Naprawianie automatów skończonych z brakującymi stanami i krawędziami

Repairing Finite Automata
with Missing Components

Julia Cygan, Patryk Flama

Praca inżynierska
Promotor: dr hab. Jakub Michaliszyn

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

18 stycznia 2026

Julia Cygan

. . . . . .
(adres zameldowania)

. . . . . .
(adres korespondencyjny)

PESEL: . . . . . .

e-mail: . . . . . .

Wydział Matematyki i Informatyki
stacjonarne studia I stopnia
kierunek: informatyka
nr albumu: . . . . . .

Patryk Flama

. . . . . .
(adres zameldowania)

. . . . . .
(adres korespondencyjny)

PESEL: . . . . . .

e-mail: . . . . . .

Wydział Matematyki i Informatyki
stacjonarne studia I stopnia
kierunek: informatyka
nr albumu: . . . . . .

Oświadczenie o autorskim wykonaniu pracy dyplomowej

Niniejszym oświadczamy, że złożoną do oceny pracę zatytułowaną *Naprawianie automatów skończonych z brakującymi stanami i krawędziami* wykonaliśmy samodzielnie pod kierunkiem promotora, dr Jakuba Michaliszyna. Oświadczamy, że powyższe dane są zgodne ze stanem faktycznym i znane nam są przepisy ustawy z dn. 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity: Dz. U. z 2006 r. nr 90, poz. 637, z późniejszymi zmianami) oraz że treść pracy dyplomowej przedstawionej do obrony, zawarta na przekazanym nośniku elektronicznym, jest identyczna z jej wersją drukowaną.

Wrocław, 18 stycznia 2026

. . . . . .
(czytelny podpis)

. . . . . .
(czytelny podpis)

Abstract

Niniejsza praca dotyczy zagadnienia naprawy uszkodzonego deterministycznego automatu skończonego na podstawie próbek pozytywnych i negatywnych. Rozważany automat może zawierać brakujące krawędzie lub stany, co uniemożliwia jego poprawne działanie. W pracy analizowana jest złożoność obliczeniowa problemu, w tym jego przynależność do klasy NP oraz własności parametryzowane w kontekście algorytmów FPT i hierarchii W. Zaproponowano algorytm rozwiązujący rozważany problem, przeprowadzono jego analizę teoretyczną oraz przedstawiono implementację wraz z omówieniem optymalizacji i heurystyk wpływających na czas działania.

This thesis addresses the problem of repairing a damaged deterministic finite automaton using positive and negative samples. The considered automaton may contain missing transitions or states, which prevents it from functioning correctly. The work analyzes the computational complexity of the problem, including its membership in the class NP and its parameterized properties in the context of FPT algorithms and the W-hierarchy. An algorithm for solving the problem is proposed, followed by its theoretical analysis, as well as an implementation accompanied by a discussion of optimizations and heuristics affecting the running time.

Spis treści

Rozdział 1

Wstęp

Teoria automatów jest dziedziną informatyki teoretycznej, zajmująca się głównie badaniem abstrakcyjnych maszyn wykorzystywanych w celu modelowania obliczeń. Automat to model, który przetwarza dane wejściowe poprzez wykonywanie przejść pomiędzy kolejnymi stanami zgodnie ze zdefiniowanym sposobem reagowania na poszczególne symbole. Automat najczęściej definiowany jest jako graf z oznaczonymi krawędziami i określonymi wierzchołkami początku i końca. Jednym z najważniejszych i najczęściej badanych modeli są automaty skończone, które charakteryzują się skończonym zbiorem stanów.

Jedną z najważniejszych klas automatów skończonych są deterministyczne automaty skończone (ang. *Deterministic Finite Automata*, DFA). W modelu tym dla każdego stanu oraz symbolu alfabetu wejściowego zdefiniowane jest dokładnie jedno przejście do kolejnego stanu. Dzięki tej własności działanie automatu jest jednoznaczne i w pełni przewidywalne, co znacząco upraszcza jego analizę oraz implementację.

W kontekście uczenia automatów wyróżnia się dwa główne podejścia: uczenie aktywne i uczenie pasywne. Uczenie pasywne polega na konstruowaniu modelu automatu wyłącznie na podstawie gotowego zbioru przykładów wejściowych, bez możliwości zadawania dodatkowych pytań czy testowania hipotez w trakcie procesu uczenia. W praktyce oznacza to, że algorytm otrzymuje skończony zbiór słów oznaczonych jako akceptowane lub odrzucane i na tej podstawie próbuje odtworzyć strukturę automatu, który najlepiej odzwierciedla obserwowane zachowanie systemu. Podejście pasywne jest szczególnie użyteczne w sytuacjach, w których brak jest dostępu do „czarnej skrzynki" systemu lub gdy interaktywne testowanie wszystkich możliwych sekwencji wejściowych jest niemożliwe lub kosztowne. Pomimo swojej prostoty, uczenie pasywne wiąże się z szeregiem trudności teoretycznych i praktycznych, w tym ograniczeniem informacji wynikającym z niekompletności danych, możliwością istnienia wielu automatów zgodnych z tym samym zbiorem próbek oraz problemem minimalizacji otrzymanego modelu.

Jednym z fundamentalnych wyników w teorii pasywnego uczenia języków formalnych było wykazanie, że klasa języków regularnych nie jest identyfikowalna w granicy wyłącznie na podstawie pozytywnych przykładów [1]. Istotnie ogranicza to możliwości pasywnego uczenia automatów skończonych bez dodatkowych założeń. Wynik ten miał istotny wpływ na dalszy rozwój wnioskowania gramatyk, wskazując na konieczność wykorzystywania zarówno przykładów pozytywnych, jak i negatywnych, bądź wprowadzania dodatkowych ograniczeń na strukturę danych uczących lub klasę rozważanych automatów.

W ostatnich latach problem pasywnego uczenia deterministycznych automatów skończonych pozostaje przedmiotem intensywnych badań. W jednej z nowszych prac autorzy koncentrują się na formalnej analizie problemu DFA-consistency, czyli określenia,

czy istnieje minimalny deterministyczny automat skończony, który akceptuje wszystkie pozytywne przykłady i odrzuca wszystkie negatywne przykłady dostarczone w zbiorze uczącym. Badając złożoność obliczeniową tego problemu oraz warianty wynikające z różnych ograniczeń na alfabet i strukturę danych uczących. Wykazano, że problem DFA-consistency jest NP-zupełny, nawet w przypadku alfabetów binarnych, co oznacza, że w ogólności nie istnieje znany algorytm wielomianowy rozwiązujący go dla wszystkich instancji [2].

Inne podejście prezentują prace, w których rekonstrukcja deterministycznych automatów skończonych realizowana jest poprzez redukcję problemu uczenia do problemów spełnialności logicznej (SAT). Przykładem takiego rozwiązania jest narzędzie DFAMiner [3], które konstruuje automat pośredni w postaci trójwartościowego automatu skończonego (3DFA), zawierającego stany akceptujące, odrzucające oraz stany typu nieistotne, umożliwiające dokładne rozpoznanie dostarczonych przykładów uczących. Następnie automat ten jest minimalizowany poprzez redukcję do problemu SAT, co pozwala na uzyskanie minimalnego automatu separującego, czyli deterministycznego automatu skończonego o najmniejszej możliwej liczbie stanów, który akceptuje wszystkie przykłady pozytywne i jednocześnie odrzuca wszystkie przykłady negatywne. Tego rodzaju automat nie musi w pełni określać języka docelowego, lecz jedynie rozdzielać (separować) dostarczone zbiory próbek. Przeprowadzone badania empiryczne wskazują, że tego typu podejścia mogą znacząco przewyższać klasyczne metody uczenia pasywnego pod względem efektywności obliczeniowej. Jednocześnie skuteczność metod opartych na redukcji do SAT pozostaje silnie uzależniona od kompletności oraz spójności danych uczących, a w przypadku próbek niepełnych lub sprzecznych liczba potencjalnych modeli rośnie wykładniczo, co istotnie komplikuje proces uczenia.

Pomimo znacznego postępu w dziedzinie pasywnego uczenia DFA, większość istniejących metod zakłada, że automat uczony jest konstruowany od podstaw na podstawie zbioru przykładów. W praktycznych zastosowaniach często spotyka się jednak sytuacje, w których dostępny jest częściowo zdefiniowany automat, zawierający brakujące stany, niepełne przejścia lub fragmentaryczną wiedzę o strukturze systemu. Tego rodzaju przypadki pojawiają się m.in. w inżynierii odwrotnej, analizie dziedziczonych systemów, rekonstrukcji protokołów komunikacyjnych oraz w procesach naprawy modeli formalnych.

Celem niniejszej pracy jest analiza problemu naprawy brakujących deterministycznych automatów skończonych na podstawie skończonego zbioru przykładów pozytywnych i negatywnych. Przez brakujący deterministyczny automat skończony rozumiany jest automat, w którym zbiór stanów oraz część przejść są określone poprawnie, natomiast pozostałe przejścia nie zostały zdefiniowane. Naprawa automatu polega na uzupełnieniu brakujących przejść w taki sposób, aby otrzymany automat był deterministyczny oraz zgodny z dostarczonym zbiorem przykładów. W rozważanym problemie zakłada się, że automat wejściowy jest dany z góry, a zbiór przykładów pozytywnych i negatywnych jest niesprzeczny, tzn. istnieje co najmniej jeden deterministyczny automat skończony, który jest zgodny zarówno z istniejącą strukturą automatu, jak i z dostarczonymi danymi uczącymi.

W pracy skoncentrowano się na teoretycznej analizie złożoności obliczeniowej problemu naprawy brakujących deterministycznych automatów skończonych. W szczególności wykazane zostanie, że rozważany problem jest NP-zupełny, a ponadto omówiona zostanie jego trudność w sensie klas parametryzowanych W[1] oraz W[2]. Oprócz wyników teoretycznych zaprezentowany zostanie algorytm, rozwiązujący ten problem w czasie lepszym niż podejście brute force, wykorzystujący skoki przez zdefiniowane krawędzie.

Rozdział 2

Teoria

2.1 Definicja problemu, framework

Definicja 2.1.1 - Deterministyczny automat skończony (DFA)
Deterministyczny automat skończony (DFA) to szóstka uporządkowana $(Q, \Sigma, \delta, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R})$, gdzie:

- Q to skończony zbiór stanów,
- Σ to skończony alfabet wejściowy,
- $\delta : Q \times \Sigma \to Q$ to funkcja przejścia,
- $q_\lambda \in Q$ to stan początkowy,
- $\mathbb{F}_\mathbb{A} \subseteq Q$ to zbiór stanów akceptujących,
- $\mathbb{F}_\mathbb{R} \subseteq Q$ to zbiór stanów odrzucających.

2.1.1 Definicja problemu

Definicja 2.1.2 - Problem naprawienia częściowego DFA
Wejście: częściowy automat deterministyczny $A = (Q, \Sigma, \delta, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R})$, w którym:

- niektóre krawędzie w automacie mogły zostać usunięte, więc dla niektórych par $(q, a) \in Q \times \Sigma$ funkcja δ nie jest określona,
- niektóre stany $q \in Q$ mogły zostać usunięte z automatu (brakujące stany), więc nie należą one ani do $\mathbb{F}_\mathbb{A}$, ani do $\mathbb{F}_\mathbb{R}$,

oraz zbiory próbek $S^+ \subseteq \Sigma^*$ (słowa akceptowane) i $S^- \subseteq \Sigma^*$ (słowa odrzucane).

Wyjście: odpowiedź, czy istnieje uzupełnienie brakujących przejść, klasyfikacji stanów i ewentualne dodanie stanów tak, aby otrzymany automat był deterministyczny, posiadał najmniejszą możliwą liczbę stanów oraz akceptował wszystkie słowa z S^+ i odrzucał wszystkie słowa z S^-. W przypadku istnienia, należy podać takie uzupełnienie.

2.1.2 Trywialność przypadku brakujących stanów

W rozważanej wersji problemu dopuszczamy dodawanie brakujących stanów. Jest to jednak przypadek trywialny, bo możemy ograniczyć maksymalną liczbę stanów w automacie do liczby stanów w drzewie prefiksowym zbudowanym z próbek - jeżeli automat jest naprawialny, to będzie to automat rozwiązujący problem. Rozmiar takiego drzewa możemy ograniczyć przez $N = |S^+ \cup S^-| \cdot \max_{w \in S^+ \cup S^-} |w|$. Możemy więc dla każdej liczby stanów n w zakresie $[1, N]$ sprawdzać, czy istnieje naprawa automatu z dokładnie n stanami; od pewnej wartości n odpowiedź staje się pozytywna, co pozwala użyć wyszukiwania binarnego bez istotnej zmiany (i tak wykładniczej) złożoności.

Dlatego w dalszej części pracy zakładamy, że liczba stanów jest ustalona i nie rozważamy dodawania nowych.

2.1.3 Definicja problemu (uproszczona)

Definicja 2.1.3 - Problem naprawienia częściowego DFA (uproszczony)
Wejście: częściowy automat deterministyczny $A = (Q, \Sigma, \delta, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R})$, w którym dla pewnych par $(q, a) \in Q \times \Sigma$ funkcja δ nie jest określona, oraz zbiory próbek $S^+ \subseteq \Sigma^*$ i $S^- \subseteq \Sigma^*$. Liczba stanów $|Q|$ jest ustalona.

Wyjście: odpowiedź, czy istnieje uzupełnienie brakujących przejść i klasyfikacji stanów tak, aby otrzymany automat był deterministyczny, akceptował wszystkie słowa z S^+ i odrzucał wszystkie słowa z S^-. W przypadku istnienia należy podać takie uzupełnienie.

2.2 NP-zupełność

2.2.1 Przynależność do NP

Problem naprawienia częściowego DFA należy do klasy NP, ponieważ dla danej instancji problemu możemy w czasie wielomianowym zweryfikować poprawność podanego rozwiązania.
Weryfikacja polega na sprawdzeniu, czy uzupełniony automat jest deterministyczny oraz czy akceptuje i odrzuca odpowiednie próbki. Sprawdzenie deterministyczności automatu wymaga przejrzenia wszystkich stanów i liter alfabetu, co zajmuje czas $O(|Q| \cdot |\Sigma|)$. Sprawdzenie klasyfikacji próbek wymaga przejścia przez każdą próbkę i symulacji jej działania na automacie, co zajmuje czas $O((|S^+| + |S^-|) \cdot M)$, gdzie M to maksymalna długość próbki. Ponieważ oba te kroki można wykonać w czasie wielomianowym względem rozmiaru wejścia, problem naprawienia częściowego DFA należy do klasy NP.

2.2.2 NP-trudność

Fakt 2.2.1
Poniższy problem *najmniejszego zgodnego automatu* jest NP-zupełny [4]:

Otrzymujemy liczbę natrualną $n \in \mathbb{N}$ oraz dwa zbiory słów nad alfabetem Σ: zbiór słów akceptowanych $S^+ \subseteq \Sigma^*$ oraz zbiór słów odrzucanych $S^- \subseteq \Sigma^*$. Należy odpowiedzieć na pytanie, czy istnieje deterministyczny automat skończony (DFA) $\mathcal{A}$, z co najwyżej n stanami, taki że wszystkie słowa z S^+ są akceptowane przez $\mathcal{A}$, a wszystkie słowa z S^- są odrzucane przez $\mathcal{A}$.

Możemy przeprowadzić redukcję z problemu *najmniejszego zgodnego automatu* do naszego problemu *naprawienia częściowego DFA*.
Weźmy instancję problemu *najmniejszego zgodnego automatu* - liczbę naturalną n oraz zbiory słów S^+ oraz S^-. Stwórzmy częściowy automat DFA $A = (Q, \Sigma, \delta, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R})$, gdzie:

$$\forall q \in Q \; \forall a \in \Sigma \; \delta(q, a) \text{ jest nieokreślone}$$

natomiast zbiory próbek są takie same jak w oryginalnym problemie.
Wtedy odpowiedź na problem *naprawienia częściowego DFA* dla automatu A oraz próbek S^+ i S^- jest pozytywna wtedy i tylko wtedy gdy odpowiedź na problem *najmniejszego zgodnego automatu* dla liczby n oraz próbek S^+ i S^- jest pozytywna.

2.3 FPT

2.3.1 W[1]-trudność

Definicja 2.3.1 - Problem k-klik
Weźmy graf nieskierowany $G = (V, E)$ oraz liczbę całkowita k. Problem k-klik polega na znalezieniu w grafie G kliki o rozmiarze co najmniej k, czyli podzbioru $V' \subseteq V$ takiego, że $|V'| \geq k$ oraz dla każdej pary wierzchołków $u, v \in V'$ zachodzi $(u, v) \in E$.
Problem k-klik należy do klasy W[1]-zupełnych problemów [5].

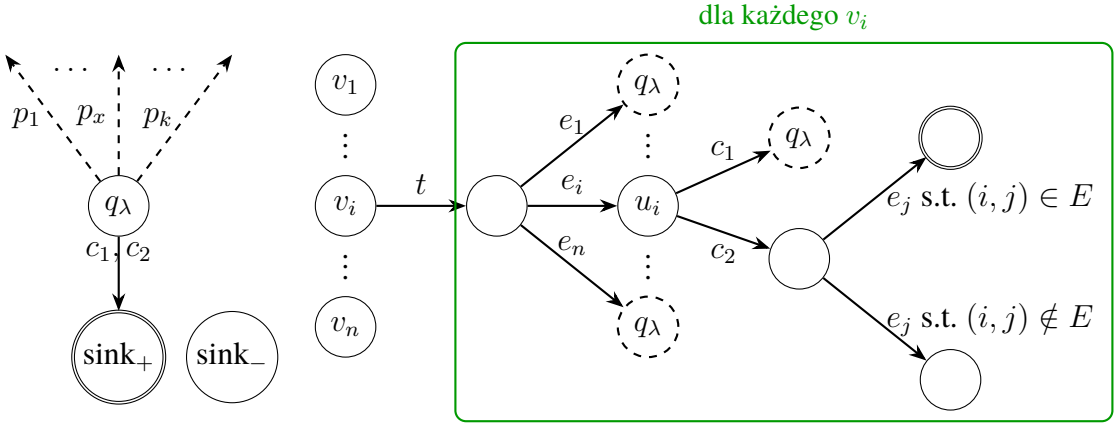
Pokażemy, jak dla dowolnego wejścia do problemu k-klik skonstruować automat oraz próbki, będące wejściem do problemu naprawienia częściowego DFA, tak aby rozwiązanie problemu naprawienia częściowego DFA istniało wtedy i tylko wtedy, gdy w grafie istnieje klika o rozmżemy, jak dla dowolnego wejścia do problemu k-klik skonstruować automat oraz próbki, będące wejściem do problemu naprawienia częściowego DFA, tak aby rozwiązanie problemu naprawienia częściowego DFA istniało wtedy i tylko wtedy, gdy w grafie istnieje klika o rozmiarze co najmniej k. Dodatkowo pokażemy jak z rozwiązanego problemu naprawienia częściowego DFA wyprowadzić rozwiązanie problemu k-klik.

Dla $|\Sigma| = O(n)$

Na rysunku 2.1 przedstawiona jest konstrukcja automatu dla redukcji z k-klik, korzystająca z alfabetu o rozmiarze zależnym od liczby wierzchołków grafu.
Każdy stan q_λ odpowiada temu samemu stanowi początkowemu automatu. Przejścia p_i oznaczone przerywaną linią odpowiadają brakującym przejściom w automacie. Każde przejście, które nie jest zaznaczone i nie wychodzi ze stanu $sink_+$ lub $sink_-$, prowadzi do stanu odrzucającego sink_-. Niezaznaczone przejścia wychodzące ze stanów sink_+ i sink_- prowadzą do tych samych stanów.
Część automatu zaznaczona ramką jest powielana dla każdego stanu $v_.$.

Idea konstrukcji:

Chcemy aby krawędzie $p_.$ prowadziły do stanów $v_.$. Jeżeli krawędź p_x prowadzi do stanu v_i, interpretujemy to jako wybranie wierzchołka V_i jako x-ty wierzchołek w klice.



Rysunek 2.1. Konstrukcja automatu dla redukcji z k-klik, korzystająca z alfabetu o rozmiarze zależnym od liczby wierzchołków grafu

$$\forall x \in \{1..k\}\ \exists i \in \{1..n\}\ \delta(q_\lambda, p_x) = v_i \tag{2.1}$$

Do tej samej kliki nie możemy wybrać tego samego wierzchołka wielokrotnie, więc musimy zagwarantować, że dla różnych przejść p_x i p_y wybieramy różne stany v_i i v_j.

$$\forall (x, y \in \{1..k\} \wedge x \neq y)\ \forall i, j \in \{1..n\}\ (\delta(q_\lambda, p_x) = v_i \wedge \delta(q_\lambda, p_y) = v_j) \implies i \neq j \tag{2.2}$$

Dodatkowo musimy zagwarantować, że wybrane wierzchołki tworzą klikę, czyli że każdy wybrany wierzchołek jest połączony z każdym innym wybranym wierzchołkiem.

$$\forall (x, y \in \{1..k\} \wedge x \neq y)\ \forall i, j \in \{1..n\}\ (\delta(q_\lambda, p_x) = v_i \wedge \delta(q_\lambda, p_y) = v_j) \implies (v_i, v_j) \in E \tag{2.3}$$

Gwarancja prowadzenia brakujących krawędzi p_x do wierzchołków v_i

Zagwarantowanie tego faktu opiera się na prostej obserwacji: jedyne krawędzie t, które nie prowadzą do stanu odrzucającego, znajdują się za wierzchołkami v_i. Oznacza to, że aby próbka mogła zostać zaakceptowana, musimy przejść przez krawędź p_x do któregoś ze stanów v_i, a następnie przez krawędź t do stanu akceptującego. W przeciwnym wypadku, w trakcie przechodzenia próbką po automacie, trafimy do stanu sink$_-$, więc próbka zostanie odrzucona.
W takim przypadku, aby zagwarantować przejście krawędzią p_x do któregoś ze stanów v_i, dodajemy w każdej próbce literę t zaraz po literze p_x.

Jedynym wyjątkiem jest stan sink$_+$. Wychodząca z niego krawędź t musi też prowadzić do sink$_+$ - w przeciwnym wypadku próbki (2.6) zostaną odrzucone gdy p_x, p_y, v_i, v_j takich, że $(i,j) \notin E \implies \exists z \in \{1..n\} v_z = \delta(q_\lambda, p_x) \wedge z \neq i$, ponieważ ze stanu v_z przejdziemy wtedy w automacie krawędzią t a następnie e_i, a $z \neq i \implies \delta(\delta(v_z, t), e_i) = \text{sink}_+$. Ta próbka nie oznacza niepoprawnego przypisana krawędzi (więc powinna zostać zaakceptowana), a wiemy że w naszej próbce występuje jeszcze jedno przejście krawędzią t, więc musi ono prowadzić do stanu sink$_+$.
Możemy ten problem rozwiązać na 2 sposoby. Korzystając z faktu, że w próbce (2.6) są

dokładnie 2 wystąpienia krawędzi t, możemy stworzyć alternatywny sink$_+$ (składający się z 2 stanów), który po pierwszym przejściu krawędzią t prowadzi do drugiego stanu sink$_+$, a po drugim przejściu krawędzią t prowadzi do stanu odrzucajcego sink$_-$.
Alternatywną metodą jest odrzucenie prowadzenie krawędzi p_x do stanu sink$_+$ poprzez dodanie próbek odrzucających, które wymuszają prowadzenie krawędzi p_x do stanu odrzucajcego:

$$\forall x \in \{1..k\}\ [p_x] \in S^- \tag{2.4}$$

Wybór różnych stanów v_i dla różnych przejść p_x

Aby uniemożliwić wybór tego samego v_i dla różnych przejść p_x oraz p_y wystarczy stworzyć próbki postaci:

$$\forall (x, y \in \{1..k\} \wedge x \neq y)\ \forall i \in \{1..n\}\ [p_x\ t\ e_i\ c_1\ p_y\ t\ e_i\ c_1] \in S^+ \tag{2.5}$$

W ten sposób, jeżeli oba przejścia p_x i p_y prowadziłyby do tego samego stanu v_i, to po pierwszym przejściu krawędziami $t\ e_i c_1$ trafilibyśmy do stanu q_λ, a następnie po drugim przejściu krawędziami $t\ e_i$ trafilibyśmy do stanu odrzucajcego q_λ, więc próbka zostałaby odrzucona.
Jeżeli jednak przejścia p_x i p_y prowadzą do różnych stanów v_i i v_j, to pierwsze lub drugie przejście krawędziami $t\ e_i$ doprowadzi do stanu q_λ, a następnie przejście krawędzią c_1 doprowadzi do stanu akceptującego sink$_+$, więc próbka zostanie zaakceptowana.
Gwarancja, że wybrane wierzchołki tworzą klikę Aby zagwarantować, że wybrane wierzchołki tworzą klikę, tworzymy próbki wymuszające sąsiedztwo wybranych wierzchołków:

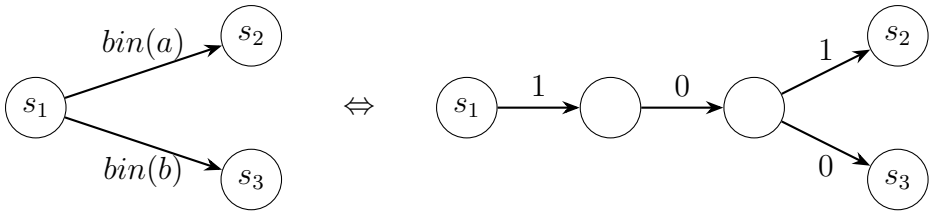
$$\forall (x, y \in \{1..k\} \wedge x \neq y)\ \forall i, j \in \{1..n\}\ [p_x\ t\ e_i\ c_1\ p_y\ t\ e_j\ c_2\ e_i] \in S^+ \tag{2.6}$$

Jeżeli dany stan nie sprawdza wyboru wierzchołków V_i lub V_j w grafie, to któraś z wybranych krawędzi p_x lub p_y nie prowadzi do stanu v_i lub v_j. Wtedy przejście krawędziami $t\ e_i$ lub $t\ e_j$ doprowadzi do stanu akceptującego sink$_+$.
Jeżeli mamy jednak sytuację, w której $\delta(q_\lambda, p_x) = v_i$ oraz $\delta(q_\lambda, p_y) = v_j$, to przejście krawędziami $t\ e_i$ doprowadzi do stanu u_i, skąd krawędzią c_1 trafimy do q_λ. Przejście krawędziami $t\ e_j$ doprowadzi do stanu u_j. Ze stanu u_j przejście krawędziami $c_2\ e_i$ doprowadzi do stanu akceptującego tylko wtedy, gdy w oryginalnym grafie istnieje krawędź $(v_j, v_i) \in E$. W przeciwnym wypadku przejście krawędziami $c_2\ e_i$ doprowadzi do stanu odrzucającego.

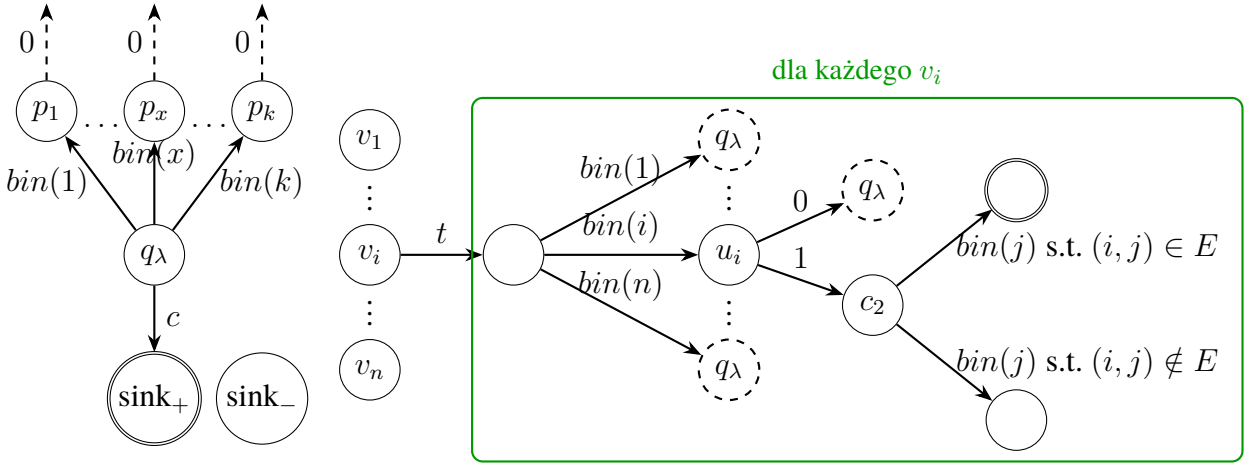
Dla $|\Sigma| = 3$

Aby zredukować liczbę potrzebnych liter z $|\Sigma| = O(n)$ do $|\Sigma| = 3$, możemy zakodować każdą literę z oryginalnego alfabetu jako unikalny ciąg liter z alfabetu 2-literowego (rysunek 2.3). W tym celu możemy użyć kodowania binarnego, gdzie każda litera z oryginalnego alfabetu jest reprezentowana jako ciąg liter '0' i '1' o długości $\lceil \log_2(n) \rceil$, gdzie n to liczba unikalnych liter w oryginalnym alfabecie. Nadal będziemy korzystać z litery testowej t
Ustalimy więc konwencję zapisu, w której dla danego a krawędź z etykietą $bin(a)$ reprezentuje ciąg stanów połączonych krawędziami o etykietach odpowiadających kolejnym bitom w kodzie binarnym $bin(a)$. Jeżeli mamy wiele krawędzi bin wychodzących z tego samego wierzchołka, to ich krawędzie o tych samych etykietach stanowią tą samą krawędź.

Na przykład, jeżeli mamy litery a oraz b reprezentowane przez 101 oraz 100, reprezentują one 4 krawędzie - najpierw ciąg dwóch, gdzie pierwsza ma etykietę 1, a druga 0 - następnie krawędzie 1 oraz 0 wychodzące z ostatniego wierzchołka (tak jak na rysunku 2.2).



Rysunek 2.2. Przykład reprezentacji krawędzi z etykietą bin.



Rysunek 2.3. Konstrukcja automatu dla redukcji z k-klik, korzystająca z alfabetu o rozmiarze 3

Idea działania oraz konstrukcja próbek pozostaje taka sama jak w przypadku alfabetu o rozmiarze $O(n)$, z tą różnicą, że każda krawędź jest teraz reprezentowana jako ciąg krawędzi zgodnie z powyższą konwencją.
W poniższym zapisie $bin(a)$ oznacza ciąg liter reprezentujący literę a w kodzie binarnym.

Gwarancja prowadzenia brakujących krawędzi do wierzchołków v_i

Zagwarantowanie tego faktu opiera się na tej samej obserwacji co w przypadku alfabetu o rozmiarze $O(n)$. Aby próbka mogła zostać zaakceptowana, musimy dwukrotnie przejść przez krawędź t - dodamy więc w każdej próbce literę t po każdym wyborze wierzchołka v_i.
Dodatkowo nadal musimy zapobiec prowadzeniu brakujących krawędzi do stanu sink$_+$. Jak uprzednio, robimy to poprzez liczenie ile razy przeszliśmy krawędzią t, lub poprzez dodanie próbek odrzucających:

$$\forall x \in \{1..k\}\ [bin(x)\ 0] \in S^- \tag{2.7}$$

Wybór różnych stanów v_i dla różnych brakujących krawędzi

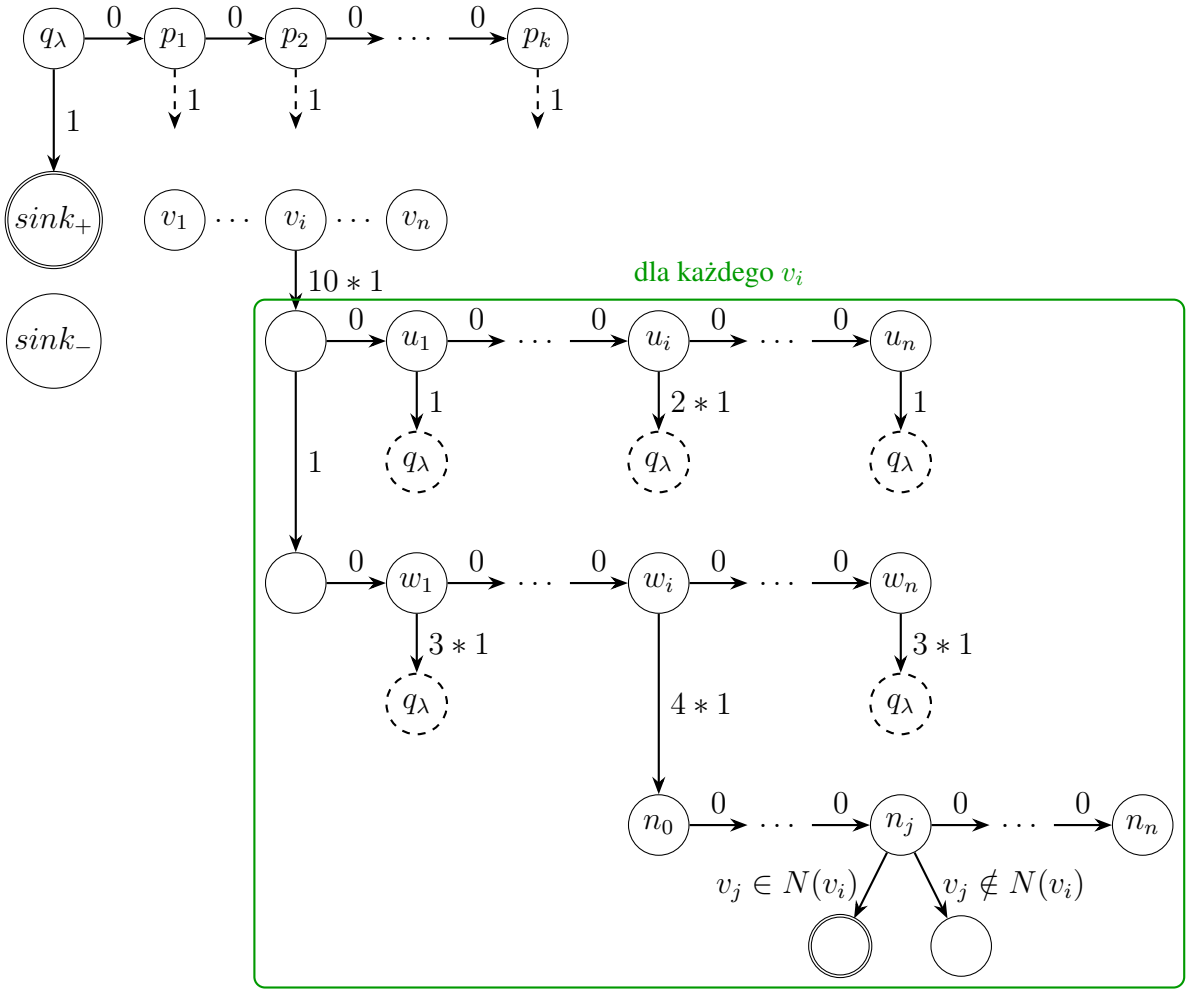
$$\forall (x, y \in \{1..k\} \wedge x \neq y)\ \forall i \in \{1..n\}\ [bin(x)\ 0\ t\ bin(i)\ 0\ bin(y)\ 0\ t\ bin(i)\ 0] \in S^+ \tag{2.8}$$

Gwarancja, że wybrane wierzchołki tworzą klikę

$$\forall(x, y \in \{1..k\} \wedge x \neq y)\, \forall i, j \in \{1..n\}\, [bin(x)\, 0\, t\, bin(i)\, 0\, bin(y)\, 0\, t\, bin(j)\, 1\, bin(i)] \in S^+ \tag{2.9}$$

Dla $|\Sigma| = 2$

Rysunek 2.4 przedstawia konstrukcję automatu dla redukcji z k-klik, korzystająca z alfabetu o rozmiarze 2.
Konstrukcja próbek dla zagwarantowania wyboru różnych stanów v_i oraz zagwarantowania, że wybrane wierzchołki tworzą klikę pozostaje analogiczna do poprzednich. Nie posiadamy już litery testowej t, więc musimy zmodyfikować zagwarantowanie prowadzenia brakujących krawędzi do wierzchołków v_i.



Rysunek 2.4. Konstrukcja automatu dla redukcji z k-klik, korzystająca z alfabetu o rozmiarze 2

Nowa idea automatu (rysunek 2.4) opiera się na przypisywaniu literom wartości od 1 do n, a następnie kodowaniu ich w systemie unarnym. W ten sposób każda litera jest reprezentowana jako ciąg liter 0.
W poniższym zapisie $un(a)$ oznacza reprezentację litery a w kodzie unarnym. Aby rozróż-

niać przejścia literą 1, będziemy je składać z różnych wielokrotności 1. Dlatego ustalamy zapis $d * a$ jako powtórzenie litery a dokładnie d razy.
Gwarancja prowadzenia brakujących krawędzi do wierzchołków v_i Zaczniemy od zagwarantowania, że brakujące krawędzie nie prowadzą do żadnego stanu akceptującego, w tym do stanu sink_+.

$$\forall x \in \{1..k\}\ [un(x)\ 1] \in S^- \tag{2.10}$$

Nie mamy już litery testowej t, więc musimy zmodyfikować automat oraz próbki tak, aby wymusić przejście brakującą krawędzią do któregoś ze stanów v_i. Możemy zastąpić literę t specjalnym ciągiem $10*1$, który nie prowadzi do $sink_-$. Wtedy ten sam ciąg $10*1$ wystąpi dwukrotnie w każdej próbce, zaraz po przejściu brakującą krawędzią.

Weźmy dowolną próbkę (2.12), oraz odpowiadające jej x, y, i, j. Jeżeli brakująca krawędź odpowiadająca stanowi p_y nie prowadzi do żadnego ze stanów $v_.$, to nie istnieje żaden inny stan do którego mogłaby ona prowadzić - musi prowadzić do stanu, po którym następuje przejście $11 * 1$, jedyne takie przejścia to $v_.$ oraz $sink_+$. Jednak przejście do $sink_+$ zostało wykluczone próbką (2.10), więc pozostaje tylko przejście do któregoś ze stanów $v_.$.

Jeżeli brakująca krawędzi odpowiadająca stanowi p_x nie prowadzi do żadnego ze stanów $v_.$, to może prowadzić tylko do stanów po których następuje przejście $10*1$. Jedynymi takimi stanami są stany $v_.$, $\delta(v_., 1)$ oraz $sink_+$ (który został wykluczony próbką (2.10)). Przejście do stanu $\delta(v_., 1)$, a następnie kolejnymi literami z próbik - $un(i)$ doprowadzi nas do krawędzi $3 * 1$ lub $4 * 1$, natomiast w próbce występuje $2 * 1$, a po nim $un(y)$ - czyli litera 0. To przejście zostanie więc zawszez odrzucone. Pozostaje więc tylko przejście do któregoś ze stanów $v_.$.

Wybór różnych stanów v_i dla różnych brakujących krawędzi

$$\forall (x, y \in \{1..k\} \wedge x \neq y)\, \forall i \in \{1..n\}\ [un(x)\ 11*1\ un(i)\ 2*1\ un(y)\ 11*1\ un(i)\ 2*1] \in S^+ \tag{2.11}$$

Gwarancja, że wybrane wierzchołki tworzą klikę

$$\forall (x, y \in \{1..k\} \wedge x \neq y)\, \forall i, j \in \{1..n\}\ [un(x)\ 11*1\ un(i)\ 2*1\ un(y)\ 12*1\ un(j)\ 4*1\ un(i)\ 1] \in S^+ \tag{2.12}$$

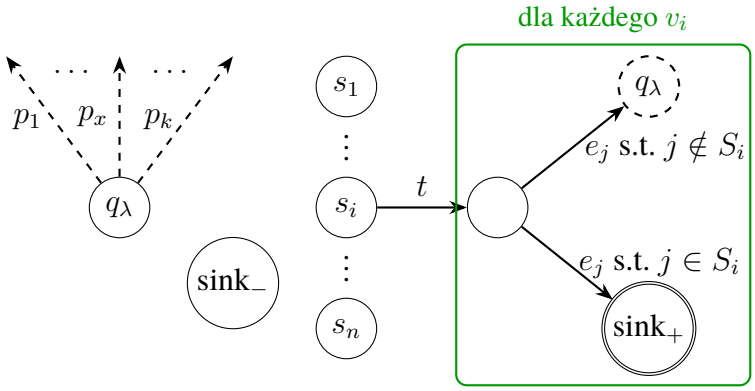
2.3.2 W[2]-trudność

Definicja 2.3.2 - Problem k-zbioru pokrywającego (k-set cover)
Weźmy rodzinę zbiorów $\mathcal{F}$ stworzoną na uniwersum U. Dla podrodziny $\mathcal{F}' \subset \mathcal{F}$ oraz podzbioru $U' \subset U$ mówimy, że $\mathcal{F}'$ pokrywa U', jeżeli $\bigcup_{S \in \mathcal{F}'} S \supseteq U'$. W problemie k-zbioru pokrywającego dane jest uniwersum U, rodzina zbiorów $\mathcal{F}$ stworzona na U oraz liczba naturalna k. Celem jest znalezienie podrodziny $\mathcal{F}' \subseteq \mathcal{F}$, takiej że $|\mathcal{F}'| \leq k$ oraz $\mathcal{F}'$ pokrywa U.
Problem k-zbioru pokrywającego należy do klasy W[2]-zupełnych problemów [5].

Pokażemy redukcję z problemu k-zbioru pokrywającego do problemu naprawienia częściowego DFA. Dla dowolnego wejścia do problemu k-zbioru pokrywającego skonstruujemy automat oraz próbki, będące wejściem do problemu naprawienia częściowego DFA, tak aby rozwiązanie problemu naprawienia częściowego DFA rozwiązywało problem k-zbioru pokrywającego.

Automat 2.5 z redukcji posiada rozmiar alfabetu zależny od rozmiaru uniwersum U - można go w analogiczny sposób jak w przypadku redukcji z k-klik zmniejszyć do stałego rozmiaru alfabetu.
Przyjmijmy też, że $n = |\mathcal{F}|$.

Idea konstrukcji:
Każdy stan s_i odpowiada zbiorowi $S_i \in \mathcal{F}$. Krawędzie e_j odpowiadają elementom $u_j \in U$. Jeżeli brakująca krawędź p_x prowadzi do stanu s_i, to interpretujemy to jako wybranie zbioru S_i jako x-tego zbioru w pokryciu.
Sekcja znajdująca się za wierzchołkami s_i prowadzi do stanu startowego (odrzucającego) q_λ, jeżeli dany zbiór S_i nie zawiera elementu u_j. W przeciwnym wypadku prowadzi do stanu akceptującego sink_+. Konstrukcja ta reprezentuje elementu z uniwersum U, które znajdują się w zbiorze S_i.



Rysunek 2.5. Konstrukcja automatu dla redukcji z k-zbioru pokrywającego, korzystająca z alfabetu o rozmiarze zależnym od liczby wierzchołków grafu

W tym problemie nie musimy gwarantować wyboru różnych stanów s_i dla różnych krawędzi p_x, ponieważ wybór tego samego zbioru wielokrotnie nie ma wpływu na pokrycie uniwersum.
Gwarancja pokrycia uniwersum
Aby zagwarantować, że wybrane zbiory pokrywają całe uniwersum, dla każdego elementu $u_j \in U$ tworzymy próbkę przechodzącą do każdego, wybranego przez krawędź p_x, stanu s_i oraz sprawdzającą czy dany element u_j znajduje się w zbiorze S_i. Jeżeli tak, próbka jest akceptowana, w przeciwnym wypadku odrzucana.

$$\forall j \in \{1..|U|\}\ [p_1\ t\ e_j\ p_2\ t\ e_j\ ...\ p_k\ t\ e_j] \in S^+ \tag{2.13}$$

Dodatkowo, w analogiczny sposób jak w przypadku redukcji z k-klik, tworzymy próbki wymuszające prowadzenie brakujących krawędzi do stanów s_i oraz wybór różnych stanów dla różnych krawędzi p_x.
Gwarancja prowadzenia brakujących krawędzi do stanów s_i
Poprowadzenie brakującej krawędzi do dowolnego stanu który nie jest s_i lub $sink_+$ doprowadzi do sink_- po przejściu następną literą t.
Aby zapobiec prowadzeniu brakujących krawędzi do stanu sink_+, dodajemy próbki odrzucające:

$$\forall x \in \{1..k\}\ [p_x] \in S^- \tag{2.14}$$

2.3.3 Przynależność do W[P]

Rozdział 3

Algorytmy

3.1 Podejście Brute Force

Podejście brute force polega na systematycznym sprawdzaniu wszystkich możliwych sposobów uzupełnienia brakujących przejść w automacie. Algorytm generuje kolejne warianty automatu poprzez przypisywanie stanów docelowych do brakujących przejść dla poszczególnych symboli alfabetu.

Każdy taki wariant traktowany jest jako osobny kandydat rozwiązania problemu. Dla wygenerowanego automatu algorytm symuluje następnie przetwarzanie wszystkich próbek, przechodząc krok po kroku przez kolejne stany zgodnie z symbolami próbek. Po zakończeniu symulacji sprawdzane jest, czy każda z próbek kończy się w stanie zgodnym z danymi wejściowymi.

Jeżeli dla danego wariantu automatu warunek ten nie jest spełniony, algorytm odrzuca go i przechodzi do analizy kolejnej kombinacji uzupełnień. Proces ten jest powtarzany aż do momentu znalezienia pierwszego wariantu, dla którego wszystkie próbki są poprawnie obsługiwane. W tym momencie algorytm kończy działanie.

Złożoność obliczeniowa

Czas algorytmu wynosi $\mathcal{O}(N^k \cdot M \cdot |S|)$, gdzie N to liczba stanów, k to liczba brakujących przejść, M to maksymalna długość próbki, a $|S|$ to liczba próbek.

3.2 Algorytm ze skokami (Jump Tables)

Algorytm ze skokami stanowi optymalizację symulacji przechodzenia próbką po automacie, z algorytmu Brute Force. Jego głównym celem jest zredukowanie liczby krawędzi, które muszą być przetworzone podczas weryfikacji próbki z automatem.
Możemy zaobserwować, że przechodzenie po *znanych* krawędziach - czyli takich przejściach które dostaliśmy na wejściu - może często prowadzić do redundantnych obliczeń, ponieważ występują one często w automacie, a utworzone z nich ścieżki są jednoznacznie określone dla dowolnej wersji naprawionego automatu.

Dlatego też konstrukcja algorytmu opiera się na stworzeniu struktury, która dla każdego możliwego sufiksu próbek oraz każdego stanu automatu, pozwala na natychmiastowe wyznaczenie stanu docelowego - osiągalnego przy użyciu wyłącznie znanych krawędzi, przeskakując przy tym możliwie najdłuższy fragment podanego sufiksu.
Do algorytmu wprowadzany jest etap wstępnego przetwarzania, gdzieopisana struktura

jest wyliczana w postaci tablic skoków. Następnie, podczas walidacji automatu, przechodząc próbką po automacie korzystamy z tablicy skoków, aby przeskoczyć fragmenty składające się ze *znanych* krawędzi.

W efekcie wykonamy jedynie tyle kroków, ile jest *brakujących* krawędzi na ścieżce odpowiadającej danej próbce w automacie.

Budowa tablic skoków

Tablice skoków budowane są niezależnie dla zbioru przykładów pozytywnych oraz negatywnych. Dla każdej próbki w o długości maksymalnie L konstruowana jest tablica DP, w której wpis $DP[i][q]$ opisuje efekt przetworzenia najdłuższego możliwego fragmentu próbki w, zaczynając od pozycji i w stanie q, bez użycia brakujących przejść.

Algorithm 1: Budowa tablic skoków

Input: Automat $A = (Q, \Sigma, \delta, q_0, F)$, zbiór próbek S
Output: Tablica skoków JT
foreach *próbka* $w \in S$ **do**
  $L \leftarrow |w|$;
  Utwórz tablicę $DP[0 \dots L][0 \dots |Q| - 1]$;
  foreach *stan* $q \in Q$ **do**
    $DP[L][q] \leftarrow (q, L)$;
  for $i \leftarrow L - 1$ **to** 0 **do**
    foreach *stan* $q \in Q$ **do**
      if $\delta(q, w[i])$ *jest nieokreślone* **then**
        $DP[i][q] \leftarrow (q, i)$;
      else
        $q' \leftarrow \delta(q, w[i])$;
        $DP[i][q] \leftarrow DP[i + 1][q']$;
  Dodaj DP do JT;
return JT

Walidacja automatu z użyciem tablic skoków

Po skonstruowaniu tablic skoków algorytm wykorzystuje je podczas walidacji każdego kandydata. Zamiast symulować próbki krok po kroku, algorytm wykonuje skoki pomiędzy pozycjami, aż napotka fragment zależny od brakującego przejścia.

Algorithm 2: Walidacja automatu z wykorzystaniem tablic skoków

Input: Automat A, próbki S, tablica skoków JT, oczekiwany wynik b
Output: true jeśli automat jest zgodny z próbkami, false w przeciwnym razie

```
foreach próbka w_i ∈ S do
    q ← q_0, pos ← 0;
    while pos < |w_i| do
        (q', pos') ← JT[i][pos][q];
        if (q', pos') = (q, pos) then
            if δ(q, w_i[pos]) jest nieokreślone then
                Przerwij symulację tej próbki;
            q ← δ(q, w_i[pos]);
            pos ← pos + 1;
        else
            q ← q', pos ← pos';
    if (q ∈ F) ≠ b then
        return false;
return true
```

Integracja z algorytmem pełnego przeszukiwania

Algorytm ze skokami nie modyfikuje samej strategii przeszukiwania przestrzeni możliwych uzupełnień brakujących przejść. Zastępuje on jedynie klasyczną procedurę walidacji automatu zoptymalizowaną wersją wykorzystującą tablice skoków. Dzięki temu zachowana zostaje pełna poprawność algorytmu brute force, przy jednoczesnym istotnym zmniejszeniu czasu weryfikacji pojedynczego kandydata w praktyce.

Analiza wydajności

Czas budowy tablic skoków wynosi $\mathcal{O}(|S| \cdot M \cdot N)$, gdzie $|S|$ to liczba próbek, M to maksymalna długość próbki, a N to liczba stanów automatu.

Z założeń wynika, że przy walidacji próbki wykonamy tyle kroków, ile jest brakujących przejść na ścieżce odpowiadającej danej próbce w automacie. W najgorszym przypadku, gdy wszystkie przejścia są brakujące lub co drugie przejście jest brakujące, czas walidacji pozostaje $\mathcal{O}(M)$. W praktyce jednak, dla automatu z niewielką liczbą brakujących przejść, czas ten może być znacznie mniejszy.

3.3 Heurystyka naprawy automatu z losowymi restartami

Algorytm hill climbing [6] jest algorytmem, który rozpoczyna działanie od pewnego początkowego rozwiązania, a następnie iteracyjnie wprowadza niewielkie modyfikacje, dążąc do poprawy wartości funkcji celu. W każdej iteracji rozważane są rozwiązania sąsiednie, różniące się od aktualnego jedynie pojedynczą zmianą, w tym przypadku: przypisaniem innego stanu docelowego do jednego przejścia automatu. Jeżeli taka modyfikacja prowadzi do poprawy wartości funkcji celu, w tym przypadku: więcej próbek osiąga docelowy stan zgodny z danymi wejściowymi, jest ona akceptowana jako nowe rozwiązanie bieżące.

Metoda ta jest podatna na zatrzymanie się w minimach lokalnych, ponieważ algorytm akceptuje jedynie zmiany prowadzące do poprawy rozwiązania. W celu ograniczenia tego problemu stosujemy losowe restarty algorytmu. Po osiągnięciu minimum lokalnego, w którym nie istnieje żadna poprawiająca modyfikacja, algorytm rozpoczyna działanie od nowej, losowo wygenerowanej konfiguracji niezidentyfikowanych przejśc automatu. Proces ten jest powtarzany wielokrotnie, co zwiększa prawdopodobieństwo odnalezienia rozwiązania.

Rozdział 4

Implementacja

Program został zaimplementowany w C++ 15. Kod jest gotowy do pobrania z publicznego repozytorium na github:

- `https://github.com/PatrykFlama/PracaInz`

Kod jest w folderze programy, w pliku main.cpp można konfigurować liczbę stanów, rozmiar alfabetu, liczbę próbek oraz ich długość, wariancję długości próbek, liczbę brakujących krawędzi, typ automatu oraz liczbę testów i wybrać algorytmy do testowania. Kod można uruchamiać za pomocą Makefile.

Rozdział 5

Eksperymenty

5.1 Środowisko

Testy zostały wykonane na środowisku o poniższych parametrach:

- System:
- Procesor:
- Architektura:

5.2 Sposób testowania

Testy przeprowadzaliśmy na stałych parametrach, jedynie badany parametr jest zmieniany. Automat oraz próbki przygotowywaliśmy w podany sposób:

- generowaliśmy losowy automat pełny(posiadający przejście dla każdego symbolu alfabetu w każdym stanie),
- na podstawie tak wygenerowanego automatu generowaliśmy losowe próbki,
- losowo usuwaliśmy ustaloną liczbę przejść z automatu, które są zawarte w próbce lub próbkach.

Dzięki temu automatycznie wykluczamy wszystkie przypadki niezdefiniowanych przejść w automacie, które mogą zostać poprowadzone w dowolny sposób. Takim sposobem eliminujemy trywialne przypadki, w których dowolny przebieg przejścia algorytmu prowadzi do poprawnego rozwiązania, koncentrując testy na trudniejszych instancjach problemu.

Przy rysowaniu wykresów średni czas wykonania obliczano z wykorzystaniem okna przesuwnego. Dane zostały posortowane względem badanego parametru, a następnie dla kolejnych fragmentów danych o stałej liczbie obserwacji wyznaczano średnią arytmetyczną wartości parametru oraz odpowiadających mu czasów wykonania. Okno przesuwne było centrowane, co oznacza, że każda wartość na wykresie reprezentuje średnią obliczoną z obserwacji znajdujących się symetrycznie wokół danego punktu. Zastosowanie okna przesuwnego pozwoliło na wygładzenie przebiegu wykresów oraz uwidocznienie ich tendencji.

5.3 Cel

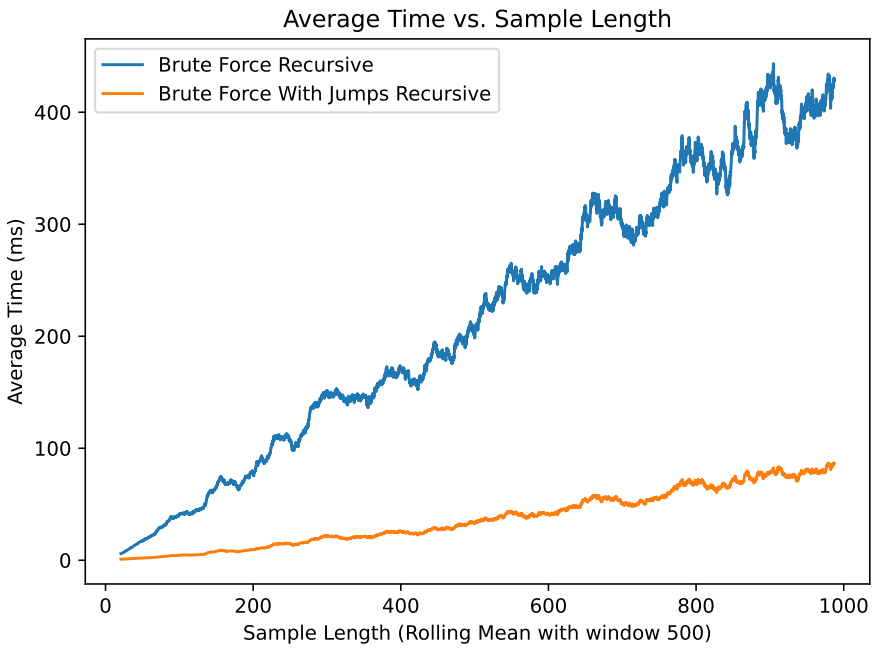
Celem przeprowadzonych eksperymentów było zbadanie wydajności algorytmu ze skokami, który stanowi najbardziej obiecujące podejście spośród rozważanych rozwiązań, oraz porównanie jego działania z algorytmem naiwnym. Testy obejmowały ocenę wpływu długości próbek oraz ich liczby na czas wykonania obu algorytmów.

5.4 Wyniki

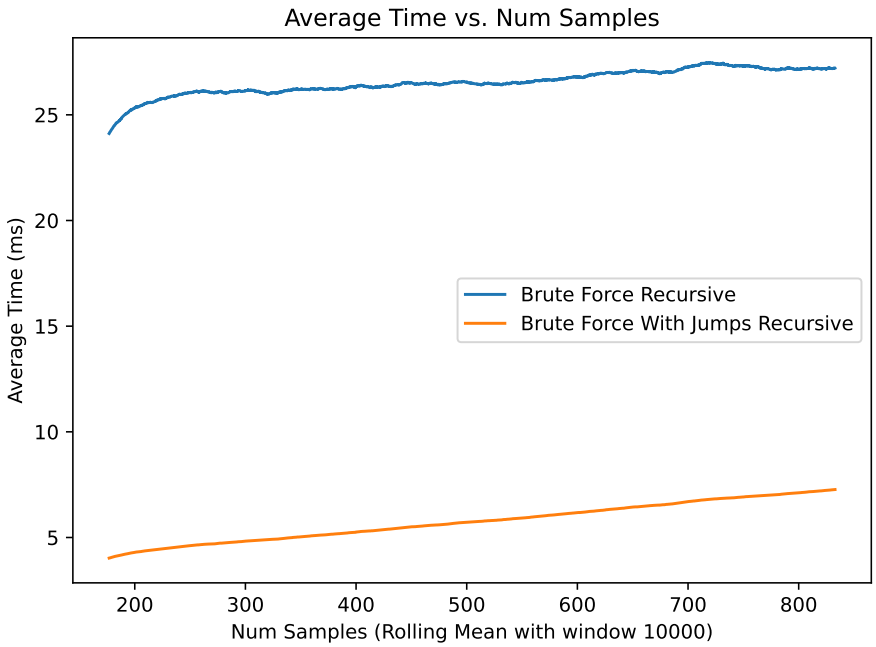
Na wykresie 5.1 widoczny jest liniowy wzrost czasu działania obu algorytmów wraz ze zwiększaniem długości próbek. Algorytm naiwny charakteryzuje się szybkim przyrostem czasu wykonania, osiągając dla największych długości próbek wartości kilkukrotnie wyższe niż algorytm ze skokami. Algorytm ze skokami wykazuje stabilniejszy przebieg oraz wolniejsze tempo wzrostu, co wskazuje na jego lepszą skalowalność względem długości próbek.

Na wykresie 5.2 podobnie obserwowany jest liniowy wzrost czasu działania obu algorytmów wraz ze zwiększaniem ilości próbek, przy czym wpływ liczby próbek jest wyraźnie mniejszy niż wpływ ich długości.

Na podstawie przeprowadzonych testów można stwierdzić, że algorytm ze skokami jest istotnie szybszy od algorytmu naiwnego dla badanych konfiguracji. Uzyskane wyniki potwierdzają jego przewagę wydajnościową zarówno względem długości, jak i liczby próbek. Zaobserwowane zachowanie jest zgodne z oczekiwaniami wynikającymi z charakterystyki analizowanych algorytmów.



Rysunek 5.1. Średni czas wykonania algorytmu w zależności od długości próbek, liczony na oknie przesuwnym 500. Parametry: 20 stanów, 30 próbek, alfabet 5-symbolowy, 4 brakujące krawędzie, wariancja długości próbek : 0,2, długość próbek z zakresu [30,1000].



Rysunek 5.2. Średni czas wykonania algorytmu w zależności od liczby próbek, liczony na oknie przesuwnym 10000. Parametry: 20 stanów, liczba próbek z zakresu [10,1000], alfabet 5-symbolowy, 4 brakujące krawędzie, wariancja długości próbek: 0,2, długość próbek: 30.

Rozdział 6

Podsumowanie

W niniejszej pracy wykazano, że rozważany problem jest W[1]-trudny oraz W[2]-trudny. Wyniki te pozwalają na umiejscowienie badanego zagadnienia w hierarchii klas złożoności parametryzowanej oraz wskazują na istotne ograniczenia możliwości skonstruowania algorytmów efektywnie parametryzowanych. Jednocześnie zasadne wydaje się podjęcie dalszych badań zmierzających do formalnego określenia, czy problem ten jest również W[P]-trudny, a także czy należy on do klasy W[P], co umożliwiłoby pełniejszą charakterystykę jego własności teoretycznych.

Przeprowadzone testy wskazują, że zaproponowany algorytm ze skokami stwarza potencjalne możliwości dalszej optymalizacji, w szczególności w zakresie zapotrzebowania na pamięć.

Istotnym kierunkiem dalszych badań jest również analiza wpływu struktury automatu na czas wykonania algorytmów. W szczególności interesujące wydaje się zbadanie automatu z różną ilością spójnych składowych.

Bibliografia

[1] E. M. Gold, „Language Identification in the Limit," *Information and Control*, s. 447–474, 1967.

[2] J. Lingg, M. de Oliveira Oliveira i P. Wolf, „Learning from Positive and Negative Examples: New Proof for Binary Alphabets," *arXiv preprint*, 2022.

[3] D. Dell'Erba, Y. Li i S. Schewe, „DFAMiner: Mining minimal separating DFAs from labelled samples," *arXiv preprint*, 2024.

[4] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[5] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk i S. Saurabh, *Parameterized Algorithms*. Springer, 2016.

[6] S. Russell i P. Norvig, *Artificial Intelligence: A Modern Approach*, 3 wyd. Prentice Hall, 2009.

[7] J. Flum i M. Grohe, *Parameterized Complexity Theory*. Springer, 2006.

Dodatek A

Aneks

A.1 Podział pracy

Julia Cygan oraz Patryk Flama wspólnie opracowali problem badawczy. Patryk Flama w przeważającej mierze rozwinął część teoretyczną pracy, w szczególności dowody twierdzeń. Julia Cygan w przeważającej mierze rozwinęła część implementacyjną pracy. Część implementacji miała charakter eksploracyjny i nie została bezpośrednio wykorzystana w treści pracy. Eksperymenty obliczeniowe zostały przeprowadzone wspólnie przez autorów.