}$, dla pewnych, dodatnich liczb całkowitych $h_1, h_2, \ldots, h_k$.

Graf $G$ typu FLG nazywamy *równym* (typu LLG, z ang. Level Linear Graph), jeśli można go zapisać jako słowo $(a^h b^h)^w$, dla pewnych dodatnich liczb całkowitych $h$ i $w$, odpowiednio *wysokości* i *szerokości* grafu.

*Sygnaturą* uliniowionego grafu $G$ nazywamy funkcję $s : \mathbb{N} \to \mathbb{N}$ taką, że

Rysunek 2.3: Najszersze podgrafy typu LLG o wysokości 2 (po lewej) i wysokości 3 (po prawej). Sygnatura grafu na rysunku ma następującą postać: $s(1) = 5$, $s(2) = 4$, $s(3) = 3$, $s(4) = 0$, itd.

$s(h)$ jest równe największej szerokości podgrafu typu LLG grafu $G$, o wysokości równej $h$. Gdy taki podgraf nie istnieje, przyjmujemy $s(h) = 0$. Rysunek 2.3 ilustruje pojęcie sygnatury.

W tym rozdziale rozważamy również pewną specjalną rodzinę grafów uliniowionych, w których zbiory krawędzi są definiowane na podstawie etykiet wierzchołków.

Niech $S = (a_1, \ldots, a_n)$ będzie słowem nad skończonym alfabetem $\Sigma$ i niech $\xi \subseteq \Sigma^2$ będzie symetryczną relacją nad $\Sigma$. Przez $\mathbf{G}_\xi(S)$ oznaczamy uliniowiony graf $n$–wierzchołkowy, w którym $p$ i $q$ są połączone krawędzią wtedy, gdy $p \neq q$ i $a_p \, \xi \, a_q$. Grafy tego typu nazywamy *ograniczonymi grafami uliniowionymi* i mówimy, że są typu RLG (ang. Restricted Linear Graph).
Szczególnym przypadkiem grafów RLG są grafy dla słów nad alfabetem $\Sigma_{\text{RNA}} = \{\text{A}, \text{U}, \text{G}, \text{C}\}$ i relacji

$$\xi_{\text{RNA}} = \{(\text{A}, \text{U}), (\text{U}, \text{A}), (\text{U}, \text{G}), (\text{G}, \text{U}), (\text{G}, \text{C}), (\text{C}, \text{G})\},$$

które służą do modelowania sekwencji ncRNA.

Przykładowy graf typu RLG dla relacji $\Sigma_{\text{RNA}}$ jest przedstawiony na rysunku 2.4.

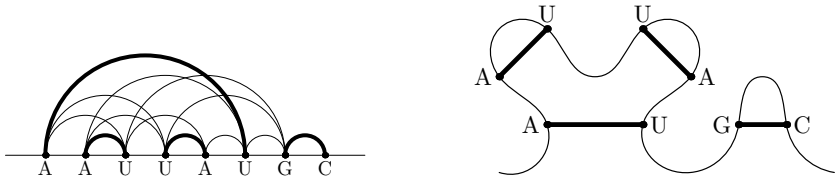
W tym rozdziale rozważamy następujące problemy:

**Problem 2.1** (MAX-NLS)

**Dane:** *Zbiór grafów uliniowionych* $\mathcal{G} = \{G_1, \ldots, G_k\}$.

**Wynik:** *Najliczniejszy pod względem liczby krawędzi graf* $G'$ *typu* NLG *i taki, że* $G'$ *jest podgrafem każdego grafu* $G_i$*. Jeśli istnieje wiele podgrafów o tej własności, to rozwiązaniem może być dowolny z nich.*

Problem MAX-NLS po raz pierwszy został zdefiniowany w pracy [16]. Autorzy zaproponowali wykorzystanie problemu MAX-NLS do rozwiązywania problemu strukturalnego uliniowienia ncRNA. Dla danego uliniowionego grafu $G$, podgrafy typu NLG odpowiadają różnym przestrzennym strukturom, które może przyjmować sekwencja, a krawędzie odpowiadają wiązaniom stabilizującym cząsteczkę ncRNA (zobacz rys. 2.4).




Rysunek 2.4: Przykład grafu uliniowionego dla sekwencji AAUUAUGC, rozwiązanie problemu MAX-NLS i odpowiadające mu powiązania między nukleotydami.

**Problem 2.2** (MAX-LLS)

**Dane:** *Zbiór grafów uliniowionych* $\mathcal{G} = \{G_1, \ldots, G_k\}$.

**Wynik:** *Najliczniejszy pod względem liczby krawędzi graf* $G'$ *typu* LLG *i taki, że* $G'$ *jest podgrafem każdego grafu* $G_i$*. Jeśli istnieje wiele podgrafów o tej własności, to rozwiązaniem może być dowolny z nich.*

Grafy typu LLG zostały wprowadzone w pracy [16] w celu uzyskania algorytmu aproksymacyjnego dla problemu MAX-NLS. W tej pracy również w rozwiązywaniu problemu MAX-NLS posługujemy się rozwiązaniem problemu MAX-LLS.

**Problem 2.3** (MNL)

**Dane:** *Graf uliniowiony* $G$.

**Wynik:** *Najliczniejszy pod względem liczby krawędzi* $\sqsubset$*–porównywalny graf* $G'$*, który jest podgrafem grafu* $G$.

W pracy [6] problem MNL został użyty w rozwiązaniu problemu przestrzennego uliniowienia. Autorzy zaproponowali w niej algorytm o złożoności czasowej $\mathcal{O}(n^2)$ (gdzie $n$ oznacza liczbę wierzchołków grafu) dla tego problemu.

## 2.3 Aproksymacyjne rozwiązanie problemu MAX-NLS

Problem MAX-NLS można rozwiązać w przybliżony sposób z pomocą dokładnego rozwiązania problemu MAX-LLS. Ten sam schemat postępowania został wcześniej zaproponowany w pracy [16]. Podsumowanie najważniejszych wyników dla problemu MAX-NLS zawiera tabela z rysunku 2.5. Ostatni z tych wyników jest omawiany w tej rozprawie. W kolejnych podrozdziałach opisujemy w jaki sposób efektywnie rozwiązywać problem MAX-LLS oraz dokonujemy analizy współczynnika aproksymacji tak otrzymanego rozwiązania dla problemu MAX-NLS.

| | współczynnik aproksymacji | złożoność czasowa |
|---|---|---|
| | 1 | $O(n^{3k})$ [26] |
| | $O(\log^2 m_{opt})$ | $O(k \cdot n^5)$ [16, 17] |
| nasze wyniki | $O(\log m_{opt})$ | $O(k \cdot n^2)$ [31] |

Rysunek 2.5: Podsumowanie wyników dotyczących problemu MAX-NLS.

### 2.3.1 Problem MAX-LLS

Algorytm zaproponowany w pracy [16], rozwiązujący problem MAX-LLS dla $k$ grafów uliniowionych o $n$ wierzchołkach każdy, działa w czasie $\mathcal{O}(k \cdot n^5)$.

W tym rozdziale opiszemy algorytm, który rozwiązuje problem MAX-LLS w czasie $\mathcal{O}(k \cdot n^2)$. Niech $G_1, \ldots, G_k$ będą $n$-wierzchołkowymi grafami uliniowionymi. Pierwszym krokiem algorytmu jest wyznaczenie dla każdego grafu $G_i$ jego sygnatury, która opisuje wszystkie maksymalne rozwiązania problemu MAX-LLS. W drugim kroku obliczamy wspólną sygnaturę dla wszystkich $k$ grafów. W ten sposób otrzymamy parametry opisujące maksymalne rozwiązania MAX-LLS$(G_1, G_2, \ldots, G_k)$.

Dla każdego grafu uliniowionego $G_i$ jego sygnatura jest wyznaczana w następujący sposób:

- W pierwszym kroku obliczamy tablicę $\mathrm{MNL}_i$, opisującą najliczniejsze podgrafy $\{\sqsubset\}$-porównywalne dla grafu $G_i$. Niech $G'_{i,p,q}$ będzie uliniowionym podgrafem $G_i$ indukowanym przez wierzchołki $p, \ldots, q$. Dla $1 \leq p < q \leq n$, $\mathrm{MNL}_i[p,q]$ jest równe rozmiarowi (w sensie liczby krawędzi) najliczniejszego podgrafu $\{\sqsubset\}$-porównywalnego grafu $G'_{i,p,q}$ (dla $p \geq q$ przyjmujemy, że $\mathrm{MNL}_i[p,q] = 0$).

  Tablicę $\mathrm{MNL}_i$ możemy wyznaczyć z pomocą programowania dynamicznego. Jeśli $(p,q) \in \mathbf{E}(G_i)$ to

  $$\mathrm{MNL}_i[p,q] = \mathrm{MNL}_i[p+1, q-1] + 1.$$

  W przeciwnym przypadku:

  $$\mathrm{MNL}_i[p,q] = \max(\mathrm{MNL}_i[p+1,q],\ \mathrm{MNL}_i[p,q-1]).$$

  Zauważmy, że ten krok wymaga czasu $\mathcal{O}(n^2)$.

- Obliczamy tablicę $\mathrm{NLW}_i$. Dla $1 \leq p \leq n$ i $1 \leq h \leq \frac{n}{2}$, $\mathrm{NLW}_i[p,h]$ jest równa najmniejszej liczbie całkowitej $j$, takiej, że podgraf $G'_{i,p,p+j}$ zawiera

podgraf $\{\sqsubset\}$-porównywalny o rozmiarze $h$, lub 0, jeśli taka liczba $j$ nie istnieje.

Tablicę $\text{NLW}_i$ można obliczyć wykonując następujący algorytm:

**Algorytm 2.3.1**: COMPUTENLW($\text{MNL}_i$)

```
1  wypełnij wszystkie pozycje tablicy NLW_i wartościami 0
2  for h = 1 to ⌊n/2⌋ do
3      for p = n − 1 downto 1 do
4          if NLW_i[p + 1, h] > 0 then
5              NLW_i[p, h] = NLW_i[p + 1, h] + 1
6          else if MNL_i[p, n] ≥ h then
7              NLW_i[p, h] = n − p
8          while NLW_i[p, h] > 1 ∧ MNL_i[p, p + NLW_i[p, h] − 1] ≥ h do
9              NLW_i[p, h] = NLW_i[p, h] − 1
10         end
11     end
12 end
```

Przyjrzyjmy się dokładniej pętli `while`. Zauważmy, że dla $p$ i $h$, jeśli $\text{NLW}_i[p,h] > 0$ i $\text{NLW}_i[p+1,h] > 0$, to pętla `while` zostanie wykonana $\text{NLW}_i[p+1,h] - \text{NLW}_i[p,h] + 1$ razy. Jeśli $\text{NLW}_i[p,h] > 0$ i $\text{NLW}_i[p+1,h] = 0$, to pętla `while` zostanie wykonana $n-p-\text{NLW}_i[p,h]$ razy. Oczywiście, gdy $\text{NLW}_i[p,h] = 0$, pętla `while` nie zostanie wykonana ani razu. Stąd liczba wszystkich iteracji pętli `while` dla danego $h$ nie przekracza $n - 1 - \text{NLW}_i(1,h) \leq n$. Stąd całkowity czas potrzebny do obliczenia tablicy $\text{NLW}_i$ wynosi $\mathcal{O}(n^2)$.

- W kroku trzecim, dla każdego $p = 1, \ldots, n$ obliczamy sygnaturę podgrafu $G'_{i,p,n}$, $\text{SIG}_i[p,h]$. Tablice $\text{SIG}_i[p,h]$, dla $h = 1, \ldots, \lfloor \frac{n}{2} \rfloor$, można wyznaczyć w czasie $\mathcal{O}(n^2)$ z pomocą następującej formuły:

$$\text{SIG}_i[p,h] = \begin{cases} \text{SIG}_i[p + \text{NLW}_i[p,h] + 1, h] + 1, & \text{jeśli } \text{NLW}_i[p,h] > 0 \\ 0 & \text{jeśli } \text{NLW}_i[p,h] = 0 \end{cases}$$

  Sygnatura grafu $G_i$ jest zapisana w $\text{SIG}_i[1,h]$, dla $h = 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$.

Łączny czas potrzebny na wykonanie wszystkich powyższych kroków wynosi $\mathcal{O}(n^2)$.

Wspólną sygnaturę SIG dla grafów $G_1, G_2, \ldots, G_k$ można otrzymać przy pomocy wzoru: $\text{SIG}[h] = \min_{i=1,\ldots,k} \text{SIG}_i[1,h]$, a rozmiar najliczniejszego wspólnego podgrafu typu LLG jest równy: $\max_{h=1,\ldots,\lfloor \frac{n}{2} \rfloor} h \cdot \text{SIG}[h]$. Używając stan-

dardowych technik dla programowania dynamicznego można zmodyfikować algorytmy obliczania tablic $\mathrm{SIG}_i$, $\mathrm{NLW}_i$ i $\mathrm{MNL}_i$ tak, by również zawierały informacje niezbędne do zrekonstruowania wspólnych podgrafów.

Warto zauważyć, że jeśli wejściowe grafy zawierają niewielką liczbę krawędzi, czyli $|\mathbf{E}(G)| \ll \mathcal{O}(n^2)$, to sygnaturę grafu można wyznaczyć szybciej. Stosując wzbogacone struktury słownikowe, można tak zapisać algorytmy opisywane w tym rozdziale, aby ich złożoność była zależna od liczby krawędzi. W ten sposób można wyznaczyć sygnaturę grafu o $n$ wierzchołkach i $m$ krawędziach w czasie $\mathcal{O}(m \log^2 n)$.

### 2.3.2 Współczynnik aproksymacji

W pracy [16] autorzy dowodzą, że dokładny algorytm dla problemu MAX-LLS jest $\mathcal{O}(\log^2 m_{opt})$-aproksymacyjnym algorytmem dla problemu MAX-NLS, gdzie $m_{opt}$ jest rozmiarem optymalnego rozwiązania problemu MAX-NLS.
Dowód przedstawiony w pracy [16] składa się z dwóch kroków. Najpierw analizuje się zależności pomiędzy optymalnymi rozwiązaniami problemów MAX-NLS i MAX-FLS, a w drugim etapie, pomiędzy optymalnymi rozwiązaniami problemów MAX-FLS i MAX-LLS. Każdy z tych kroków powoduje zwiększenie współczynnika aproksymacji o czynnik $\log m_{opt}$.

W tym rozdziale bezpośrednio badamy zależności pomiędzy optymalnymi rozwiązaniami dla problemów MAX-NLS i MAX-LLS, co pozwala zmniejszyć współczynnik aproksymacji do $\mathcal{O}(\log m_{opt})$.

**Twierdzenie 2.3.1** *Niech $G_1, \ldots, G_k$ będą $n$-wierzchołkowymi grafami uliniowionymi, $m_{opt}$ i $h_{opt}$, odpowiednio, rozmiarem i wysokością pewnego optymalnego rozwiązania problemu* MAX-NLS$(G_1, \ldots, G_k)$*, natomiast $l$ rozmiarem optymalnego rozwiązania problemu* MAX-LLS$(G_1, \ldots, G_k)$.
*Wówczas pomiędzy $m_{opt}$ i $l$ zachodzą następujące zależności*

$$m_{opt} \leq c_1 \cdot l \cdot \log(h_{opt} + 1) \leq c_2 \cdot l \cdot \log(m_{opt} + 1)$$

*dla pewnych dodatnich stałych całkowitych $c_1, c_2$.*

**Dowód:** Niech $T$ będzie lasem reprezentującym optymalne rozwiązanie problemu MAX-NLS$(G_1, \ldots, G_k)$. Oczywiście $T$ zawiera $m_{opt}$ węzłów.

Dla każdego węzła $v \in T$, przez $h(v)$ oznaczamy wysokość poddrzewa o korzeniu w $v$, a przez PATH$(v)$ oznaczamy dowolną, ale ustaloną ścieżkę długości $h(v)$ rozpoczynającą się w $v$ i kończącą się w liściu poddrzewa $v$. Można zauważyć, że PATH$(v)$ reprezentuje zagnieżdżoną pętlę o $h(v) + 1$ krawędziach.

Przez $L(h)$ oznaczamy zbiór wszystkich węzłów $T$ o wysokości $h$, $0 \leq h \leq h_{opt}$, a przez $S(h)$ zbiór wszystkich rozłącznych ścieżek PATH długości $h$, rozpoczynających się w węzłach z $L(h)$. Ponieważ każdy węzeł $v \in T$ jest zawarty w dokładnie jednym zbiorze $L(h)$, to $\sum_{i=0}^{h_{opt}} |L(i)| = m_{opt}$. Zauważmy, że dla dowolnego $h = 0, \ldots, h_{opt}$, zbiór $S(h)$ jest podgrafem typu $LLS$ o szerokości $|L(h)|$, wysokości $h$, oraz rozmiarze $s(h) = (h+1) \cdot |L(h)|$. Niech $h_{\text{MAX}}$ będzie wysokością, dla której wartość $s(h)$ jest największa. Ponieważ $l$ jest rozmiarem optymalnego rozwiązania problemu MAX-LLS, to $s(h_{\text{MAX}}) \leq l$. Dla dowolnego $i = 0, \ldots, h_{opt}$, mamy $|L(i)| \leq \frac{s(h_{\text{MAX}})}{i+1} \leq \frac{l}{i+1}$, i dalej

$$m_{opt} = \sum_{i=0}^{h_{opt}} |L(i)| \leq \sum_{i=0}^{h_{opt}} \frac{l}{i+1} \leq l \cdot \sum_{i=0}^{h_{opt}} \frac{1}{i+1}$$

Suma $\sum_{i=0}^{h_{opt}} \frac{1}{i+1} = \sum_{i=1}^{h_{opt}+1} \frac{1}{i}$, to $(h_{opt}+1)$-sza liczba harmoniczna. Zatem

$$l \cdot \sum_{i=0}^{h_{opt}} \frac{1}{i+1} \leq c_1 \cdot l \cdot \log(h_{opt}+1) \leq c_2 \cdot l \cdot \log(m_{opt}+1)$$

Dla pewnych stałych $c_1, c_2$. ■

Powyższe ograniczenie jest dla naszego algorytmu asymptotycznie dokładne. Dla rodziny drzew zdefiniowanych w [16] współczynnik aproksymacji wynosi $\Theta(\log m_{opt})$.

### 2.3.3 Ograniczone grafy uliniowione

W tym rozdziale zajmujemy się problemem MAX-NLS dla ograniczonych grafów uliniowionych. Okazuje się, że dla takich grafów istnieje algorytm $\mathcal{O}(1)$–aproksymacyjny i działający w czasie $\mathcal{O}(kn)$.

Dla $(p,q) \in \Sigma^2$, niech $\text{MNL}_{(p,q)}(S)$ będzie rozwiązaniem problemu MNL dla podgrafu grafu $\mathbf{G}_\xi(S)$ składającego się jedynie z krawędzi $(i,j)$ takich, że $a_i = p$ i $a_j = q$ (lewy koniec krawędzi jest oznaczony symbolem $p$, natomiast prawy symbolem $q$). Niech $\text{MNL}_\xi(S)$ będzie najliczniejszym rozwiązaniem spośród wszystkich $\text{MNL}_{(p,q)}(S)$, dla $(p,q) \in \xi$.

**Twierdzenie 2.3.2** $\text{MNL}_\xi(S)$ *można wyznaczyć w czasie* $\mathcal{O}(n)$*, przy założeniu że* $|\xi| = O(1)$.

**Dowód:** Dla $p \in \Sigma$, tablice $l[i,p] = \#_p(a_1, \ldots, a_i)$ i $r[i,p] = \#_p(a_i, \ldots, a_n)$, $1 \leq i \leq n$, można obliczyć w czasie $\mathcal{O}(n)$. Pozwalają one w prosty sposób policzyć rozmiary zbiorów $\mathbf{E}(\text{MNL}_\xi(S))$. Do tego celu wykorzystujemy formułę:

$$|\mathbf{E}(\text{MNL}_\xi(S))| = \max_{i \in 1,\ldots,n-1,\ (p,q) \in \xi} \min(l[i,p], r[i+1,q]).$$

Ponieważ $\xi$ jest rozmiaru $\mathcal{O}(1)$, to całkowity czas działania powyższego algorytmu wynosi $\mathcal{O}(n)$. ■

**Lemat 2.3.3** *Załóżmy, że* $(p,q) \in \xi$ *oraz* $(q,p) \in \xi$. *Jeśli* $\#_p(S) \geq l \geq 1$ *i* $\#_q(S) \geq l$, *to*

$$|\mathbf{E}(\mathrm{MNL}_\xi(S))| \geq \frac{l}{2}.$$

**Dowód:** Łatwo zauważyć, że istnieje indeks $i$ ($1 \leq i \leq n-1$) taki, że

$$\#_p(a_1, \ldots, a_i) = \#_q(a_{i+1}, \ldots, a_n) = c,$$

dla pewnego $c \geq 0$. Rozpatrujemy dwa przypadki:

- Jeśli $c \geq \frac{l}{2}$, to mamy $|\mathbf{E}(\mathrm{MNL}_{(p,q)})| \geq c \geq \frac{l}{2}$.
- Jeśli $c < \frac{l}{2}$, to w tym wypadku

  $$\#_q(a_1, \ldots, a_i) \geq l - c \quad \text{i} \quad \#_p(a_{i+1}, \ldots, a_n) \geq l - c.$$

  Stąd $|\mathbf{E}(\mathrm{MNL}_{(q,p)})| \geq l - c \geq \frac{l}{2}$.

■

**Twierdzenie 2.3.4** *Niech* $\Sigma$ *będzie skończonym alfabetem,* $\xi \subseteq \Sigma \times \Sigma - \{(i,i) : i \in \Sigma\}$ *relacją symetryczną na* $\Sigma$. *Dla całkowitych* $k, n > 0$, *niech* $S_1, S_2, \ldots, S_k$ *będą słowami długości* $n$ *nad* $\Sigma$. *Istnieje algorytm aproksymacyjny dla problemu* MAX-NLS$(\mathbf{G}_\xi(S_1), \ldots, \mathbf{G}_\xi(S_k))$ *o współczynniku aproksymacji* $|\xi|$ *i działający w czasie* $\mathcal{O}(kn)$.

**Dowód:** Dla każdej sekwencji $S_i$ możemy w czasie $\mathcal{O}(n)$ obliczyć $\mathrm{MNL}_\xi(S_i)$. Następnie znajdujemy indeks $j$, $1 \leq j \leq k$, dla którego wartość $|\mathbf{E}(\mathrm{MNL}_\xi(S_j))|$ jest minimalna, a za wynik działania algorytmu wybieramy $\mathrm{MNL}_\xi(S_j)$. Oczywiście $\mathrm{MNL}_\xi(S_j)$ jest wspólnym podgrafem dla $\mathbf{G}_\xi(S_1), \ldots, \mathbf{G}_\xi(S_k)$, a całkowity czas działania algorytmu wynosi $\mathcal{O}(kn)$.

Niech $S_j = (a_1, \ldots, a_n)$ i niech $e = |\mathbf{E}(\text{MAX-NLS}(\mathbf{G}_\xi(S_j)))|$. Wystarczy pokazać, że $|\mathbf{E}(\mathrm{MNL}_\xi(S_j))| \geq \frac{e}{|\xi|}$. Każdą krawędź $(r,s)$ należącą do rozwiązania MAX-NLS$(\mathbf{G}_\xi(S_j))$ oznaczamy etykietą $(\min(a_r, a_s), \max(a_r, a_s))$, przy dowolnym uporządkowaniu $\Sigma$. Ponieważ istnieje co najwyżej $|\xi|/2$ różnych etykiet na $e$ krawędziach, stąd istnieje para znaków $(p,q) \in \xi$ taka, że co najmniej $2\frac{e}{|\xi|}$ krawędzi jest oznaczonych etykietą $(p,q)$. Stąd $\#_p(S_j) \geq 2\frac{e}{|\xi|}$ i $\#_q(S_j) \geq 2\frac{e}{|\xi|}$. Korzystając z lematu 2.3.3 dostajemy $|\mathbf{E}(\mathrm{MNL}_\xi(S_j))| \geq \frac{e}{|\xi|}$. ■

Powyższe twierdzenie pokazuje, że podejście zaproponowane w pracy [16] jest zbyt ogólne i abstrakcyjne w przypadku modelowania strukturalnego uliniowienia ncRNA. Korzystając z założenia o ograniczonym alfabecie i własności

relacji $\xi$, można uzyskać szybsze rozwiązania i udowodnić lepsze współczynniki aproksymacji.

Bezpośrednio z twierdzenia 2.3.4 wynika, że dla sekwencji ncRNA współczynnik aproksymacji algorytmu opartego na wyznaczaniu $\mathrm{MNL}_{\xi}$ wynosi 6, ponieważ $|\xi_{\mathrm{RNA}}| = 6$. Biorąc pod uwagę specjalne własności relacji $\xi_{\mathrm{RNA}}$, można udowodnić istnienie nieznacznie lepszego współczynnika aproksymacji wynoszącego 4.

**Wniosek 2.3.5** *Istnieje algorytm aproksymacyjny ze współczynnikiem 4, który rozwiązuje problem* MAX-NLS$(\mathbf{G}_{\xi_{\mathrm{RNA}}}(S_1), .., \mathbf{G}_{\xi_{\mathrm{RNA}}}(S_k))$*, dla* $k$ *słów* $S_i \in \Sigma^n_{RNA}$.

# Rozdział 3

# Uwarunkowane cykle Eulera

## 3.1 Wprowadzenie

Wyznaczanie ścieżek (lub cykli) Eulera jest bardzo dobrze znanym problemem, który posiada wiele efektywnych i prostych rozwiązań ([45]). W tym rozdziale zajmujemy się wyznaczaniem cykli Eulera, które spełniają dodatkowe warunki. Bardzo podobne problemy kombinatoryczne są rozważane w biologii obliczeniowej. Jedną z metod odczytywania sekwencji DNA jest *sekwencjonowanie przez hybrydyzację* (SBH – ang. *Sequencing By Hybrydization*). W pracy [37] Pevzner pokazał, w jaki sposób zredukować rekonstrukcję sekwencji DNA przy użyciu metody SBH do wyznaczania ścieżki Eulera w grafie. Niestety standardowa metoda SBH pozwala poprawnie odczytać jedynie krótkie sekwencje. Zaproponowano wiele rozszerzeń tej metody, które redukują problemy z niejednoznacznością odczytu (np. [23], [24], [38]).

W pracy [38] wprowadzono problem *superścieżki Eulera* (ESP — ang. *Eulerian Superpath Problem*) polegający na znajdowaniu ścieżki Eulera, która zawiera w sobie wszystkie ścieżki z zadanego zbioru. Autorzy zaproponowali wielomianowy algorytm dla tego problemu w przypadku skierowanych grafów prostych, oraz rozwiązania heurystyczne dla multigrafów.

W pracy [34] autorzy rozważają złożoność problemu z negatywnymi warunkami, to jest problemu znajdowania ścieżki Eulera, która nie zawiera żadnej ścieżki z zadanego zbioru ścieżek. Autorzy dowodzą, że w takim przypadku problem jest NP–trudny, nawet jeśli każda z zabronionych ścieżek posiada dokładnie 3 wierzchołki.

W niniejszym rozdziale rozważamy problemy dotyczące cykli Eulera, zamiast ścieżek Eulera. Jednak wszystkie rozwiązania dotyczące uwarunkowanych cykli Eulera można łatwo przekształcić w rozwiązania dotyczące uwarunkowanych ścieżek Eulera. Opisujemy liniowy algorytm, który dla zadanego eulerowskiego

grafu prostego oraz zbioru ścieżek $\mathcal{P}$ sprawdza, czy istnieje cykl Eulera zawierający każdą ścieżkę ze zbioru $\mathcal{P}$. Następnie pokazujemy dowód NP–trudności tego problemu dla multigrafów. Przedstawiamy również liniowy algorytm, który dla zadanego eulerowskiego grafu prostego oraz ścieżki $\pi$ sprawdza, czy istnieje cykl Eulera, który nie zawiera ścieżki $\pi$.

## 3.2 Definicje

W tym rozdziale rozważamy proste grafy skierowane oraz multigrafy skierowane, czyli grafy z wielokrotnymi, równoległymi krawędziami. Zakładamy, że grafy nie zawierają pętli (czyli krawędzi typu ($u$,$u$)) oraz izolowanych wierzchołków.

**Fakt ([45])** *Graf $G$ jest grafem eulerowskim wtedy i tylko wtedy, gdy:*

- *$G$ jest silnie spójny, czyli dla każdej pary wierzchołków $u, v \in V$, istnieje ścieżka z $u$ do $v$,*
- *dla każdego wierzchołka $v \in V$, zachodzi $indeg(v) = outdeg(v)$.*

Piszemy, że ścieżka $\pi' = \langle v'_0, \ldots, v'_p \rangle$ jest zawarta w ścieżce $\pi = \langle v_0, \ldots, v_n \rangle$ jeśli ciąg $v'_0, \ldots, v'_p$ występuje jako spójny podciąg w ciągu $v_0, \ldots, v_n$, czyli istnieje indeks $k$, $0 \leq k \leq n - p$, taki, że dla każdego $0 \leq i \leq p$ mamy $v_{k+i} = v'_i$.

Analogicznie piszemy, że ścieżka $\pi' = \langle v'_0, \ldots, v'_p \rangle$ jest zawarta w cyklu $C = \langle v_0, \ldots, v_n = v_0 \rangle$, jeśli ciąg $v'_0, \ldots, v'_p$ występuje jako spójny podciąg w ciągu $v_0, \ldots, v_{n-1}, v_0, \ldots, v_n, v_0, v_1 \ldots$.
W niniejszym rozdziale rozważamy trzy następujące problemy:

**Problem 3.1 (EulRS)**
**Dane:** *Graf eulerowski $G$ oraz* wymagana *ścieżka $\pi = \langle v_0, \ldots, v_k \rangle$.*
**Pytanie:** *Czy w $G$ istnieje cykl Eulera $C$, który zwiera $\pi$?*

**Problem 3.2 (EulRMS)**
**Dane:** *Graf eulerowski $G$ oraz zbiór* wymaganych *ścieżek $\mathcal{P} = \{\pi_1, \pi_2, .., \pi_l\}$.*
**Pytanie:** *Czy w $G$ istnieje cykl Eulera $C$, który zawiera każdą ścieżkę $\pi_i \in \mathcal{P}$?*

**Problem 3.3 (EulFS)**
**Dane:** *Graf eulerowski $G$ oraz* zabroniona *ścieżka $\pi = \langle v_0, \ldots, v_k \rangle$.*
**Pytanie:** *Czy w $G$ istnieje cykl Eulera $C$, który nie zwiera $\pi$?*

Wszystkie powyższe problemu są sformułowane jako problemy decyzyjne, jednak wszystkie algorytmy z pracy mogą być łatwo zmodyfikowane tak, by wyznaczały cykle Eulera spełniające sformułowane warunki.

Tabela z rysunku 3.1 zawiera podsumowanie wyników przedstawionych w tym rozdziale.

| Problem | Grafy proste | Multigrafy |
|---|---|---|
| EulRS | $\mathcal{O}(n)$ | wielomianowy [38] |
| | | $\mathcal{O}(n)$ |
| EulRMS | $\mathcal{O}(n)$ | NP-zupełny |
| EulFS | $\mathcal{O}(n)$ | - |

Rysunek 3.1: Podsumowanie wyników dotyczących złożoności problemów uwarunkowanych cykli Eulera.

## 3.3 Cykle Eulera zawierające ścieżki z zadanego zbioru

Problem *EulRS*, czyli sprawdzenie, czy graf posiada cykl Eulera zawierający zadaną ścieżkę $\pi$, można prosto rozstrzygnąć w czasie liniowym (nawet dla multigrafów).

**Fakt** *Problem* EulRS *można rozwiązać w czasie liniowym ze względu na rozmiar grafu.*

Problem EulRS sprowadzamy do wyznaczania dowolnego cyklu Eulera w specjalnie skonstruowanym grafie $G'$. Graf $G'$ konstruujemy z $G$ przez zamianę krawędzi występujących w $\pi$ na pojedynczą krawędź pomiędzy wierzchołkami stanowiącymi początek i koniec ścieżki oraz usunięcie izolowanych wierzchołków. Graf $G$ posiada cykl Eulera zawierający $\pi$ wtedy i tylko wtedy, gdy $G'$ jest grafem eulerowskim.

Problem *EulRMS*, czyli sprawdzanie, czy w grafie istnieje cykl Eulera zawierający wszystkie ścieżki z zadanego zbioru jest nieznaczenie trudniejszy. W pracy [38] autorzy przedstawiają wielomianowy algorytm dla tego problemu. Pokażemy, w jaki sposób problem może zostać rozwiązany w czasie liniowym w przypadku grafów prostych, oraz dowód NP–zupełności tego problemu dla multigrafów.

**Twierdzenie 3.3.1** *Dla grafów prostych problem* EulRMS *posiada rozwiązanie liniowe ze względu na rozmiar grafu i sumę długości ścieżek.*

**Dowód:** Niech $G$ będzie grafem, a $\mathcal{P} = \{\pi_1, \ldots, \pi_l\}$ zbiorem wymaganych ścieżek. Pierwszy krok algorytmu polega na sprawdzeniu, czy wszystkie krawędzie

występujące w ścieżkach są krawędziami grafu $G$. Niech $E_{\mathcal{P}}$ będzie zbiorem krawędzi zawartych w ścieżkach z $\mathcal{P}$. Zbiór $E_{\mathcal{P}}$ może zostać wyznaczony w czasie liniowym, w tym samym czasie możemy również zweryfikować, czy $E_{\mathcal{P}} \subseteq E$.

Każdy cykl Eulera w grafie $G$ daje nam pewne (cykliczne) uporządkowanie krawędzi, w którym każda krawędź $e \in E$ ma dokładnie jednego następnika i poprzednika. Każda ze ścieżek z $\mathcal{P}$ nakłada na cykl dodatkowe ograniczenia — dla ścieżki $\pi_i = \langle v_0^i, v_1^i, v_2^i, \ldots, v_{k_i}^i \rangle$ następnikiem krawędzi $(v_0^i, v_1^i)$ powinna być krawędź $(v_1^i, v_2^i)$, krawędzi $(v_1^i, v_2^i)$ — $(v_2^i, v_3^i)$, itd.

Bardziej formalnie, niech $G_E = (E, E')$ będzie grafem takim, że $(e_1, e_2) \in E'$ wtedy i tylko wtedy, gdy $e_1 = (x, y)$ i $e_2 = (y, z)$, dla pewnych trzech wierzchołków $x, y, z \in V$, oraz istnieje ścieżka $\pi_i \in \mathcal{P}$ taka, że $\pi_i = \langle \ldots, x, y, z, \ldots \rangle$. Graf $G_E$ zawiera pełną informację o ograniczeniach nałożonych na cykl przez ścieżki z $\mathcal{P}$. Graf $G_E$ może zostać skonstruowany w czasie $\mathcal{O}(|V| + |E| + \sum_{\pi \in \mathcal{P}} |\pi|)$. Rozważmy cztery przypadki:

(i) Istnieje krawędź $e \in E$ taka, że $outdeg_{G_E}(e) > 1$ (lub $indeg_{G_E}(e) > 1$). Przykładowo niech $(e, e_1), (e, e_2) \in E'$, $e = (x, y)$, $e_1 = (y, z_1)$, $e_2 = (y, z_2)$, $z_1 \neq z_2$. Ponieważ $G$ jest grafem prostym, to nie zawiera cyklu Eulera, który zawiera jednocześnie ścieżki $\langle x, y, z_1 \rangle$ oraz $\langle x, y, z_2 \rangle$.

(ii) W grafie $G_E$ istnieje cykl elementarny $C$ taki, że $|C| < |E|$. Tak jak w poprzednim przypadku dowodzi to braku rozwiązania dla grafu $G$ i ścieżek $\mathcal{P}$.

(iii) W $G_E$ istnieje cykl elementarny $C$ i $|C| = |E|$. W taki przypadku cykl $C$ reprezentuje cykl Eulera w grafie $G$ o żądanych własnościach.

(iv) Krawędzie $G_E$ tworzą rozłączne wierzchołkowo ścieżki elementarne. Każda ścieżka w $G_E$ reprezentuje ciąg wierzchołków w $G$ i może zostać zastąpiona przez pojedynczą krawędź w $G$. Niech $G'$ oznacza graf skonstruowany z $G$ przez zamianę każdej ścieżki (usunięcie krawędzi) z $G_E$ przez pojedynczą krawędź i usunięcie izolowanych wierzchołków. Graf $G$ posiada cykl Eulera o żądanych ograniczeniach wtedy i tylko wtedy, gdy graf $G'$ jest eulerowski.

Podsumowując, algorytm przedstawiony na rysunku 3.2 rozwiązuje problem *EulRMS* dla grafów prostych.

Wszystkie kroki powyższego algorytmu można zaimplementować tak, by ich złożoność czasowa była liniowa względem rozmiaru grafu oraz sumy rozmiarów ścieżek. Tak więc całkowity czas działania algorytmu wynosi $\mathcal{O}(|E| + |V| + \sum_{\pi \in \mathcal{P}} |\pi|)$. ■

**Algorytm 3.3.1**: SOLVEEULRMS

**Dane**: prosty, eulerowski graf skierowany $G = (\mathbf{V}, \mathbf{E})$ oraz zbiór ścieżek $\mathcal{P} = \{\pi_1, \ldots, \pi_k\}$

```
1  wyznacz E_P
2  if E_P ⊈ E then return False
3  wyznacz graf G_E zdefiniowany w twierdzeniu 3.3.1
4  if istnieje v ∈ V(G_E) taki, że outdeg(v) > 1 lub indeg(v) > 1 then
5      return False (przypadek (i))
6  else if w G_E istnieje cykl elementarny C taki, że |C| < |E| then
7      return False (przypadek (ii))
8  else if w G_E istnieje cykl elementarny C taki, że |C| = |E| then
9      return True (przypadek (iii))
10 else
11     (przypadek (iv))
12     wyznacz graf G' zdefiniowany w twierdzeniu 3.3.1
13     if G' jest grafem eulerowskim then
14         return True
15     else
16         return False
17     end
18 end
```

Rysunek 3.2: Algorytm rozwiązujący problem *EulRMS* dla grafów prostych.

W praktycznych zastosowaniach (np. w biologii obliczeniowej) bardziej istotne znaczenie ma problem *EulRMS* dla multigrafów. Udowodnimy NP–zupełność problemu *EulRMS* dla multigrafów. W dowodzie wykorzystamy fakt, że problem *najkrótszego nadsłowa* (SSP — ang. *Shortest Superstring Problem*) jest NP-zupełny [20].

**Problem 3.4 (SSP)**

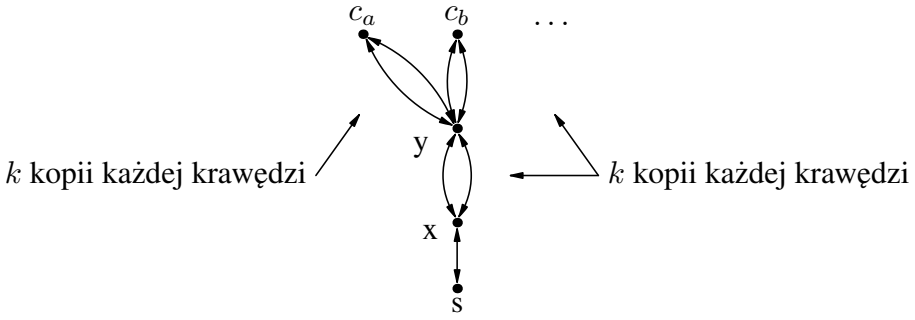
**Dane:** *Liczba naturalna $k$ oraz zbiór słów $\mathcal{S} = \{s_1, \ldots, s_n\}$ nad alfabetem $\Sigma$.*

**Pytanie:** *Czy istnieje słowo $w$ o długości nie większej niż $k$, które zawiera jako podsłowo każde słowo $s_i \in \mathcal{S}$.*

**Lemat 3.3.2 ([36])** *Problem SSP jest NP–zupełny, nawet jeśli wszystkie słowa wejściowe $s_i$ mają długość 3.*

**Twierdzenie 3.3.3** *Problem* EulRMS *dla multigrafów jest NP–zupełny, nawet gdy wymagane ścieżki mają długości nie większe od 10.*

**Dowód:** Łatwo zauważyć, że problem *EulRMS* jest w NP.
Pokażemy teraz wielomianową redukcję SSP do EulRMS. Dla danego zbioru słów $\mathcal{S} = \{s_1, \ldots, s_n\}$, $|s_i| = l_i$, oraz liczby całkowitej $k$, naszym celem jest skonstruowanie instancji $(G, \mathcal{P})$ problemu *EulRMS*, dla której istnieje pozytywna odpowiedź wtedy i tylko wtedy, gdy $\mathcal{S}$ posiada nadsłowo długości co najwyżej $k$.
Niech $G = (\mathbf{V}, \mathbf{E})$ będzie multigrafem skierowanym, w którym $V = \{s, x, y\} \cup \{c_z : z \in \Sigma\}$. Graf $G$ posiada $k$ kopii każdej krawędzi postaci: $(y, c_z)$, $(c_z, y)$, $(y, c_z)$, $(c_z, y)$ (dla $z \in \Sigma$), oraz $(x, y)$, $(y, x)$, oraz dokładnie po jednej krawędzi $(s, x)$ i $(x, s)$ — zobacz rysunek 3.3.




Rysunek 3.3: Konstrukcja grafu z dowodu NP–zupełności problemu EulRMS.

Dla każdego słowa $s_i \in \mathcal{S}$ tworzymy jedną ścieżkę $\pi_i$:

$$\pi_i = x, y, c_{s_i[1]}, y, x, y, c_{s_i[2]}, \ldots, c_{s_i[l_i-1]}, y, x, y, c_{s_i[l_i]}$$

Jeśli istnieje cykl Eulera $C$ w grafie $G$ i zawiera on wszystkie ścieżki z $\mathcal{P} = \{\pi_1, \ldots, \pi_n\}$, to możemy skonstruować słowo $T'$, które zawiera wszystkie słowa ze zbioru $\mathcal{S}$.

Niech $C = \langle v_0, v_1, \ldots, v_m \rangle$, $v_0 = v_m = s$, i niech $\Phi$ będzie następująco zdefiniowaną funkcją:

$$\Phi(i) = \begin{cases} v_{i+2} & \text{jeśli } v_i = x, v_{i+1} = y, v_{i+2} \in \{c_z : z \in \Sigma\} \\ \epsilon & \text{w przeciwnym przypadku} \end{cases}$$

Słowo $T' = \Phi(1)\Phi(2), \ldots, \Phi(m-2)$ jest poszukiwanym nadsłowem.

Multigraf $G$ zawiera $k$ kopii krawędzi $(x, y)$, stąd $T'$ ma długość co najwyżej $k$. Jeśli $C$ zawiera ścieżkę $\pi_i$ na pozycji $j$, to słowo $s_i$ występuje jako prefiks słowa $\Phi(j)\Phi(j+1), \ldots, \Phi(m-2)$. Tak więc $T'$ zawiera wszystkie słowa ze zbioru $\mathcal{S}$.

W ten sam sposób możemy z dowolnego nadsłowa $T$ o długości nie większej niż $k$ i zawierającego wszystkie słowa ze zbioru $\mathcal{S}$ otrzymać cykl Eulera zawierający wszystkie ścieżki z $\mathcal{P}$. ■

## 3.4 Cykle Eulera unikające zabronionej ścieżki

W tym rozdziale zajmujemy się negatywnymi ograniczeniami na cykle Eulera — czyli problemem *EulFS*. Niestety negatywne ograniczenia są trudniejsze w analizie, nawet w przypadku, gdy mamy tylko jedną zabronioną ścieżkę.

Warto zauważyć, że negatywne ograniczenia nie są całkowicie abstrakcyjnym problemem. Pytanie, czy istnieje cykl Eulera nie zawierający zadanej ścieżki, jest równoważne pytaniu, czy wszystkie cykle Eulera zawierają zadaną ścieżkę. W odniesieniu do metody *SBH* problem *EulFS* może być wykorzystywany do badania jednoznaczności odczytu.

W tym rozdziale zakładamy, że wszystkie krawędzie ścieżki $\pi$ są zawarte w $E(G)$. W przeciwnym przypadku problem jest trywialny.

**Lemat 3.4.1** *Dla grafów prostych istnieje algorytm wielomianowy rozwiązujący problem* EulFS.

**Dowód:** Zauważmy, że jeśli w grafie $G$ istnieje cykle Eulera $C'$, który nie zawiera ścieżki $\pi$, to istnieje wierzchołek $v_i$ $(0 \leq i < k-1)$ oraz wierzchołek $x \neq v_{i+2}$ taki, że $C'$ zawiera ścieżkę $\langle v_i, v_{i+1}, x\rangle$. Łatwo możemy odwrócić to stwierdzenie i zauważyć, że jeśli w grafie prostym $G$ istnieje cykl Eulera $C'$, który zawiera ścieżkę $\langle v_i, v_{i+1}, x\rangle$ $(x \neq v_{i+2})$, to cykl $C'$ nie zawiera ścieżki $\pi$. W takim razie możemy rozwiązać problem *EulRS* dla wszystkich możliwych ścieżek postaci $\langle v_i, v_{i+1}, x\rangle$. Jeśli dla pewnej ścieżki otrzymamy odpowiedź pozytywną, to możemy również udzielić odpowiedzi pozytywnej dla problemu *EulFS*. W przeciwnym przypadku możemy uznać, że każdy cykl Eulera musi zawierać ścieżkę $\pi$. Ponieważ ścieżek postaci $\langle v_i, v_{i+1}, x\rangle$ jest co najwyżej $\mathcal{O}(|k| \cdot |V|)$, a bez utraty ogólności możemy przyjąć, że $k \leq |E|$, to problem *EulFS* ma rozwiązanie w czasie $\mathcal{O}(|E|^2 \cdot |V|)$. ■

Przedstawimy teraz algorytm działający w czasie $\mathcal{O}(|V| + |E|)$ i tym samym udowodnimy twierdzenie:

**Twierdzenie 3.4.2** *Problem* EulFS *dla grafów prostych ma rozwiązanie w czasie liniowym ze względu na rozmiar grafu $G$ i długość zabronionej ścieżki $\pi$.*

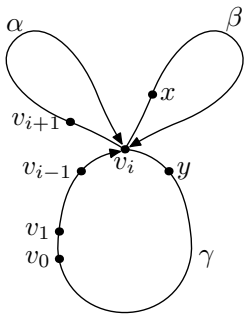
W dowodzie twierdzenia wykorzystamy dwa pomocnicze lematy.

**Lemat 3.4.3** *Niech $G = (\mathbf{V}, \mathbf{E})$ będzie skierowanym grafem eulerowskim, a $\pi = \langle v_0, \ldots, v_k\rangle$ zabronioną ścieżką. Jeśli istnieje wierzchołek $v_i$ z $outdeg(v_i) > 2$, $0 < i < k$, to $G$ posiada cykl Eulera nie zawierający ścieżki $\pi$.*

**Dowód:** Rozważmy cykl Eulera $C'$ zawierający ścieżkę $\pi$. Cykl $C'$ można zapisać w postaci:

$$C' = \langle v_0, \ldots, v_{i-1}, v_i, v_{i+1},\ \alpha\ , v_i, x,\ \beta\ , v_i, y,\ \gamma \rangle$$

gdzie $x, y \neq v_{i+1}$, $\alpha$, $\beta$, $\gamma$ to podścieżki z rysunku 3.4. Ścieżka $\pi$ jest prefiksem cyklu.




Rysunek 3.4: Cykl $C'$ z dowodu lematu 3.4.3.

Możemy jednak tak zmodyfikować cykl $C'$, aby otrzymać inny cykl Eulera $C$, który nie zawiera ścieżki $\pi$. Mianowicie:

$$C = \langle v_0, \ldots, v_{i-1}, v_i, x,\ \beta\ , v_i, v_{i+1},\ \alpha\ , v_i, y,\ \gamma \rangle$$

■

**Lemat 3.4.4** *Niech $G = (\mathbf{V}, \mathbf{E})$ będzie skierowanym grafem eulerowskim, a $\pi = \langle v_0, \ldots, v_k \rangle$ zabronioną ścieżką. Jeśli $\pi$ jest cyklem prostym oraz istnieje wierzchołek $v_i$, $0 < i < k$, dla którego $outdeg(v_i) > 1$, to $G$ posiada cykl Eulera nie zawierający ścieżki $\pi$.*

**Dowód:** Rozważmy cykl Eulera $C'$ zawierający ścieżkę $\pi$. Cykl $C'$ można zapisać w postaci:

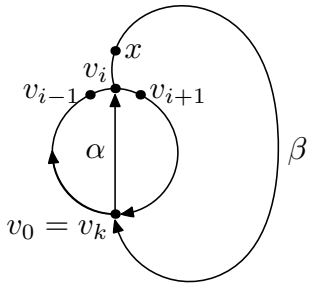
$$C' = \langle v_0, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_0,\ \alpha\ , v_i, x,\ \beta \rangle$$

Tak jak poprzednio, można tak zmodyfikować cykl $C'$, aby otrzymać nowy cykl Eulera $C$, który nie zawiera ścieżki $\pi$:

$$C = \langle v_0, \ldots, v_{i-1}, v_i, x,\ \beta\ , v_0,\ \alpha\ , v_i, v_{i+1}, \ldots \rangle$$

■

Rozważmy następujący algorytm wyznaczania cyklu Eulera w grafie zorientowanym (dokładny opis oraz dowód poprawności tego algorytmu można odnaleźć w [45]). Poniższy algorytm będzie pomocny w konstrukcji algorytmu dla problemu *EulFS*.




Rysunek 3.5: Cykl $C'$ z lematu 3.4.4.

**Algorytm 3.4.1**: FINDEULERCYCLE

**Dane**: eulerowski graf skierowany $G = (V, E)$ oraz drzewo rozpinające $T$ grafu $G$ o ustalonym korzeniu w $r \in V$ (krawędzie drzewa są skierowane w kierunku korzenia)

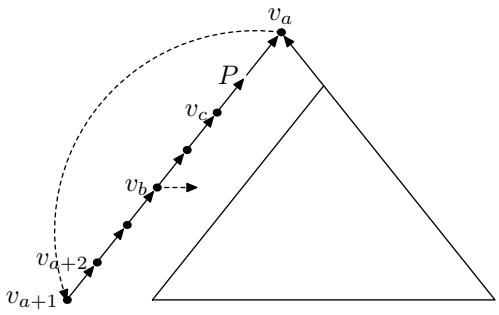
```
1  C = ε;   v = r
2  while |C| < |E| do
3      if istnieje krawędź (v, x) ∉ C i taka, że x ≠ parent(v) then
4          C = C + (v, x)
5          v = x
6      else
7          C = C + (v, parent(v))
8          v = parent(v)
9      end
10 end
11 return C
```

Rysunek 3.6: Algorytm wyznaczania Cyklu Eulera w grafie skierowanym.

**Lemat 3.4.5** *Niech $G = (V, E)$ będzie skierowanym grafem eulerowskim, a $\pi$ zabronioną ścieżką $\pi = \langle v_0, \ldots, v_k \rangle$. Jeśli istnieją indeksy $a, b, c$ takie, że $0 \leq a < b < c \leq k$, $outdeg(v_b) \geq 2$, oraz istnieje ścieżka $P$ z wierzchołka $v_c$ do $v_a$, dla której $(V(\pi) \cap V(P)) - \{v_a, v_c\} = \emptyset$, to graf $G$ posiada cykl Eulera $C$, który nie zawiera ścieżki $\pi$.*

**Dowód:** Można zauważyć, że w grafie $G$ istnieje drzewo rozpinające o korzeniu $v_a$, które zawiera ścieżkę $\langle v_{a+1}, \ldots, v_c, P \rangle$. Jeśli cykl Eulera wyznaczymy algorytmem przedstawionym na rysunku 3.6, to taki cykl nie będzie zawierał ścieżki $\pi$.

■




Rysunek 3.7: Drzewo rozpinające $T$ z lematu 3.4.5.

Zdefiniujemy teraz funkcję $low(v)$, która pozwoli na efektywne testowanie, czy warunki lematu 3.4.5 są spełnione. Dla zabronionej ścieżki $\pi = \langle v_0, \ldots, v_k \rangle$,

$$low(v) = \min \{i : \text{istnieje ścieżka } P \text{ z } v \text{ do } v_i, \text{ taka, że } V(P) \cap V(\pi) \subseteq \{v, v_i\}\}$$

Dla $v = v_0$ wartość funkcji $low(v)$ pozostaje niezdefiniowana. Dla $v \in \mathbf{V}(G) - \{v_0\}$ wartości funkcji $low(v)$ można wyznaczyć w czasie liniowym ze względu na rozmiar grafu i ścieżki. Funkcję $low(v)$ możemy wyznaczyć analizując silne spójne składowe grafu $G'$ otrzymanego z $G$ przez zastąpienie każdego wierzchołka $v_i \in P$ przez dwa nowe wierzchołki $v_i^{in}$ i $v_i^{out}$, przy czym wszystkie krawędzie, które wcześniej wchodziły do wierzchołka $v_i$ łączymy z $v_i^{in}$, a krawędzie, które wychodziły z $v_i$, zamieniamy na wychodzące z $v_i^{out}$.

Opiszemy teraz algorytm dla problemu *EulFS* w przypadku, gdy $\pi$ jest ścieżką bez powtórzeń wierzchołków. Później pokażemy, w jaki sposób rozszerzyć to rozwiązanie na dowolną ścieżkę.

Jeśli algorytm zwraca odpowiedź negatywną, to graf $G$ ma postać przedstawioną na rysunku 3.9. Zauważmy, że w takim przypadku każdy cykl Eulera musi zawierać ścieżkę $\pi$.

Wróćmy do ogólnego przypadku, gdy $\pi$ nie musi być ścieżką prostą. Niech $cycles(\pi)$ będzie liczbą różnych cykli elementarnych w grafie rozpiętym na krawędziach ścieżki $\pi$.

**Lemat 3.4.6** *Niech $G$ będzie skierowanym grafem eulerowskim i niech $\pi$ będzie ścieżką posiadającą następujące własności:*

- $\pi = \langle \ldots, x, \alpha, x, \ldots \rangle$ *oraz* $\langle x, \alpha, x \rangle$ *jest cyklem elementarnym*
- *każdy wierzchołek $v \in \alpha$ występuje w $\pi$ dokładnie raz.*

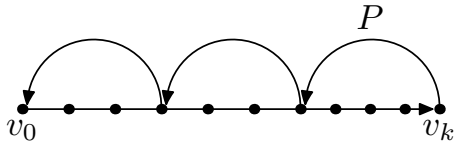
*Wówczas istnieją graf $G'$ oraz ścieżka $\pi'$ i takie, że:*

**Algorytm 3.4.2**: SOLVEEULFS-SIMPLEPATH

**Dane**: prosty, skierowany graf eulerowski $G = (V, E)$ oraz zabroniona ścieżka $\pi = \langle v_0, \ldots, v_k \rangle$, $v_i \neq v_j$ dla $i \neq j$.

**if** *istnieje indeks* $i$, $0 < i < k$, *taki, że* $outdeg(v_i) > 2$ **then**
    **return True** (patrz lemat 3.4.3)
**else**
    { dla każdego $i$, $0 < i < k$), $outdeg(v_i) \leq 2$ }
    wyznacz wartości funkcji $low(v)$ dla wszystkich $v \in V - \{v_0\}$
    $i = k$
    **while** $i > 0$ **do**
        $i' = low(v_i)$
        **if** *dla każdego* $j$, $i' < j < i$, $outdeg(v_j) = 1$ **then**
            $i = i'$
        **else**
            { istnieje $j$, $i' < j < i$, $outdeg(v_j) = 2$, zatem spełnione są założenia lematu 3.4.5 }
            **return True**
    **end**
    { graf ma postać z rysunku 3.9 }
    **return False**

Rysunek 3.8: Algorytm dla problemu *EulFS* gdy $\pi$ jest ścieżką bez powtórzeń wierzchołków.



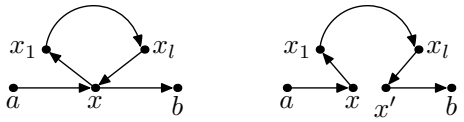

Rysunek 3.9: Ścieżka z $v_k$ do $v_0$.

- $cycles(\pi') < cycles(\pi)$,
- $(G, \pi)$ *posiada cykl Eulera, który nie zawiera* $\pi$ *wtedy i tylko wtedy, gdy* $(G', \pi')$ *posiada cykl Eulera nie zawierający ścieżki* $\pi'$.

**Dowód:** Niech $G = (V, E)$ i $\pi = \langle \ldots, a, x, x_1, \ldots, x_l, x, b, \ldots \rangle$, gdzie $\alpha = \langle x_1, \ldots, x_l \rangle$. Graf $G' = (V', E')$ definiujemy następująco:

- $V' = V \cup \{x'\}$,
- $E' = E - \{(x_l, x), (x, b)\} \cup \{(x_l, x'), (x', b)\}$

Za $\pi'$ bierzemy ścieżkę $\langle \ldots, a, x, x_1, \ldots, x_l, x', b, \ldots \rangle$. Z opisanej konstrukcji wprost wynika teza lematu. ■




Rysunek 3.10: Konstrukcja grafu $G'$ w lemacie 3.4.6.

Wykorzystując schemat postępowania opisany w dowodzie lematu możemy usunąć wszystkie cykle z grafu rozpiętego na krawędziach ścieżki $\pi$. Jeśli po drodze uzyskamy cykl, w którym każdy wierzchołek na stopień $outdeg(v) = 1$, to stosujemy lemat 3.4.6. W przeciwnym przypadku z lematu 3.4.4 wynika, że możemy zatrzymać dalsze obliczenia i odpowiedzieć pozytywnie.

Podsumowując, każdą ścieżkę $\pi$ możemy zamienić przez równoważną ścieżkę $\pi^*$ bez powtórzeń wierzchołków lub otrzymać cykl Eulera $C$, który nie zawiera ścieżki $\pi$ (z lematu 3.4.4).

Z naszych dotychczasowych rozważań otrzymujemy kompletny algorytm rozwiązujący problem *EulFS* dla grafów prostych i działający w czasie liniowym.

**Algorytm 3.4.3**: SOLVEEULFS

**Dane**: prosty, skierowany graf eulerowski $G = (V, E)$, zabroniona ścieżka $\pi = \langle v_0, \ldots, v_k \rangle$

```
1  jeśli istnieje wierzchołek v_i, 0 < i < k, dla którego outdeg(v_i) > 2, to
   return True /* lemat 3.4.3) */
2  π* = π; G* = G
3  while ścieżka π* zawiera cykl elementarny C do
4      niech π* = ⟨v_0, ..., v'_k⟩, C = v_a, ..., v_b, a < b, oraz v_a = v_b
5      if istnieje indeks i, a < i < b, outdeg(v_i) > 1 then
6          return True /* lemat 3.4.4 */
7      else
8          usuń cykl C ze ścieżki π* tak jak w dowodzie lematu 3.4.6,
           otrzymując nowy graf G* i ścieżkę π*
9      end
10 end
11 return SOLVEEULFS-SIMPLEPATH(G*, π*)
```

Rysunek 3.11: Algorytm dla problemu *EulFS* dla grafów prostych.

Niestety problem *EulFS* w przypadku multigrafów jest trudniejszy, obecnie nie jest znany algorytm o złożoności wielomianowej dla tego przypadku.

# Bibliografia


[1] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computer and System Sciences*, 21(1):136–153, 1980.

[2] D. A. Bader, B. M. E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. In F. K. H. A. Dehne, J.-R. Sack, and R. Tamassia, editors, *WADS*, volume 2125 of *Lecture Notes in Computer Science*, pages 365–376. Springer, 2001.

[3] D. A. Bader, B. M. E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8(5):483–491, 2001.

[4] V. Bafna, S. Muthukrishnan, and R. Ravi. Computing similarity between RNA strings. In *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 1–16, 1995.

[5] V. Bafna, H. Tang, and S. Zhang. Consensus folding of unaligned rna sequences revisited. *Proceedings of The 9th International Conference on Computational Molecular Biology (RECOMB)*, 3500:172–187, 2005.

[6] S. Bereg and B. Zhu. RNA multiple structural alignment with longest common subsequences. *Computing and Combinatorics (COCOON)*, 3595:32–41, 2005.

[7] A. Caprara. Sorting by reversals is difficult. In *Proceedings of the First International Conference on Computational Molecular Biology*, pages 75–83. ACM Press, 1997.

[8] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(4):302–315, 2005.

[9] D. A. Christie and R. W. Irving. Sorting strings by reversals and by transpositions. *SIAM Journal on Discrete Mathematics*, 14(2):193–206, 2001.

[10] M. Chrobak, P. Kolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. In *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 84–95, 2004.

[11] M. Chrobak, P. Kolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. *ACM Transactions on Algorithms*, 1(2):350–366, 2005.

[12] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms (Second Edition)*. MIT Press, 2001.

[13] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA)*, pages 667–676, 2002.

[14] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Transactions on Algorithms*, 3(1), 2007.

[15] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.

[16] E. Davydov and S. Batzoglou. A computational model for RNA multiple structural alignment. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 3109 of *Lecture Notes in Computer Science*, pages 254–269. Springer-Verlag, 2004.

[17] E. Davydov and S. Batzoglou. A computational model for rna multiple structural alignment. *Theoretical Computer Science*, 368(3):205–216, 2006.

[18] M. Farach. Optimal suffix tree construction with large alphabets. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 137–143, 1997.

[19] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 246–251, 1983.

[20] J. Gallant, D. Maier, and J. Storer. On finding minimal length superstrings. *Journal of Computer and System Sciences*, 20:50–58, 1980.

[21] A. Goldstein, P. Kolman, and J. Zheng. Minimum Common String Partition Problem: Hardness and Approximations. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC)*, volume 3341 of *Lecture Notes in Computer Science*, pages 484–495, 2004.

[22] A. Goldstein, P. Kolman, and J. Zheng. Minimum Common String Partition Problem: Hardness and Approximations. *The Electronic Journal of Combinatorics*, 12(1), Sept. 2005.

[23] D. Gusfield, R. Karp, L. Wang, and P. Stelling. Graph traversals, genes and matroids: and efficient case of the travelling salesman problem. *Discrete Applied Mathematics*, 88:167–180, 1998.

[24] S. Hannenhalli, W. Feldman, H. Lewis, S. Skiena, and P. Pevzner. Positional sequencing by hybridization. *Computer Applications in the Biosciences*, 12:19–24, 1996.

[25] S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1–27, 1999.

[26] I. Holmes and G. M. Rubin. Pairwise RNA structure comparison with stochastic context-free grammars. In *Pacific Symposium on Biocomputing*, pages 163–174, 2002.

[27] P. Kolman. Approximating reversal distance for strings with bounded number of duplicates. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 3618 of *Lecture Notes in Computer Science*, pages 580–590, 2005.

[28] P. Kolman and T. Waleń. Reversal distance for strings with duplicates: Linear time approximation using hitting set. In T. Erlebach and C. Kaklamanis, editors, *Workshop on Approximation and Online Algorithms (WAOA)*, volume 4368 of *Lecture Notes in Computer Science*, pages 279–289. Springer, 2006.

[29] P. Kolman and T. Waleń. Approximating reversal distance for strings with bounded number of duplicates. *Discrete Applied Mathematics*, 155(3):327–336, 2007.

[30] P. Kolman and T. Waleń. Reversal distance for strings with duplicates: Linear time approximation using hitting set. *Electronic Journal of Combinatorics*, 14(1), 2007.

[31] M. Kubica, R. Rizzi, S. Vialette, and T. Waleń. Approximation of rna multiple structural alignment. In M. Lewenstein and G. Valiente, editors, *Combinatorial Pattern Matching (CPM)*, volume 4009 of *Lecture Notes in Computer Science*, pages 211–222. Springer, 2006.

[32] G. Lin, Z.-Z. Chen, T. Jiang, and J. Wen. The longest common subsequence problem for sequences with nested arc annotations. *Journal of Computer and System Sciences*, 65(3):465–480, 2002. Special issue on computational biology.

[33] J. Liu, J. T. Wang, J. Hu, and B. Tian. A method for aligning RNA secondary structures and its application to RNA motif detection. *BMC Bioinformatics*, 6(89), 2005.

[34] J. Maxova and J. Nesetril. Complexity of compatible decompositions of eulerian graphs and their transformations. *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, pages 711–722, 2002.

[35] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of ACM*, 23(2):262–272, 1976.

[36] M. Middendorf. More on the complexity of common superstring and supersequence problems. *Theoretical Computer Science*, 125(2):205–228, 1994.

[37] P. Pevzner. $l$–tuple dna sequencing: Computer analysis. *Journal of Biomolecular Structure and Dynamics*, 7:63–73, 1989.

[38] P. Pevzner, H. Tang, and M. Waterman. A new approach to fragment assembly in dna sequencing. *Proceedings of the 5th International Conference on Computational Molecular Biology (RECOMB)*, 2001.

[39] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *Symposium on the Theory of Computing (STOC)*, pages 475–484, 1997.

[40] E. Tannier, A. Bergeron, and M.-F. Sagot. Advances on sorting by reversals. *Discrete Applied Mathematics*, 155(6-7):881–888, 2007.

[41] E. Ukkonen. Constructing suffix trees on-line in linear time. In J. van Leeuwen, editor, *IFIP Congress (1)*, volume A-12 of *IFIP Transactions*, pages 484–492. North-Holland, 1992.

[42] S. Vialette. On the computational complexity of 2-interval pattern matching. *Theoretical Computer Science*, 312(2-3):223–249, 2004.

[43] M. Waterman. *Introduction to computational biology - Maps, sequences and genomes*. Chapman and Hall, London, 1995.

[44] P. Weiner. Linear pattern matching algorithms. In *14th IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[45] D. B. West. *Introduction to Graph Theory (2nd Edition)*. Prentice Hall, 2000.