



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, 2021

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

1. Wprowadzenie	7
1.1. Cele pracy	7
1.2. Przegląd prac	7
1.3. Zawartość pracy	7
2. Wstęp teoretyczny	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
2.4.1. Algorytmy genetyczne	9
2.4.2. Ewolucja genetyczna a inne algorytmy uczenia maszynowego	9
2.4.3. Ewolucja gramatyczna	9
2.5. Gramatyka	9
2.5.1. BNF	9
2.5.2. Tworzenie gramatyki pod kątem ewolucji	9
2.6. Metryki	10
2.6.1. Prostota	10
2.6.2. Odwzorowanie	10
2.6.3. Precyzja	10
2.6.4. Generalizacja	10
2.6.5. Złożoność	10
3. Projekt i implementacja	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	14
3.4.	Implementacja	15
3.5.	Wybór parametrów algorytmu	26
4.	Dyskusja rezultatów	27
4.1.	Przykładowe wyniki	27
4.2.	Porównanie z innymi algorytmami	27
4.3.	Wyniki w zależności od przyjętych metryk	27
4.4.	Wnioski	27
5.	Podsumowanie	29

1. Wprowadzenie

1.1. Cele pracy

Celem pracy jest

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

2.4.1. Algorytmy genetyczne

Pierwszy raz zaproponowane w [1]

2.4.2. Ewolucja genetyczna a inne algorytmy uczenia maszynowego

2.4.3. Ewolucja gramatyczna

2.5. Gramatyka

2.5.1. BNF

2.5.2. Tworzenie gramatyki pod kątem ewolucji

Tworząc gramatykę pod kątem wykorzystania jej w procesie ewolucji ważne jest, żeby ilość produkcji jak najlepiej odzwierciedlała jak często chcemy uzyskać dany stan.

2.6. Metryki

[2]

2.6.1. Prostota

$$M_{pro} = 1 - \frac{\text{ilosc duplikatow w modelu} + \text{ilosc brakujacych wartosci w modelu}}{\text{ilosc unikalnych zdarzen w logu} + \text{ilosc zdarzen w modelu}}$$

2.6.2. Odwzorowanie

$$M_o = (1 - \sum_0^{\text{ilosc procesow w logu}} \frac{\text{blad odwzorowania logu w modelu}}{\text{minimalna dugosc sciezki w modelu} + \text{dugosc sciezki w logu}})^4$$

2.6.3. Precyzja

$$M_{pre} = (1 - \sum_0^{\text{ilosc zdarzen w modelu}} \frac{\text{ilosc osiagalnych zdarzen w modelu} - \text{ilosc osiagalnych zdarzen w logu}}{\text{ilosc osiagalnych zdarzen w modelu}})^{\frac{1}{3}}$$

2.6.4. Generalizacja

$$M_g = \frac{1 - \sum_0^{\text{ilosc zdarzen w logu}} \frac{1}{\sqrt{\text{ilosc wystapien zdarzenia}}}}{\text{ilosc zdarzen w logu}}$$

2.6.5. Złożoność

$$M_z = 1 - \frac{1}{\sqrt{1 - \text{odwzorowanie}} * \sqrt{\text{zlozonosc modelu}}}$$

3. Projekt i implementacja

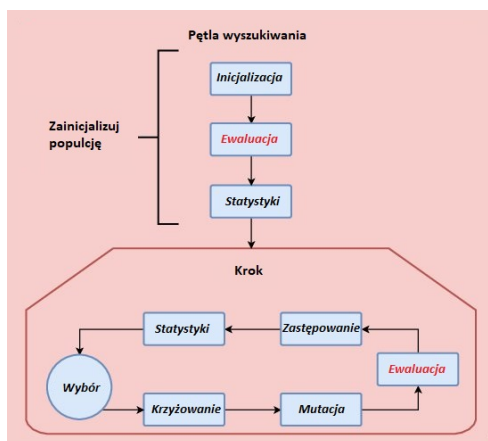
3.1. Wykorzystane technologie

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:n` jest rozszerzenie do gramatyki zapewnianym przez PonyGE2, które umożliwia dodanie ilości zmiennych, czyli `GE_RANGE:2` oznacza 0|1|2. Wzorując się na Zapis `GE_RANGE:dataset_vars` jest rozszerzenie do gramatyki zapewnianym przez PonyGE2, które umożliwia dodanie ilości zmiennych odpowiadającej ich ilości w zbiorze danych.

```
<e> ::= <slot><slot><anygate><slot><slot>

<anygate> ::= <anygate><anygate> | <name>(<slots>) | {<event>}

<slot> ::= <anygate> | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' '

<slots> ::= <slot><slot><anygate><slot><slot>

<name> ::= and | xor | seq | opt | lo<0_n>

<event> ::= GE_RANGE:dataset_vars

<0_n> ::= GE_RANGE:5
```

Listing 3.1. Gramatyka procesu biznesowego

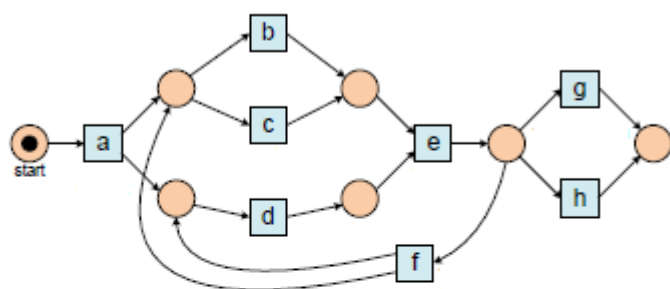
Przykład wygenerowanej gramatyki: `and({d}opt({f})and({a}{c})lop(seq(lop({a}){e})))`

Wszystkie bramki mają nazwy tej samej długości - 3 znaki, co pozwoli na łatwiejsze parsowanie gramatyki.

longate - oznacza pętle Poniższy przykład pokazuje gramatykę, którą ciężko opisać przy pomocy podstawowych bramek logicznych:

Jest to możliwe za pomocą notacji: `{a}and(xor({b}{c}){d}){e}lop({f}and(xor({b}{c}){d}){e})xor({g}{h})`

Użycie powyższej notacji rodzi jednak kilka problemów, Musimy mieć produkcje `{a}lo1({f}and(xor({b}{c}){d}){e})xor({g}{h})`

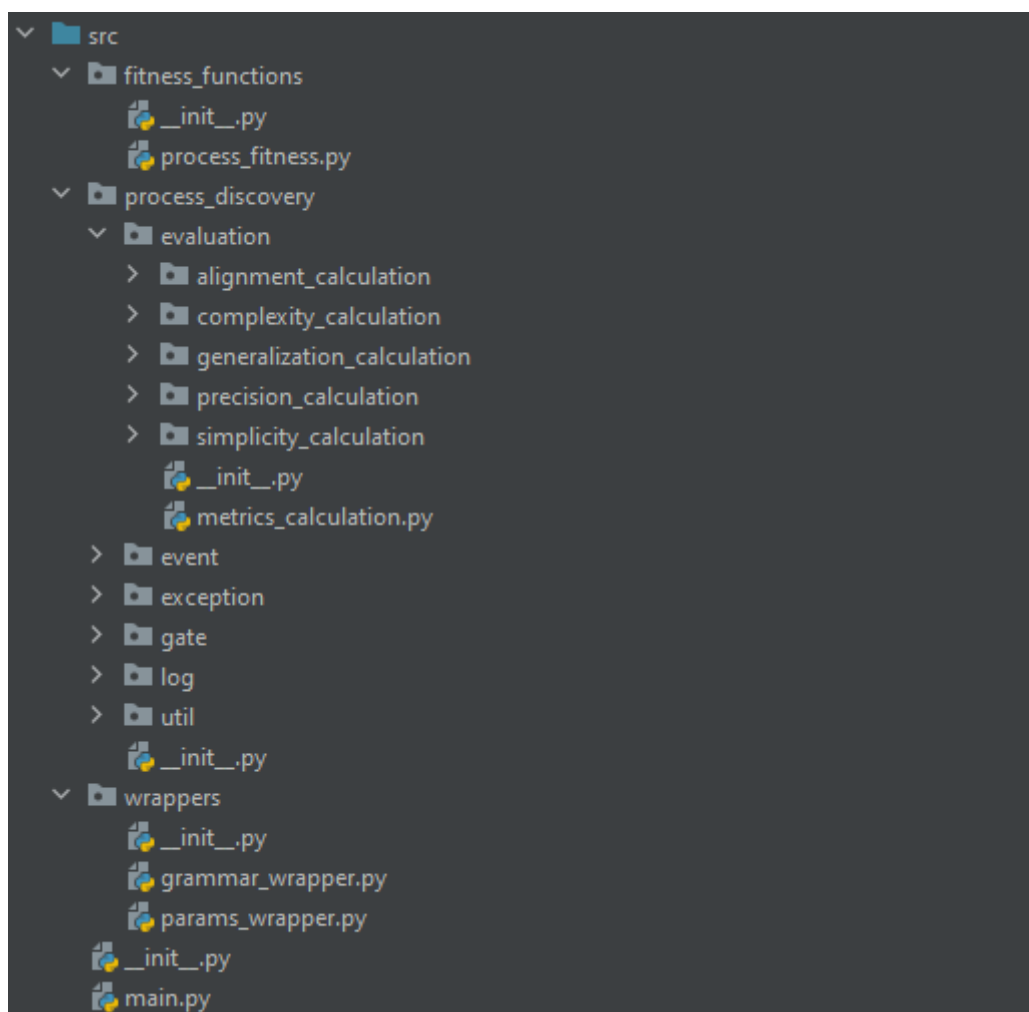
**Rys. 3.2.** Przykład problemu

3.3. Projekt systemu

3.3.0.1. Podział na moduły

Implementację podzielone na następujące moduły:

- wrappers - PonyGE2 nie jest przystosowane do zaimportowania jako biblioteka, dlatego żeby oddzielić kod PonyGE2 od naszego kodu należało rozszerzyć lub nadpisać część z modułów PonyGE2. Moduły, które nadpisano to params, który zawiera konfigurację aplikacji oraz grammar, gdzie dodano zmiany w jaki sposób parsowana jest podana gramatyka.
- fitness_functions - zawiera klasę bazowy moduł, gdzie znajduje się bazowa klasa dla obliczania metryk
- process_discovery - moduł zawiera całą logikę obliczenia metryk



Rys. 3.3. Podział na moduły

3.3.0.2. Model

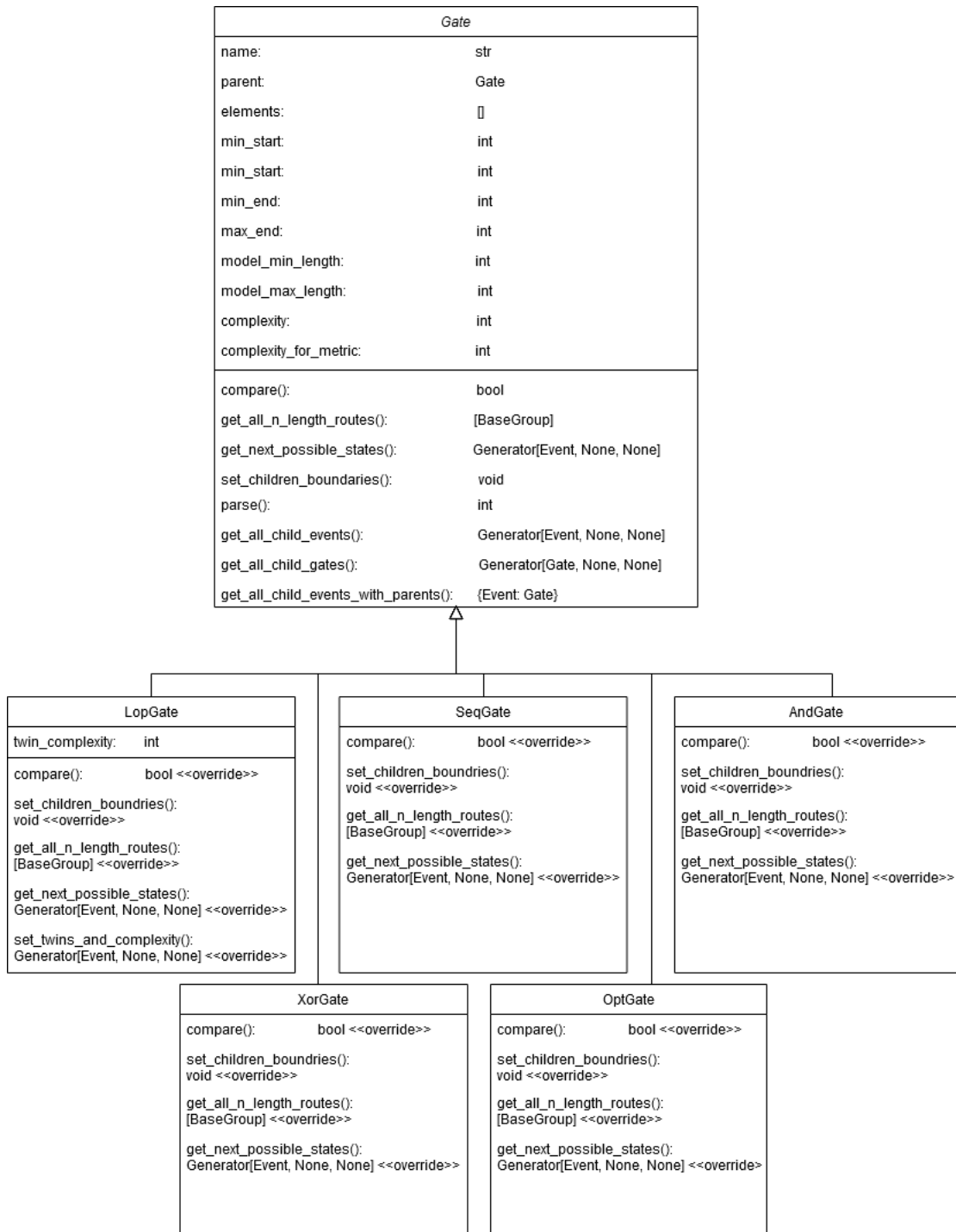
Podział na dwie klasy przydatne na różnym etapie procesowania:

Gate:

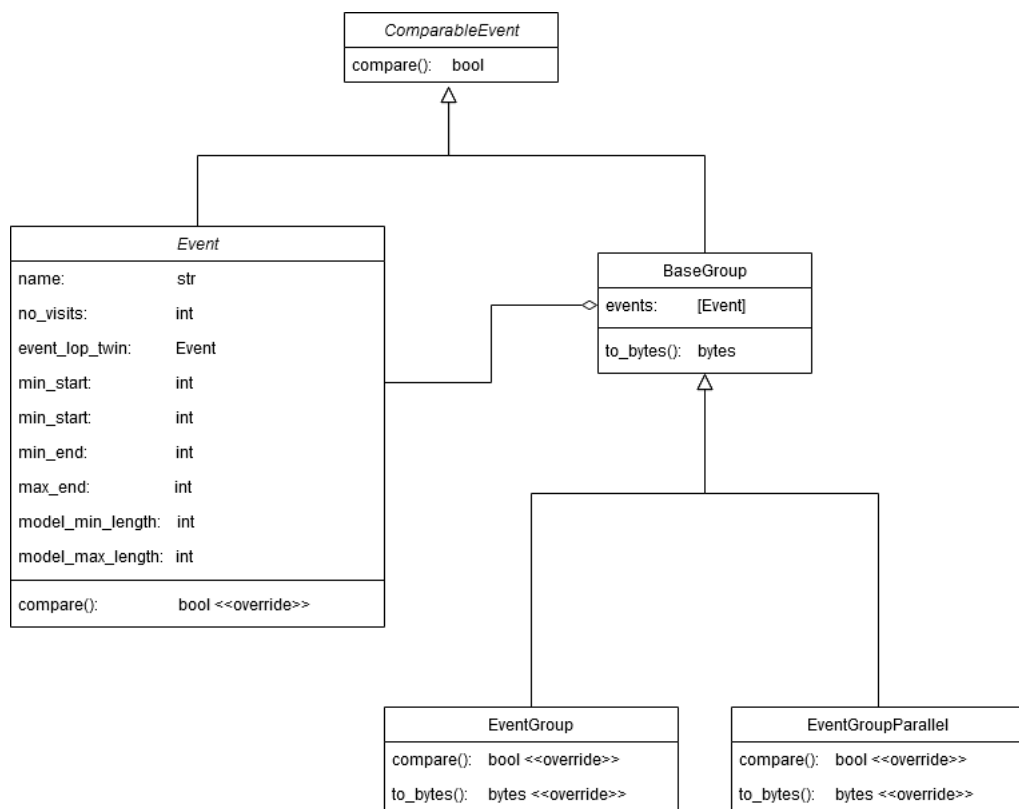
Event group:

3.4. Implementacja

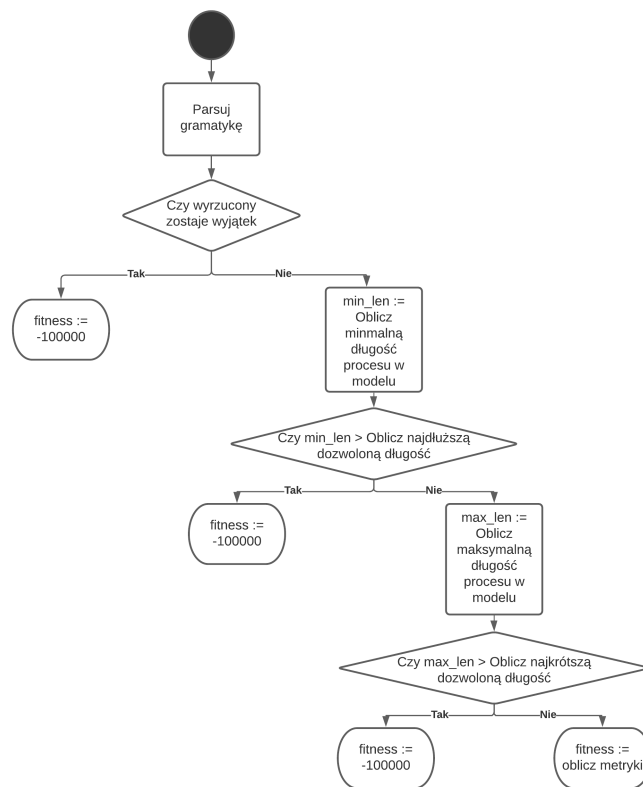
3.4.0.1. Ogólny schemat blokowy



Rys. 3.4. Gate UML



Rys. 3.5. Event UML



Rys. 3.6. Ogólny schemat blokowy

3.4.0.2. Parsowanie gramatyki

Parser pozwala na przetworzenie wyników uzyskanych na drodze ewolucji gramatycznej na postać, na której łatwiej będzie operować. Rezultaty uzyskane na drodze ewolucji 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.

3.4.0.3. Obliczanie metryk

Mając już sparsowany model musimy obliczyć metryki. Najbardziej problematyczną metryką do obliczenia jest dopasowanie. Obliczanie dopasowania można opisać następującymi krokami:

- Znalezienie ścieżek o długości n w modelu.
- Przerobienie ścieżek na postać zawierającą BaseGroup.
- Obliczenie dopasowania.

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

    for i w zakresie długość_wyrazenia:
        if wyrażenie[i] == "{":
            zdarzenie := Event(wyrazenia[i + 1])
            dodaj zdarzenie do aktualnie parsowanej bramki
            i += 2
        elif wyrażenie[i] == ")":
            return i+1
        elif i+4 < długość_wyrazenia:
            if wyrażenie[i:i + 3] == "seq" and (self.name == "seq" or self.name == "lop"):
                usuń_niepotrzebe_bramki
            else:
                if wyrażenie[i:i+2] == 'lo' and wyrażenie[i:i+3] != 'lop':

                    bramka := stwórz nową bramkę Gate typu zgodnego z wyrażeniem
                    i += 3
                    przeparsowane_znaki = bramka.parsuj(wyrazenie[i+4:])
                    dodaj zdarzenie do aktualnie parsowanej bramki
                    i += ilość_przeparsowanych_znaków
        else:
            wyrzucić wyjątek
```

Listing 3.2. Parser gramatyki

Metryką, która nie wymaga obliczenia dopasowania jest prostota, dlatego możemy ją obliczyć wcześniej co przy niskim wyniku pozwala na wstępne odrzucenie części rezultatów bez konieczności kosztownego obliczania dopasowania. Pozostałe metryki wymagają obliczenia dopasowania i są obliczane dla najlepiej dopasowanej gramatyki. Łatwo można zauważyć, że jeżeli zdarzenie znajduje się w logu, a nie znajduje się w modelu dopasowanie nie będzie dobre. Pozwala to odrzucić rezultaty, które nie przekraczają progu.

```

def oblicz_metryki(log_info, model, najkrótsza_dozwolona_długość,
                  najdłuższa_dozwolona_długość, cache) -> int:

    metryki['PROSTOTA'] := oblicz_metrykę_prostota(lista_zdarzeń_w_procesie),
        unikalne_zdarzenia_w_logu)
    if metryki['PROSTOTA'] < 2/3:
        return 0

    stosunek_wspólnych_zdarzeń_w_logu_i_modelu :=
        oblicz_stosunek_wspólnych_zdarzeń_w_logu_i_modelu()
    if stosunek_wspólnych_zdarzeń_w_logu_i_modelu <
        MINIMALNY_STOSUNUK_WSPÓLNYCH_ZDARZEŃ_W_LOGU_I_MODELU:
        return stosunek_wspólnych_zdarzeń_w_logu_i_modelu/10

    idealnie_dopasowane_logi := pusty_słownik
    skumulowany_błąd := 0

    for proces w log:
        najlepszy_błąd_lokalny, najlepiej_dopasowana_ścieżka, najlepszy_process :=
            oblicz_metryki_dla_jednego_procesu(proces, model, minimalna_długość,
            maksymalna_długość, cache)
        if jakikolwiek proces w najlepiej_dopasowanej_ścieżce nie znajduje się w modelu:
            value, best_aligned_process = oblicz_dopasowanie_bez_cache(best_event_group,
                list(process), dict())
            best_local_error = calculate_alignment_metric(value, oblicz_długość(proces),
                oblicz_długość(best_event_group))
            if najlepszy_błąd_lokalny == 0:
                idealnie_dopasowane_logi.dodaj() [tuple(best_aligned_process)] =
                    log_info.log[process]
            add_executions(model_events_list, best_aligned_process, log_info.log[process])

    metryki := oblicz_metryki
    najlepszy_wynik := oblicz_średnia_ważona_metryk
    return najlepszy_wynik

```

Listing 3.3. Obliczanie metryk

3.4.0.4. Obliczanie metryk dla jednego procesu

3.4.0.5. Wyszukiwanie w modelu procesów o określonej długości

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

3.4.0.6. Obliczanie dopasowania

Pomysł zaczerpnięty z algorytmu Needleman-Wunsch [5], który jest uogólnieniem odległości Levenshteina. Tworzymy macierz o wymiarach długość modelu i długość logu, w której obliczać będziemy rozwiązania. Rozwinięty o możliwość przeszukiwania modelu rekurencyjnie oraz o możliwość podawania listy równoległych zdarzeń.

3.4.0.7. Znajdowanie ścieżki w modelu

Potrzebne do obliczenia precyzji oraz generalizacji.

```

def oblicz_metryki_dla_jednego_procesu(proces, model, najkrótsza_dozwolona_długość,
                                       najdłuższa_dozwolona_długość, cache):
    długość_procesu := oblicz_długość(proces)
    n := długość_procesu
    i := 1
    minimalny_błąd_dopasowania := -(długość_procesu + model.model_min_length)
    while not (dolny_limit_osiagnięty and górny_limit_osiagnięty):
        if n >= min(oblicz_maksymalna_dozwolona_długość(długość_procesu),
                   długość_procesu - minimalny_błąd_dopasowania):
            górny_limit_osiagnięty := True
            n += (-i if i % 2 == 1 else i); i += 1
            continue
        if n <= max(oblicz_minimalna_dozwolona_długość(długość_procesu),
                   długość_procesu + minimalny_błąd_dopasowania):
            dolny_limit_osiagnięty := True
            n += (-i if i % 2 == 1 else i); i += 1
            continue
    if najkrótsza_dozwolona_długość <= n <= najdłuższa_dozwolona_długość:
        ustaw_najwcześniejsze_i_najpóźniejsze_wystąpienie_zdarzenia(model, n)
        ścieżki = model.znajdź_wszystkie_ścieżki_długości_n(n, proces)
        if ścieżki istnieją:
            for ścieżka in ścieżki:
                procent_wspólnych_zdarzeń := oblicz_procent_wspólnych_zdarzeń_
                                              w_modelu_i_logu(ścieżka, proces)
                if procent_wspólnych_zdarzeń >= 1 - TOLERANCJA:
                    dodaj_ścieżkę_do_listy_ścieżek_do_obliczenia
            posortowane_ścieżki := posortuj_listę_ścieżek_do_obliczenia
            for ścieżka in posortowane_ścieżki:
                if procent_wspólnych_zdarzeń <= 1 + minimalny_błąd_dopasowania /
                   długość_procesu:
                    break
            błąd_dopasowania, najlepsze_dopasowane_zdarzenia, jest_z_cache :=
                oblicz_najlepsze_dopasowanie_z_cache(ścieżka, proces, cache)
            if błąd_dopasowania > minimalny_błąd_dopasowania:
                minimalny_błąd_dopasowania := błąd_dopasowania
                najlepsze_dopasowane_zdarzenia := dopasowane_zdarzenia
                najlepsza_ścieżka := ścieżka
                jest_najlepszy_z_cache := jest_z_cache
            if błąd_dopasowania == 0:
                return minimalny_błąd_dopasowania, najlepsze_dopasowane_
                       zdarzenia, najlepsza_ścieżka, jest_najlepszy_z_cache
    n += (-i if i % 2 == 1 else i); i += 1
    return minimalny_błąd_dopasowania, najlepsze_dopasowane_zdarzenia,
        najlepsza_ścieżka, jest_najlepszy_z_cache

```

Listing 3.4. Obliczanie metryk dla jednego procesu

```
def znajdź_wszystkie_ścieżki_długości_n(n: int, proces) -> []:
    if n == 0:
        return []
    if self.model_max_length < n or n < self.model_min_length:
        return None

    min_lengths = self.get_children_min_length()
    max_lengths = self.get_children_max_length()
    global_list = []

    for elem in self.elements:
        local_list = []
        if isinstance(elem, Event):
            local_list.append(elem)
            min_lengths.pop(0)
            max_lengths.pop(0)
        else:
            lower_limit, upper_limit = self.get_goal_length_range(n, global_list, min_lengths,
                                                                    max_lengths)
            for i in range(lower_limit, upper_limit + 1):
                try:
                    child_all_n_length_routes = elem.get_all_n_length_routes(i, proces)
                except ValueError:
                    return None
                if child_all_n_length_routes is not None:
                    local_list.append(child_all_n_length_routes)

            if local_list:
                global_list.append(local_list)

    result = []
    if global_list:
        for elem in flatten_values(global_list):
            if self.check_length(n, elem):
                if n == 1:
                    # because always 1 elem list
                    result.append(elem[0])
                else:
                    self.check_valid_for_get_n_length(elem)
                    result.append(EventGroupParallel(elem))
    if result:
        return result
    else:
        return None
```

Listing 3.5. Wyszukiwanie procesów o długości n

```
def oblicz_dopasowanie(model, log):
    bład := {'DOPASOWANIE': 0, 'BRAK_DOPASOWANIA': -2, 'PRZERWA': -1}
    m = długość(model) + 1 # Macierz rozwiązań ilość wierszy.
    n = długość(log) + 1 # Macierz rozwiązań ilość kolumn.
    najlepiej_dopasowana_ścieżka := [None] * m
    macierz_rozwiazań := Zainicjalizuj macierz zerami
    # Wypełnij osie macierzy właściwymi wartościami
    for j in range(n):
        macierz_rozwiazań[0][j] := bład['PRZERWA'] * j

    for i in range(1, m):
        if should_go_recurrent(model[i-1]):
            macierz_rozwiazań[i], najlepiej_dopasowana_ścieżka_podmodelu[i] :=
                dopasowanie_rekurencyjne(macierz_rozwiazań[i - 1], model[i - 1],
                                         [x for x in odwrócone_substringi(log)], i)
        elif długość(model[i-1]) > 1:
            macierz_rozwiazań[i], najlepiej_dopasowana_ścieżka_podmodelu[i] :=
                dopasowanie_równoległe(macierz_rozwiazań[i - 1], model[i - 1],
                                       [x for x in odwrócone_substringi(log)], kara, i)
        else:
            macierz_rozwiazań[i][0] := macierz_rozwiazań[i-1][0] + kara['PRZERWA']
            dopasowanie(macierz_rozwiazań, model[i - 1], log, kara, i, n)

    ścieżka := znajdź_ścieżkę(macierz_rozwiazań, bład['PRZERWA'], model,
                             log, najlepiej_dopasowana_ścieżka_podmodelu)

    return macierz_rozwiazań[m-1], najlepiej_dopasowana_ścieżka
```

Listing 3.6. Obliczanie dopasowania


```

def znajdź_sciezke(macierz_rozwiazan, model, log, rozwiazania_podmodeli):
    sciezka = []
    while i != 0:
        długość_grupy_zdarzeń = długość(model[i - 1])
        if model_results_local[i] is not None:
            znaleziono_dopasowanie := False
            if macierz_rozwiazan[i][j] == macierz_rozwiazan[i - 1][j] + długość_grupy_zdarzeń *
                [model_result.append(None) for _ in range(długość_grupy_zdarzeń)]
            macierz_rozwiazan[i][j] := 0
            i -= 1
        else:
            for k in range(j):
                zdarzenia := get_not_none(model_results_local[i][k]
                    [długość(model_results_local[i][k]) - (j-k)], log)
                if macierz_rozwiazan[i][j] == macierz_rozwiazan[i - 1][k] +
                    (długość_grupy_zdarzeń + (j-k) - 2 * długość(processes)) * błąd['PRZERWA']
                [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(długość_grupy_zdarzeń - len(processes))]
                macierz_rozwiazan[i][j] = 0
                i -= 1; j = k
                znaleziono_dopasowanie = True
                break
            if not znaleziono_dopasowanie:
                if macierz_rozwiazan[i][j] == macierz_rozwiazan[i][j - 1] + błąd['PRZERWA']
                macierz_rozwiazan[i][j] = 0
                j -= 1
        else:
            if macierz_rozwiazan[i][j] == macierz_rozwiazan[i - 1][j] + kara:
                model_result.append(None)
                macierz_rozwiazan[i][j] := 0
                i -= 1
            elif macierz_rozwiazan[i][j] == macierz_rozwiazan[i][j - 1] + kara:
                macierz_rozwiazan[i][j] := 0
                j -= 1
            elif macierz_rozwiazan[i][j] == macierz_rozwiazan[i - 1][j - 1]:
                model_result.append(model[i-1])
                log = log.replace(model[i-1].name, "", 1)
                macierz_rozwiazan[i][j] := 0
                i -= 1; j -= 1
    return sciezka

```

Listing 3.7. Znajdowanie ścieżki w modelu

3.4.0.8. Cache

Wiele obliczeń się powtarza, dlatego żeby przyspieszyć działanie rezultaty są cachowane w dwóch miejscach: Cache genotypów, dostarczane przez Cache obliczeń dopasowanie, które jest najbardziej kosztownym obliczeniem. Wybrane metodę cachowania LRU.

3.5. Wybór parametrów algorytmu

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.