# Repairing Finite Automata with Missing Components

Naprawianie automatów skończonych
z brakującymi stanami i krawędziami

**Julia Cygan, Patryk Flama**

Bachelor of Engineering Thesis
**Supervisor**: PhD Jakub Michaliszyn

University of Wrocław
Faculty of Mathematics and Computer Science
Institute of Computer Science

4 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 *Repairing Finite Automata with Missing Components* 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, 4 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 **gold1967**. 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 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 **binproof2022**.

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 **dfaminer2024**, 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], jak również jego przynależność do klasy W[p]. Oprócz wyników teoretycznych zaprezentowany zostanie algorytm, rozwiązujący ten problem w czasie lepszym niż podejście brute force, wykorzystujący skoki przez zdefiniowane krawędzie. Skuteczność zaproponowanego rozwiązania zostanie porównana z heurystyką lokalnej naprawy automatów. Dodatkowo przeanalizowany zostanie wpływ struktury automatu oraz zastosowanych strategii algorytmicznych na czas działania proponowanych metod.

# Rozdział 2

# Teoria

## 2.1 Definicja problemu, framework

**Definicja 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,
- $\Sigma$ 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 - 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 $\delta$ 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 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 $\delta$ 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

### 2.2.2 NP-trudność

## 2.3 FPT

### 2.3.1 W[1]-trudność

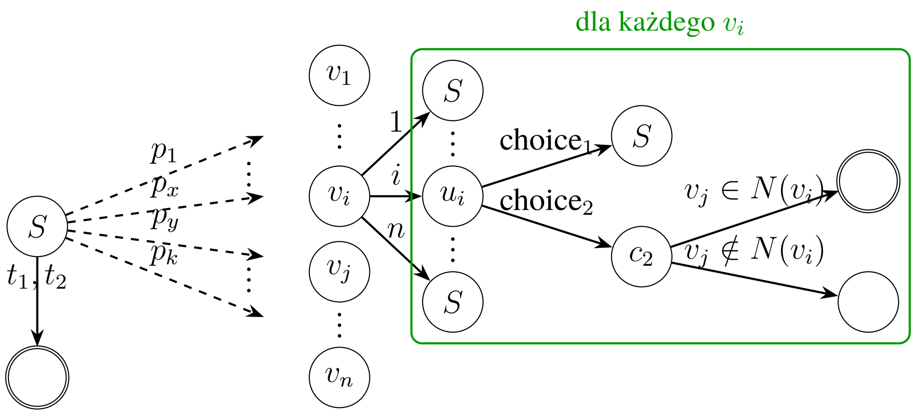
**Dla** $|\Sigma| = O(n)$

**Dla** $|\Sigma| = 3$

**Dla** $|\Sigma| = 2$

### 2.3.2 W[2]-trudność

### 2.3.3 Przynależność do W[P]




*Rysunek 2.1. Konstrukcja automatu dla redukcji z k-klik*

# Rozdział 3

# Algorytmy

## 3.1 Podejście Brute Force

## 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 $DP[0 \ldots L][0 \ldots |Q| - 1]$;
    **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* $w_i \in S$ **do**
    $q \leftarrow q_0$, $pos \leftarrow 0$;
    **while** $pos < |w_i|$ **do**
        $(q', pos') \leftarrow JT[i][pos][q]$;
        **if** $(q', pos') = (q, pos)$ **then**
            **if** $\delta(q, w_i[pos])$ *jest nieokreślone* **then**
                Przerwij symulację tej próbki;
            $q \leftarrow \delta(q, w_i[pos])$;
            $pos \leftarrow pos + 1$;
        **else**
            $q \leftarrow q'$, $pos \leftarrow pos'$;
    **if** $(q \in F) \neq b$ **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 $\mathcal{O}(N \cdot M \cdot |Q|)$, gdzie $N$ to liczba próbek, $M$ to maksymalna długość próbki, a $|Q|$ 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, czas walidacji pozostaje $\mathcal{O}(M)$, jednak w praktyce, dla automatu z niewielką liczbą brakujących przejść, czas ten może być znacznie mniejszy.

### 3.2.1 Heurystyka naprawy automatu z losowymi restartami

W celu praktycznego rozwiązania problemu naprawy brakujących przejść w deterministycznym automacie skończonym zastosowano heurystykę opartą na lokalnym przeszukiwaniu przestrzeni rozwiązań, uzupełnioną o mechanizm losowych restartów. Metoda ta ma na celu znalezienie automatu zgodnego ze wszystkimi przykładami pozytywnymi i negatywnymi, przy istotnie mniejszym koszcie obliczeniowym niż pełne przeszukiwanie metodą brutalną.

#### Ocena poprawności próbek

Podstawową operacją wykorzystywaną w algorytmie jest symulacja działania automatu na danej próbce wejściowej. Próbka uznawana jest za poprawnie sklasyfikowaną, jeżeli:

- w trakcie symulacji nie zostanie napotkane brakujące przejście,
- automat zakończy działanie w stanie akceptującym dla próbki pozytywnej,
- automat zakończy działanie w stanie nieakceptującym dla próbki negatywnej.

#### Hill Climbing

Główna część heurystyki opiera się na algorytmie hill climbing, który iteracyjnie modyfikuje przejścia automatu w celu minimalizacji liczby błędnie sklasyfikowanych próbek.

W każdej iteracji algorytm:

1. identyfikuje zbiór próbek błędnie sklasyfikowanych przez aktualny automat,
2. dla każdej takiej próbki przechodzi przez kolejne symbole słowa,
3. dla każdego przejścia testuje wszystkie możliwe stany docelowe,
4. wybiera przejście minimalizujące całkowitą liczbę błędnych klasyfikacji.

Jeżeli w danej iteracji nie nastąpi żadna poprawa, algorytm kończy działanie, uznając, że osiągnięto minimum lokalne.

### Losowe restarty

Ponieważ algorytm hill climbing może zatrzymać się w minimum lokalnym, zastosowano mechanizm losowych restartów. Każdy restart polega na:

- przywróceniu wybrakowanego automatu do stanu początkowego,
- losowej inicjalizacji wszystkich brakujących przejść,
- ponownym uruchomieniu procedury hill climbing.

Jeżeli w trakcie któregoś restartu zostanie znaleziony automat zgodny ze wszystkimi próbkami, algorytm kończy działanie sukcesem.

### Złożoność obliczeniowa

Niech:

- $n$ oznacza liczbę stanów automatu,
- $m$ maksymalną długość próbki,
- $|S|$ łączną liczbę próbek,
- $I$ maksymalną liczbę iteracji hill climbing,
- $R$ liczbę losowych restartów.

Złożoność heurystyki z losowymi restartami wynosi:

$$\mathcal{O}(R \cdot I \cdot |S| \cdot m \cdot n),$$

co w praktyce pozwala na znaczące przyspieszenie względem algorytmu brutalnego kosztem braku gwarancji znalezienia rozwiązania optymalnego.

**Pseudokod algorytmu**

**Algorithm 3:** Heurystyczna naprawa automatu z losowymi restartami

```
Input: Wybrakowany automat A = (Q, Σ, δ, q0, F), próbki pozytywne S+,
       próbki negatywne S−
Output: Naprawiony automat A′ lub informacja o niepowodzeniu
for r ← 1 to R do
    A_r ← A;
    Losowo uzupełnij wszystkie brakujące przejścia w A_r;
    for i ← 1 to I do
        invalid ← {w ∈ S+ ∪ S− | A_r błędnie klasyfikuje w};
        if invalid = ∅ then
            return A_r
        any_change ← false;
        foreach w ∈ invalid do
            q ← q0;
            for j ← 0 to |w| − 1 do
                a ← w[j];
                q_old ← δ(q, a);
                best ← q_old;
                best_errors ← |invalid|;
                foreach p ∈ Q do
                    if p ≠ q_old then
                        tymczasowo ustaw δ(q, a) ← p;
                        errors ← liczba błędnie sklasyfikowanych próbek z
                          S+ ∪ S−;
                        if errors < best_errors then
                            best ← p;
                            best_errors ← errors;
                ustaw δ(q, a) ← best;
                if best ≠ q_old then
                    any_change ← true;
                q ← best;
        if any_change = false then
            przerwij
return niepowodzenie
```

# Dodatek A

# Aneks

append an x