



**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,  
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

*Automatyczne odkrywanie procesów biznesowych przy użyciu  
programowania genetycznego*

*Automated Business Process Discovery using Genetic Programming*

Autor:

*Piotr Seemann*

Kierunek studiów:

*Informatyka*

Opiekun pracy:

*dr inż. Krzysztof Kluza*

Kraków, 2020

*Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.*

*Serdecznie dziękuję ...*



## Spis treści

<b>1. Wprowadzenie</b>	7
1.1. Cele pracy	7
1.2. Przegląd prac	7
1.3. Zawartość pracy	7
<b>2. Wstęp teoretyczny</b>	9
2.1. Procesy biznesowe	9
2.1.1. Procesy Biznesowe	9
2.1.2. Dzienniki zdarzeń	9
2.2. Modelowanie procesów biznesowych	9
2.3. Algorytmy do wykrywania procesów biznesowych	9
2.4. Ewolucja genetyczna	9
2.5. Gramatyka	9
2.5.1. BNF	9
2.5.2. Możliwe problemy przy tworzeniu gramatyki	9
2.6. Metryki	9
2.6.1. Stopień złożoności	10
2.6.2. Odwzorowanie	10
2.6.3. Precyzja	10
2.6.4. Generalizacja	10
<b>3. Projekt i implementacja</b>	11
3.1. Wykorzystane technologie	11
3.1.1. Python 3.8.1	11
3.1.2. PonyGE2	11
3.2. Tworzenie gramatyki procesu biznesowego	11
3.3. Projekt systemu	13
3.4. Implementacja	13
3.5. Wybór parametrów algorytmu	16

<b>4. Dyskusja rezultatów.....</b>	<b>19</b>
4.1. Przykładowe wyniki .....	19
4.2. Porównanie z innymi algorytmami.....	19
4.3. Wyniki w zależności od przyjętych metryk.....	19
4.4. Wnioski.....	19
<b>5. Podsumowanie .....</b>	<b>21</b>

# **1. Wprowadzenie**

## **1.1. Cele pracy**

## **1.2. Przegląd prac**

## **1.3. Zawartość pracy**





## **2. Wstęp teoretyczny**

### **2.1. Procesy biznesowe**

#### **2.1.1. Procesy Biznesowe**

#### **2.1.2. Dzienniki zdarzeń**

### **2.2. Modelowanie procesów biznesowych**

### **2.3. Algorytmy do wykrywania procesów biznesowych**

#### **2.3.0.1. Alpha algorithm**

#### **2.3.0.2. The ILP Miner**

#### **2.3.0.3. Heuristic Miner**

#### **2.3.0.4. Multi-phase Miner**

### **2.4. Ewolucja genetyczna**

[1]

### **2.5. Gramatyka**

#### **2.5.1. BNF**

#### **2.5.2. Możliwe problemy przy tworzeniu gramatyki**

### **2.6. Metryki**

[2]

**2.6.1. Stopień złożoności****2.6.2. Odwzorowanie****2.6.3. Precyzja****2.6.4. Generalizacja**

## 3. Projekt i implementacja

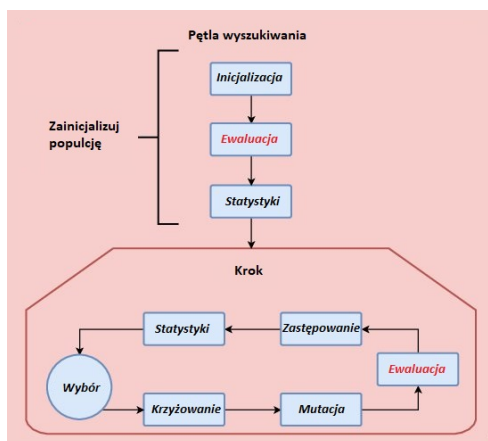
### 3.1. Wykorzystane technologie

#### 3.1.1. Python 3.8.1

Do implementacji algorytmu został użyty Python. Jest to najpopularniejszy język programowania w dziedzinie uczenia maszynowego. Wymagana jest wersja 3.8+ ze względu na użycie w implementacji metod dostępnych od tej wersji.

#### 3.1.2. PonyGE2

PonyGE2 [3] jest implementacją ewolucji genetycznej w języku Python. Pozwala na łatwą konfigurację parametrów ewolucji genetycznej oraz łatwą możliwość dodania własnych problemów oraz sposobów ewaluacji wyników.



Rys. 3.1. Pętla wyszukiwania

### 3.2. Tworzenie gramatyki procesu biznesowego

Przy tworzeniu gramatyki procesu biznesowego ważnym jest, żeby znaleźć balans, jeśli chodzi o poziom skomplikowania zaproponowanej gramatyki. W pracy [4] autorzy przeanalizowali składniki języka BPMN pod kątem częstotliwości ich stosowania. z pracy wynika, że najczęściej stosowanymi elementami

modelów procesu biznesowego, jeśli chodzi o bramki są: xor, and oraz pętle lop. Do przedstawionej poniżej gramatyki dodano także bramkę opt, czyli or jako uogólnienie bramki xor w celu uniknięcia zagnieżdżonych bramek xor. Ponadto koniecznym jest posiadanie bramki seq, która oznacza normalny przepływ procesów.

Zapis `GE_RANGE:dataset_n_vars` jest rozszerzenie do gramatyki zapewnianym przez PonyGE2, które umożliwia dodanie ilości zmiennych odpowiadającej ich ilości w zbiorze danych.

```
<e> ::= <anygateexceptseq>

<anygateexceptseq> ::= <anygateexceptseq><anygateexceptseq> | <andgate> |
<xorgate> | <optgate> | <lopgate> | {<event>}

<anygate> ::= <anygate><anygate> | <andgate> | <xorgate> | <seqgate> | <optgate> |
<lopgate> | {<event>}

<andgate> ::= and(<anygate><anygate>)

<xorgate> ::= xor(<anygate><anygate>)

<seqgate> ::= seq(<anygateexceptseq><anygateexceptseq>)

<optgate> ::= opt(<anygatenosingleopt>)

<optdoublegate> ::= opt(<anygate><anygate>)

<anygatenosingleopt> ::= <anygate><anygate> | <andgate> | <xorgate> | <seqgate> |
<lopgate> | {<event>}

<lopgate> ::= lop(<anygatenosinglelop>)

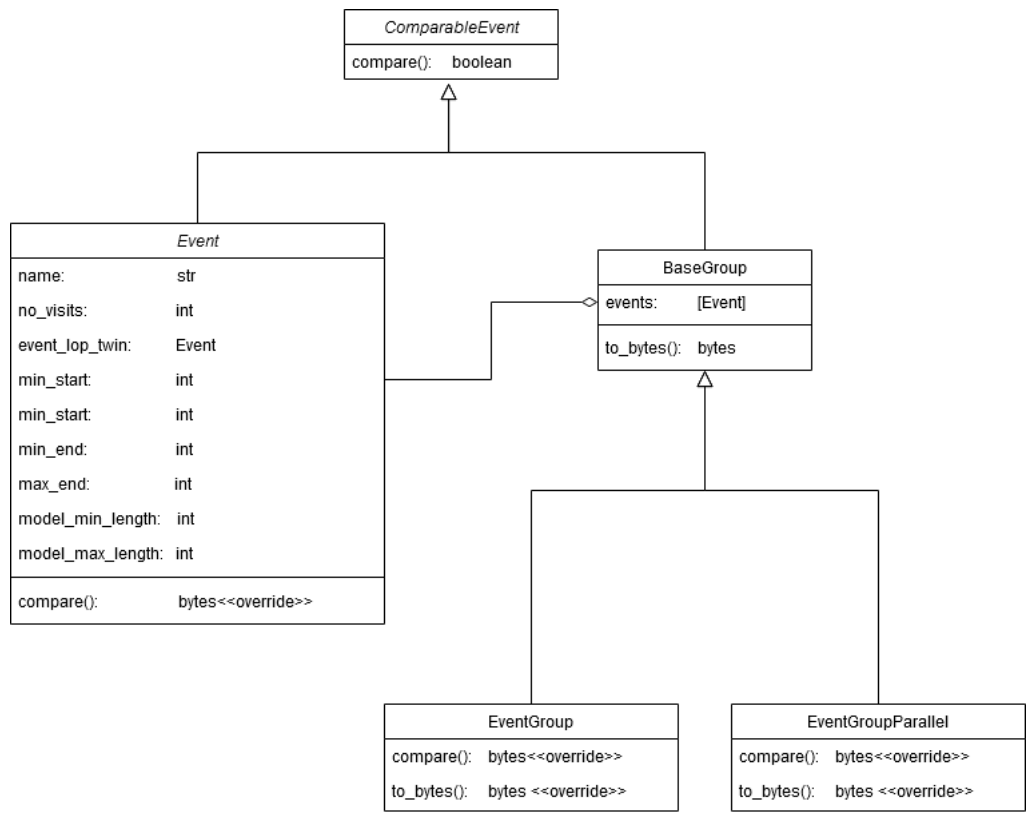
<anygatenosinglelop> ::= <anygateexceptseq><anygateexceptseq> | <andgate> |
<xorgate> | <seqgate> | <optdoublegate> | {<event>}

<event> ::= GE_RANGE:dataset_n_vars
```

**Listing 3.1.** Gramatyka procesu biznesowego

Przykład wygenerowanej gramatyki: `and({d}opt({f})and({a}{c})lop(seq(lop({a}){e})))`

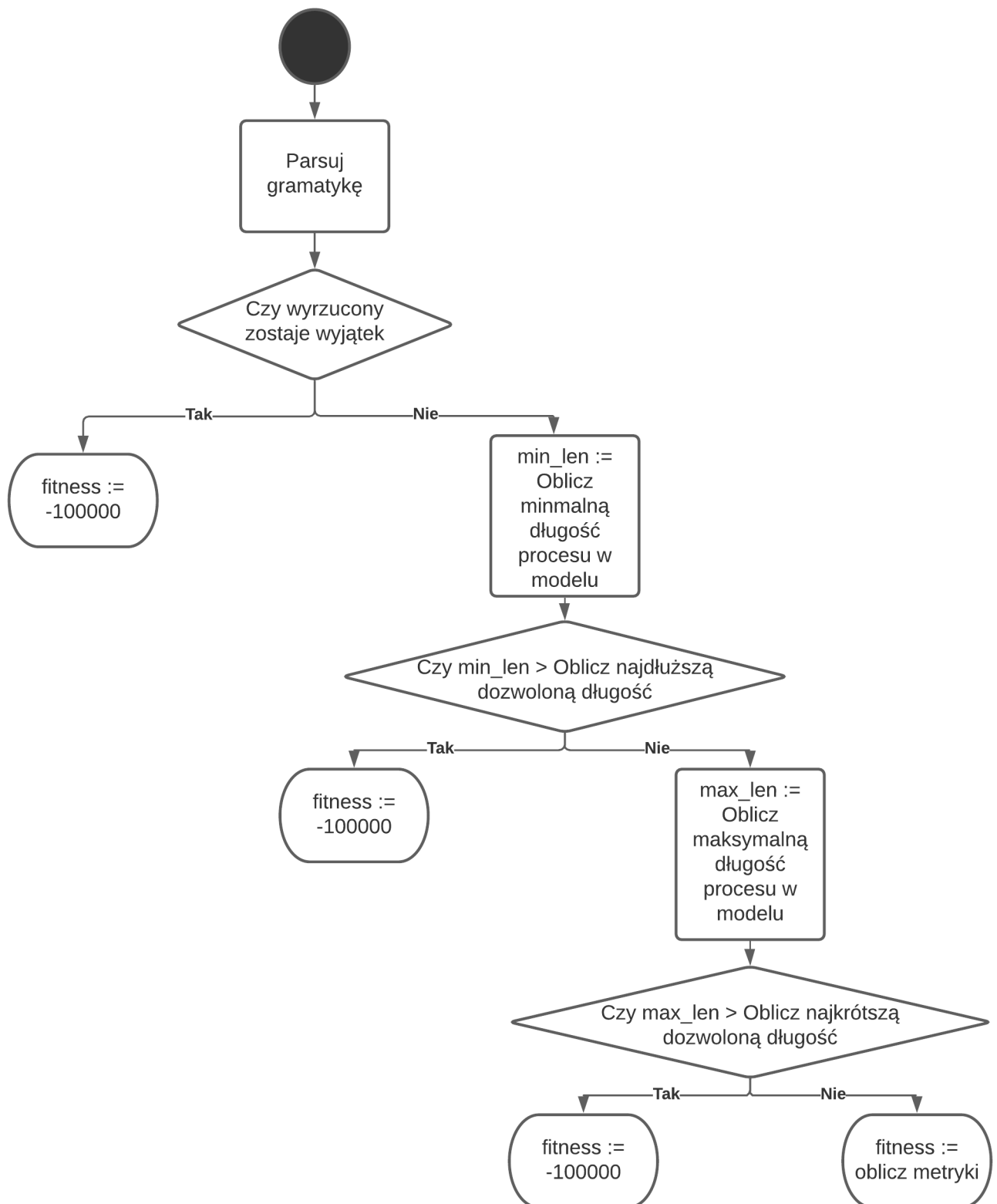
3.3. Projekt systemu



Rys. 3.2. Event UML

3.4. Implementacja

3.4.0.1. Ogólny schemat blokowy



Rys. 3.3. Ogólny schemat blokowy

### 3.4.0.2. Parsowanie gramatyki

Parser pozwala na przedtworzenie wyników uzyskanych na drodze ewolucji gramatycznej na postać, na której łatwiej będzie operować. Rezultaty uzyskane na drodze rewolucji gramatycznej w PonyGE2 są w formie tekstowej, z którą praca byłaby niewygodna, dlatego używamy parsera, żeby otrzymać wynik w postaci drzewa obiektów Gate, którego liśćmi będą obiekty Event. Parsując korzystamy z faktu, że przy projektowaniu gramatyki wszystkie bramki logiczne zostały oznaczone 3 literowymi symbolami, a wszystkie zdarzenia otoczone nawiasami klamrowymi. Tworząc każdy obiekt Event dodajemy informację o liczbie dzieci tego obiektu, co przyda nam się przy obliczaniu metryki precyzja.

```
def parsuj(wyrażenie: str) -> int:

    lokalnie_dodane_zdarzenia = []
    for i w zakresie długość_wyrażenia:
        if wyrażenie[i] == "{":
            zdarzenie := Event(wyrażenie[i + 1])
            dodaj zdarzenie do lokalnie_dodane_zdarzenia
            dodaj zdarzenie do aktualnie parsowanej bramki
            i += 2
        elif wyrażenie[i] == ")":
            return i+1
        elif i+4 < długość_wyrażenia:
            bramka := stwórz nową bramkę Gate typu zgodnego z wyrażeniem
            i += 3
            przeparsowane_znaki = bramka.parsuj(wyrażenie[i+4:])
            dodaj zdarzenie do aktualnie parsowanej bramki
            i += ilość_przeparsowanych_znaków
        else:
            wyrzuc wyjątek
    for zdarzenie w lokalnie_dodane_zdarzenia:
        zdarzenie.ilość_dzieci += 1
```

**Listing 3.2.** Parser gramatyki

### 3.4.0.3. Obliczanie metryk

### 3.4.0.4. Wyszukiwanie w modelu procesów o określonej długości

Algorytm rekurencyjny. Implementacja różni się w zależności od przeszukiwanej bramki logicznej.

### 3.4.0.5. Obliczanie dopasowania

Pomysł zaczerpnięty z algorytmu Needleman-Wunsch [5], który jest uogólnieniem odległości Levenshteina. Rozwinięty o możliwość przeszukiwania modelu rekurencyjnie oraz o możliwość podawania listy równołącznych zdarzeń.

```

def oblicz_metryki(log, długość_logu, średnia_długość_procesu_w_logu, model,
                  najkrótsza_dozwolona_długość,
                  najdłuższa_dozwolona_długość) -> int:
    n = zaokrąglij(średnia_długość_processu_w_logu)
    i = 1
    najlepszy_wynik = 0
    while n znajduje się pomiędzy największą i najmniejszą
        dopuszczalną długością logu:
        if najkrótsza_dozwolona_długość <= n <= najdłuższa_dozwolona_długość:
            procesy_o_dlugosci_n = model.wyszukaj_procesy_o_określonej_długości(n)
            wszystkie_procesy = model.wyszukaj_wszystkie_procesy
            wszyscy_rodzice_procesów = model.wyszukaj_wszystkich_rodziców_procesów
            if istnieją procesy_o_dlugosci_n:
                najlepszy_błąd_lokalny = 0
                for elem w log:
                    minimalny_błąd_lokalny = 1023
                    for proces w procesy_o_dlugosci_n:
                        błąd_lokalny, ścieżka = oblicz_dopasowanie(event_group, elem)
                        if błąd_lokalny < minimalny_błąd_lokalny:
                            minimalny_błąd_lokalny = błąd_lokalny
                            najlepsza_ścieżka = ścieżka
                    najlepszy_błąd_lokalny += minimalny_błąd_lokalny
                oblicz_metryki odwzorowanie, precyzja, generalizacja, prostota
                najlepszy_wynik_lokalny = suma_obliczonych_metryk / liczba_metryk
                if najlepszy_wynik_lokalny > najlepszy_wynik:
                    najlepszy_wynik = najlepszy_wynik_lokalny
            if i % 2 == 1:
                n -= i
            else:
                n += i
            i += 1
    return najlepszy_wynik

```

**Listing 3.3.** Obliczanie metryk**Listing 3.4.** Wyszukiwanie procesów o długości n

#### 3.4.0.6. Znajdowanie ścieżki w modelu

Potrzebne do obliczenia precyzji oraz generalizacji.

### 3.5. Wybór parametrów algorytmu



```
def oblicz_dopasowanie(model, log):
    kara = {'DOPASOWANIE': 0, 'BRAK_DOPASOWANIA': -2, 'PRZERWA': -1}
    wyniki_lokalne = [None] * dlugosc_macierzy_rozwiazan
    macierz_rozwiazan = Zainicjalizuj_macierz_zerami
    for i in range(m):
        macierz_rozwiazan[i][0] = kara['PRZERWA'] * i
    for j in range(n):
        macierz_rozwiazan[0][j] = kara['PRZERWA'] * j
    #Wypełnij osie macierzy właściwymi wartościami

    for i in range(1, m):
        if should_go_recurrent(model[i-1]):
            al_mat[i], model_results_local[i] =
                recurrent_alignment(al_mat[i - 1], model[i - 1],
                                   [x for x in substrings_of_string_reversed(log)], i)
        elif len(model[i-1]) > 1:
            al_mat[i], model_results_local[i] =
                parallel_alignment(al_mat[i - 1], model[i - 1],
                                  [x for x in substrings_of_string_reversed(log)], kara, i)
        else:
            al_mat[i][0] = al_mat[i-1][0] + penalty['GAP']
            basic_alignment(al_mat, model[i - 1], log, kara, i, n)

    model_results = get_all_tracebacks(al_mat, penalty['GAP'], model,
                                       log, model_results_local)

    return macierz_rozwiazan[m-1] #ostatnia linijka, model_results
```

**Listing 3.5.** Obliczanie dopasowania

```

def znajdz_sciezke(macierz_rozwiazan, model, log, rozwiazania_podmodeli):
    sciezka = []

    while i != 0:
        event_group_full_length = len(model[i - 1])
        if model_results_local[i] is not None:
            matched_flag = False
            if array[i][j] == array[i - 1][j] + event_group_full_length * penalty_gap:
                [model_result.append(None) for _ in range(event_group_full_length)]
                array[i][j] = 0
                i -= 1

            else:
                for k in range(j):
                    processes = get_not_none(model_results_local[i][k]
                        [len(model_results_local[i][k]) - (j-k)], log)
                    if array[i][j] == array[i - 1][k] +
                        (event_group_full_length + (j-k) - 2 * len(processes)) * penalty_gap:
                        [model_result.append(x) for x in processes]
                        for x in processes:
                            log = log.replace(x.name, "", 1)
                        [model_result.append(None)
                            for _ in range(event_group_full_length - len(processes))]
                        array[i][j] = 0
                        i -= 1
                        j = k
                        matched_flag = True
                        break

                if not matched_flag:
                    if array[i][j] == array[i][j - 1] + penalty_gap:
                        array[i][j] = 0
                        j -= 1

            else:
                if array[i][j] == array[i - 1][j] + penalty_gap:
                    model_result.append(None)
                    array[i][j] = 0
                    i -= 1
                elif array[i][j] == array[i][j - 1] + penalty_gap:
                    array[i][j] = 0
                    j -= 1
                elif array[i][j] == array[i - 1][j - 1]:
                    model_result.append(model[i-1])
                    log = log.replace(model[i-1].name, "", 1)
                    array[i][j] = 0
                    i -= 1
                    j -= 1

    return sciezka

```

## **4. Dyskusja rezultatów**

### **4.1. Przykładowe wyniki**

### **4.2. Porównanie z innymi algorytmami**

### **4.3. Wyniki w zależności od przyjętych metryk**

### **4.4. Wnioski**



## **5. Podsumowanie**



## Bibliografia

- [1] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992. ISBN: 0262111705.
- [2] J. C. A. M. Buijs, B. F. van Dongen i W. M. P. van der Aalst. „Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalization and Simplicity”. W: *International Journal of Cooperative Information Systems* 23.01 (2014), s. 1440001. DOI: 10.1142/S0218843014400012. eprint: <https://doi.org/10.1142/S0218843014400012>.
- [3] Michael Fenton i in. „PonyGE2”. W: *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2017). DOI: 10.1145/3067695.3082469.
- [4] Michael zur Muehlen i Jan Recker. „How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation”. W: *Advanced Information Systems Engineering*. Red. Zohra Bellahsene i Michel Léonard. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 465–479. ISBN: 978-3-540-69534-9.
- [5] Saul B. Needleman i Christian D. Wunsch. „A general method applicable to the search for similarities in the amino acid sequence of two proteins”. English (US). W: *Journal of Molecular Biology* 48.3 (mar. 1970), s. 443–453. ISSN: 0022-2836. DOI: 10.1016/0022-2836(70)90057-4.