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

1.1. Zarys tematyki pracy

Zdefiniowanie kroków potrzebnych do osiągnięcia danego efektu jest konieczne do zrozumienia podejmowanych działań i wprowadzenia ewentualnych udoskonaleń. Z czasem biznes zdał sobie z tego sprawę i kierując się zasadą: „Jeżeli nie jesteś w stanie opisać czegoś jako proces, nie masz pojęcia, co robisz", firmy zaczęły podejmować próby uporządkowania i zamknięcia swoich działań w ramy, co doprowadziło do wzrostu popularności procesów biznesowych.

Identyfikacja i opis procesów biznesowy sprawia, że wszystkie operacje w firmie stają się przejrzyste i łatwiejsze do zrozumienia. Analiza procesów biznesowych może pozwolić na zwiększenie produktywności oraz redukcję kosztów. Procesy biznesowe mogą pozwolić na przewidywanie przyszłych zdarzeń na podstawie danych, znajdowanie wąskich gardeł, a także zmniejszają zależność firm od poszczególnych ludzi.

W związku z możliwością gromadzenia coraz większej ilości danych, a także chęcią ich wykorzystania oraz rosnącą popularnością analizy danych (*eng. data science*), biznes zdał sobie sprawę z możliwości wykorzystanie technologii informatycznych w kontekście procesów biznesowych. Zapoczątkowało to powstanie na pograniczu zarządzania procesami biznesowymi i metod informatycznych używanych do analizy danych, wśród wielu innych, dziedziny zwanej eksploracją procesów (*eng. process mining*).

1.2. Cele pracy

Celem pracy jest projekt i implementacja metody odkrywania procesów biznesowych przy użyciu programowania genetycznego. W pracy zbadano jak wybór metod programowania genetycznego, wybór gramatyki, a także parametrów programu wpływa na jakość rozwiązania. Zaprezentowano też przykłady użycia algorytmu do okrywania procesów biznesowych oraz porównano z innymi dostępnymi algorytmami. Ponadto w pracy zostały zbadana hipoteza czy rozwiązywania problemu najpierw dla prostych przypadków i wykorzystanie rozwiązań tego problemu może mieć korzystny wpływ na rozwiązanie bardziej skomplikowanego problemu.

1.3. Zawartość pracy

Praca zastała podzielona na cztery części. We wstępie teoretycznym zostały przybliżone zagadnienia potrzebne do zrozumienia pracy, takie jak procesy biznesowe, eksploracja procesów oraz ewolucja gramatyczna. W kolejnej części została przedstawiona gramatyka stworzona na potrzebny odkrywania procesów oraz projekt i implementacja algorytmu do wyszukiwania procesów genetycznych. Następnie zaprezentowane zostały wyniki działanie algorytmu dla przykładowych dzienników zdarzeń. Omówione zostało też jak na czas znajdowania rozwiązania oraz jego jakość wpływają przyjęte parametry algorytmu w szczególności wybór metryk oraz wagi z jakimi każda metryka powinna być brana pod uwagę.

2. Wstęp teoretyczny

2.1. Procesy biznesowe

2.1.1. Procesy biznesowe

W każdym dużym przedsiębiorstwie każdego dnia wykonywana jest ogromna ilość czynności koniecznych do funkcjonowania tej organizacji. Ludzie oraz systemy realizują rozmaite działania związane z różnymi, często niemającymi wiele wspólnego zadaniami jak, chociażby procesowanie płatności, składnie zamówień, wytwarzanie produktów czy ich transport. Przykłady te można mnożyć w zależności od sektora, w jakim obraca się dana firma. Im jest ona większa, tym trudniej jest osobom nią zarządzającym zrozumieć i opisać poszczególne czynności. W pewnym momencie, kiedy ilość różnych zadań rośnie do setek czy tysięcy, staje się to niemożliwe i potrzebny jest sposób na zebranie wiedzy o pojedynczych operacjach i zamknięcie ich w uporządkowaną strukturę. Stąd narodził się pomysł na wykorzystanie procesów biznesowych.

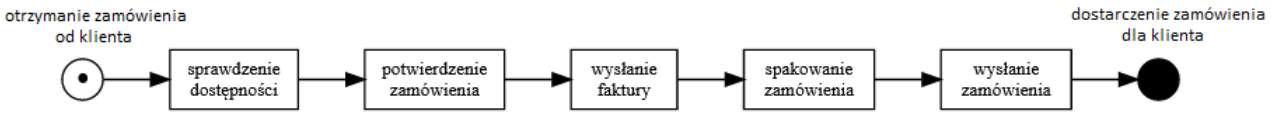
Procesy biznesowe opisują zbiór aktywność, które podejmuje grupa podmiotów w celu osiągnięcia celu biznesowego. W literaturze brakuje jednej ogólnie przyjętej definicji procesu biznesowego. W latach 90. XX wieku proponenci BPR, czyli Przeprojektowania procesów biznesowych (*eng. Business process re-engineering*) starali się sprecyzować pojęcie procesu biznesowego. W książce „Process Innovation: Reengineering Work through Information Technology" [1] określono termin ten jako „Ustrukturyzowany, mierzalny zbiór działań, których celem jest wytworzenie określonego produktu dla określonego klienta lub rynku". Autor położył nacisk na zbiór kroków prowadzących do celu, raczej niż na końcowy efekt. W dalszej części autor pisze „Proces jest zatem określonym uporządkowaniem czynności roboczych w czasie i przestrzeni, z początkiem i końcem oraz jasno określonymi wejściami i wyjściami: strukturą działania.". Inni pionierzy BPR Michael Hammer i James Champy zaproponowali podejście „Proces biznesowy to zbiór działań, który pobiera jeden lub więcej rodzajów danych wejściowych i tworzy wynik, który ma wartość dla klienta" [2]. Autorzy dają większą dowolność, co do definicji procesu, nie wspominając o konieczności jego logicznej organizacji czy mierzalności. Z kolei Jacobson zupełnie pomija konieczność zamknięcia procesu w jakiekolwiek ramy: „Zestaw czynności wewnętrznych wykonywanych w celu obsługi klienta" [3]. Nacisk na konieczność odniesienia procesów do wymiernych środków firmy widzimy w definicji: „Procesy biznesowe są aktywną częścią biznesu. Opisują funkcje

firmy i obejmują zasoby, które są używane, przekształcane lub wytwarzane. Proces biznesowy to abstrakcja, która pokazuje współpracę między zasobami i transformację zasobów w biznesie. Podkreśla, w jaki sposób wykonywana jest praca, zamiast opisywać produkty lub usługi wynikające z tego procesu." [4]. Szczególnie ważny jest tutaj fragment o transformacji zasobów, gdyż każe on rozumieć poszczególne aktywności w procesie jako powiązane ze sobą i kończące się namacalnymi rezultatami. Definicja „Proces biznesowy to seria kroków mających na celu wytworzenie produktu lub usługi. W wyniku niektórych procesów produkt lub usługa jest odbierana przez zewnętrznego klienta organizacji. Nazywamy je podstawowymi procesami. Inne procesy wytwarzają produkty, które są niewidoczne dla klienta zewnętrznego, ale są niezbędne do efektywnego zarządzania firmą. Nazywamy je procesami wsparcia" [5] wprowadza rozgraniczenie na podtypy procesów. Ważnym jest jednak, że nie jest koniecznością, aby rezultaty procesu były widoczne na zewnątrz organizacji. Warto też zaznaczyć, że procesy biznesowe nie dotyczą jednej osoby czy nawet działu, a raczej udział w nich bierze wiele ludzi, maszyn czy systemów z różnych działów połączonych celem dostarczenia wspólnej wartości biznesowej.

Powyższe definicji skupiają się na delikatnie odmiennych aspektach procesów biznesowych, nie zawsze szczegółowo wspominając o innych. Starając się usystematyzować powyższe sformułowania, chcąc zbudować bazę do dalszej analizy tematu, można przyjąć, że procesy biznesowe charakteryzują:

- Określony cel, którym jest wytworzenie wartości dla klienta zewnętrznego lub pośrednio firmy - klienta wewnętrznego. Jednak warto jeszcze raz zaznaczyć, że proces biznesowy skupia się na sposobie osiągnięcia celu, a nie opisie celu samego w sobie.
- Dyskretny, jasno zdefiniowany i identyfikowalny zbiór aktywności.
- Jasno określony początek - wejście i koniec - wyjście.
- Zależność przyczynowo-skutkowa pomiędzy kolejnymi procesami.

Żeby lepiej zilustrować, czym jest proces biznesowy, poniżej znajduje się prosty przykład często spotykanego procesu. Oczywiście, prawdziwy proces będzie składał się z o wiele większej liczby aktywności.



Rys. 2.1. Przykład prostego procesu

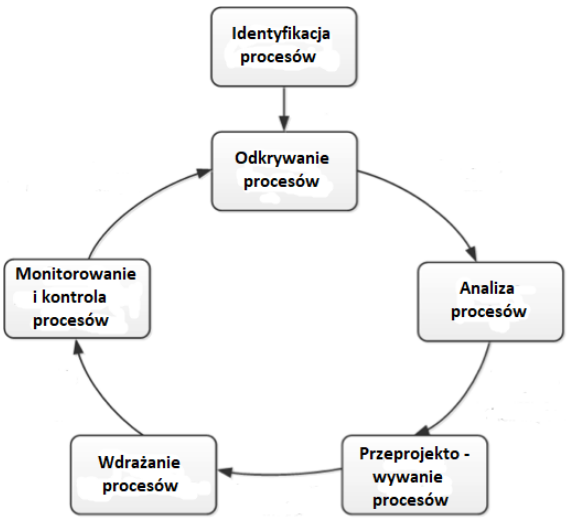
Zauważmy, że mamy jasno zdefiniowany wejście - otrzymanie zamówienia od klienta oraz wyjście, kiedy dostarczamy oczekiwaną wartość dla klienta, a całość składa się z serii tworzących logiczną całość aktywności. Aktywności są konkretnie zdefiniowane. Standardem jest definiowanie aktywności w formie równoważników zdań.

2.1.2. Zarządzanie procesami biznesowymi

Zdefiniowanie proces biznesowego otwiera wiele możliwości analizy działań przedsiębiorstwa i wskutek tego wprowadzanie usprawnień. Dziedziną, która się tym zajmuje, jest zarządzanie procesami biznesowymi (*eng. Business process modeling*) zwane w skrócie BPM. Sercem jest proces, a samo BPM jest dyscypliną używającą różnych metod, technik i sposobów w celu projektowania, wprowadzania w życie, zarządzania i analizy procesów biznesowych [6].

Celem stosowania metod zarządzanie procesami biznesowym jest udoskonalanie procesów w danej organizacji biznesowej. Udoskonalanie może być rozumiane w różnoraki sposób w zależności od kierunku rozwoju firmy. Może to być na przykład redukcja czasu, kosztów, czy dostarczanie lepszego produktu końcowego. Ważne jest, aby było to podejście całościowe i odnosiło się do całego zbioru aktywności w ramach danego procesu. Usprawnianie pojedynczej aktywności to nie BPM. Patrząc na przykład powyżej, jeśli wprowadzilibyśmy usprawnienia w ramach wysyłania faktury, robiąc to elektronicznie zamiast tradycyjną pocztą, mimo że taka zmiana przyniosłaby poprawę wydajności, nie mielibyśmy do czynienia z zarządzaniem procesami biznesowymi. O BPM moglibyśmy mówić, gdybyśmy znaleźli sposób, żeby przeprojektować cały proces tak, żeby wysyłanie faktury nie było potrzebne lub odwrotnie, jeśli dodalibyśmy nowe aktywności, która usprawniłaby proces jako całość czy nawet zmieli kolejności zadań w procesie, gdyż zmiany w ramach poszczególnych, jednostkowych aktywności nie są konieczne, żeby ulepszyć proces jako całość [7].

Zarządzanie procesami biznesowymi jest zbiorem praktyk, działań mających na celu udoskonalanie procesów. Trzeba więc rozumieć BPM jako pojęcie abstrakcyjne, jednak szczególnie w dzisiejszym świecie, zarządzanie procesów biznesowych nie może się obyć bez wsparcia ze strony oprogramowania czy technik znanych z różnych dziedzin informatyki [8]. Na lepsze zrozumienie czym zajmuje się zarządzanie procesami biznesowymi oraz w jaki sposób możemy zastosować informatykę, a w szczególności eksplorację procesów w tej dziedzinie, może pozwolić definicja cyklu życia procesu biznesowego.



Rys. 2.2. Cykl życia procesu biznesowego

Cykl życia procesu biznesowego (*eng. Business process lifecycle*) przedstawiono na rys. 2.3 [9]. Jest to zbiór kroków niezbędnych do skutecznego zarządzania procesami biznesowymi. W celu dostosowania do zmieniającej się rzeczywistości poszczególne kroki powinny być co pewien czas powtarzane.

Konieczność powtarzania elementów cyklu życia procesu biznesowego sygnalizuje przewagę komputerów i algorytmów nad wykonywaniem tych operacji przez człowieka. Metody informatyczne są stosowane, na każdym z wymienionych etapów. W szczególności dane zebrane w wyniku monitorowania procesów dają nam możliwość zastosowania metod z zakresu eksploracji procesów (sekcja 2.2). Praca skupia się w głównej mierze na odkrywaniu procesów, czyli znajdowaniu istniejących już procesów na podstawie realnych danych. Należy zaznaczyć, że identyfikacja polega na ogólnym rozpoznaniu i nazwaniu zachodzących procesów, podczas gdy odkrywanie jest bardziej szczegółowe, a w jego wyniku otrzymujemy dokładny model.

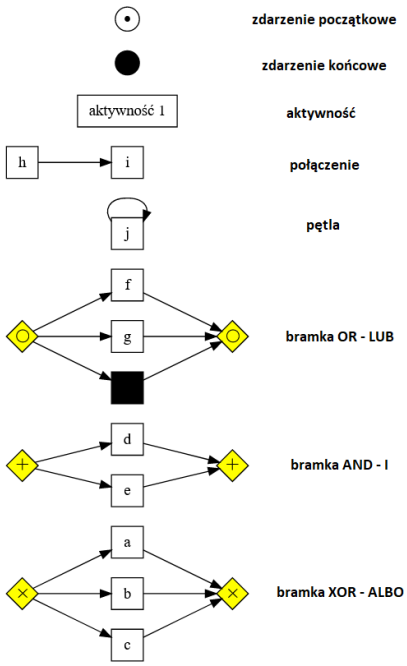
2.2. Eksploracja procesów

2.2.1. Modelowanie procesów biznesowych

Na rys. 2.1 przedstawiono przykład uproszczonego procesu biznesowego. Łatwo sobie wyobrazić, że proces ten w rzeczywistości może być znacznie bardziej skomplikowany. Część aktywności może być wykonywana równolegle, niektóre zdarzenia w ogóle nie zaistnieją lub będą występować kilkukrotnie w ramach jednego procesu.

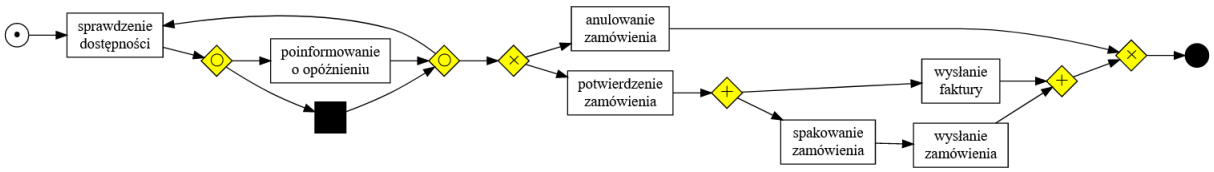
W sytuacji, w której zamówiony przez klienta towar będzie niedostępny, logiczne wydaje się poinformowanie go o opóźnieniu oraz danie mu możliwości anulowanie zamówienia lub jego kontynuacja i ponowne sprawdzenie dostępności. Ponadto, czynności takie jak wysłanie faktury oraz spakowanie i wysłanie zamówienia mogą być wykonane w dowolnej kolejności czy nawet jednocześnie przez dwie różne osoby. Proces staje się bardziej skomplikowany i konieczna do stworzenia jego modelu jest bardziej złożona notacja niż użyta do przedstawienia prostego procesu. Istnieje wiele notacji do modelowania procesów biznesowych, wśród nich można wymienić schematy blokowe, diagramy aktywności UML, łańcuchy procesu sterowanego zdarzeniami (*eng. Event-driven Process Chains*), sieci Petriego [10]. Obecnie najpopularniejszą notacją używaną do opisu procesów biznesowych jest Business Process Model and Notation, w skrócie BPMN [11]. Daje ona możliwość opisania w jednoznaczny sposób skomplikowanych procesów czy stworzenia diagramów współdziałania procesów, jednocześnie pozostając łatwą do zrozumienia.

Na grafice poniżej przedstawiono notację opartą o elementy BPMN, używaną w dalszej części pracy. Składają się na nią zdarzenia początkowe i końcowe, połączenia, bramki logiczne oraz aktywności. Czarnym kwadratem oznaczono sytuację, w której żadna aktywność nie jest wykonywana, możliwe tylko w bramce LUB.



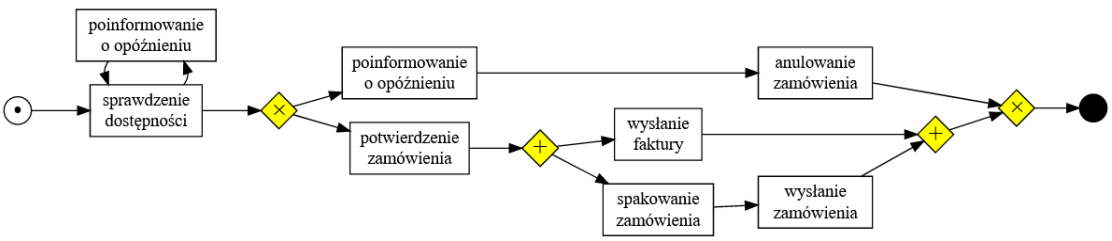
Rys. 2.3. Elementy BPMN

Korzystając z tej notacji, można przedstawić opisany wcześniej proces. Na rys. 2.4 widać model po modyfikacjach.



Rys. 2.4. Rozbudowany model procesu - przykład 1

Możliwe jest teraz poinformowanie klienta o opóźnieniu, a następnie anulowanie zamówienia lub powtórne sprawdzenie dostępności. Model ten jednak nie jest wystarczająco precyzyjny i pozwala na potwierdzenie zamówienia po informacji o jego opóźnieniu, a bez uprzedniego ponownego sprawdzenia dostępności. Można zaproponować inny model (rys. 2.5), który rozwiązuje powyższe problemy, jednak aktywność - poinformowanie o opóźnieniu - występuje w nim dwukrotnie, co jest niepożądane i pogarsza jego czytelność.



Rys. 2.5. Rozbudowany model procesu - przykład 2

Co więcej, w pewnych przypadkach klient może mieć możliwość rezygnacji z zamówienia bez ówczesnego informowania go o opóźnieniu, a z czego nie zdawano sobie sprawy, wtedy konieczne może być stworzenie zupełnie innego modelu. Aby radzić sobie z tymi problemami, powstał szereg zestawów wytycznych, którymi warto się kierować, modelując procesy biznesowe. Wśród takich zasad można wymienić: zminimalizuj liczbę elementów w modelu, zminimalizuj liczbę ścieżek w modelu, używaj jednego zdarzenia początkowego i jednego końcowego, unikaj bramek LUB - OR , zdekomponuj model zawierający więcej niż 50 elementów [12].

Modelowania procesów biznesowych jest próbą stworzenia uproszonej wersji rzeczywistości na podstawie przewidywań i założeń. Modele dają abstrakcję, użyteczne przybliżenie rzeczywistości, jednak należy pamiętać, że „Wszystkie modele są błędne" i rzeczywisty proces najprawdopodobniej będzie różnił się od nawet najlepszego modelu.

2.2.2. Eksploracja procesów

W dzisiejszych czasach standardem jest, że organizacje biznesowe korzystają z systemów informatycznych, takich jak, chociażby systemy ERP czy CRM, wspierających ich działalność. Systemy te rejestrują dane o procesach, które wspierają. Dane te mogą być później analizowane i wykorzystane do wprowadzenia usprawnień w działaniu firmy.

Tradycyjne metody są wolne, kosztowne i narażone na błędy ludzkie, a konieczność ich ciągłego powtarzania, połączona z wszechobecnym w biznesie trendem automatyzacji sprawiają, że eksploracja procesów zyskuje na znaczeniu [13]. Ważna jest możliwość szybkiej adaptacji do zmian, a automatyzacja odkrywania procesów biznesowych pozwala na wykonywanie powtarzalnych zadań, eliminując przy tym błędy, co idealnie wpisuje się w ten trend.

Jest to szeroko pojęta dziedzina, która zawiera różne aplikacje inteligencji obliczeniowej, uczenia maszynowego i eksploracji danych do modelowania i analizy procesów. Jest wartościowym dodatkiem do innych metod eksploracji danych, gdyż zamiast skupiać się pojedynczym rezultacie i tworzyć dotyczące jego predykcje, celem jest zrozumienie całego procesu i akcji, które prowadzą do końcowego wyniku. Jest to trudniejsze, ale wyjątkowo cenne z punktu widzenia biznesowego, gdyż jakakolwiek zmiana w trakcie procesu może sprawić, że przewidywania będą kompletnie trafione, a zrozumienie całego procesu pozwala na pełniejszy obraz i łatwiejsze dostosowywanie się do zmian.

Ponadto, procesy biznesowe są zazwyczaj domeną analityków i menadżerów, którzy nie zawsze podchodzą do tematu ich analizy w sposób ścisły i mający odniesie w faktach, często opierając się na własnych przeczuciach czy doświadczeniach, wprowadzając czynnik ludzki, który może być przyczyną błędów. Metoda na stworzenie pomostu między metodami informatycznymi a biznesem i stworzenie możliwości na ścisłe, powtarzalne i sprawdzalne analizowanie procesów jest więc nad wyraz cenna. Eksploracja procesów biznesowych oparta jest bowiem na danych i nie ma w niej dużo miejsca na przypuszczenia i domysły.

Podsumowując, eksploracja procesów są to techniki, narzędzia oraz metody odkrywania, monitorowania i usprawniania rzeczywistych procesów poprzez wiedzę wyodrębnioną z dzienników zdarzeń powszechnie dostępnych w systemach informatycznych [14][15]. Wyróżnia się 3 podkategorie:

- automatyczne odkrywanie procesów
- sprawdzanie zgodności (*eng. conformance checking*)
- udoskonalanie procesu (*eng. performance mining*)

2.2.3. Dzienniki zdarzeń

Danymi wejściowymi dla algorytmów z dziedziny eksploracji procesów są dzienniki zdarzeń, zwane często logami.

nr przypadku	aktywność	data	osoba wykonująca	zakładany czas wykonania
1	zgłoszenie problemu - a	2021.02.03 20:29:38	tester	6.5 dnia
1	programowanie (development) - b	2021.02.04 12:31:25	programista 1	6.5 dnia
2	zgłoszenie problemu - a	2021.02.05 19:13:32	klient	5.5 dnia
2	analiza - c	2021.02.06 02:43:09	analityk	5.5 dnia
2	programowanie (development) - b	2021.02.07 01:37:13	programista 2	5.5 dnia
2	testowanie - d	2021.02.08 12:43:45	tester	5.5 dnia
3	zgłoszenie problemu - a	2021.02.09 15:39:42	tester	4.5 dnia
3	development - b	2021.02.10 15:36:21	programista 1	4.5 dnia
1	analiza - c	2021.02.11 12:31:43	analityk	6.5 dnia
1	programowanie (development) - b	2021.02.12 00:01:54	programista 2	6.5 dnia
1	testowanie - d	2021.02.13 21:35:39	tester	6.5 dnia
4	zgłoszenie problemu - a	2021.02.14 09:23:59	tester	3.5 dnia
5	zgłoszenie problemu - a	2021.02.15 16:37:13	analityk	2.5 dnia
3	analiza - c	2021.02.16 02:29:56	analityk	4.5 dnia
3	programowanie (development) - b	2021.02.17 09:48:51	programista 1	4.5 dnia
3	testowanie - d	2021.02.18 20:50:28	tester	4.5 dnia
4	analiza - c	2021.02.19 15:48:37	analityk	3.5 dnia
4	programowanie (development) - b	2021.02.20 21:29:16	programista 1	3.5 dnia
4	sprawdzenie kodu (review) - e	2021.02.21 04:22:30	programista 2	3.5 dnia
5	uznanie problemu za rozwiązany - f	2021.02.22 06:28:29	programista 2	2.5 dnia
5	testowanie - d	2021.02.23 08:36:07	tester	2.5 dnia
4	testowanie - d	2021.02.24 21:17:54	tester	3.5 dnia

Rys. 2.6. Przykład dziennika zdarzeń

Przyjmuje się, że aby mówić o dzienniku zdarzeń powinien on zawierać 3 informacje: numer przypadku, czyli unikalny identyfikator zbioru aktywności, nazwę poszczególnych aktywności oraz datę jej wykonania - ważną tylko w kontekście kolejności wykonywania pojedynczych aktywności. Ponadto może on zawiera inne zbędne w kontekście odkrywania procesów biznesowych dodatkowa informacje, takie jak: podmiot wykonującym daną aktywność, miejsce, koszt czy aktualny postęp wykonania. Oczywiście te pozostałe dane mogą być wykorzystywane w kolejnych etapach analizy i usprawniania procesu.

Mając do dyspozycji te 3 informacje - poszczególne przypadki, aktywności na nie się składające oraz ich kolejność, zliczamy, jak często poszczególne aktywności występują w danej kolejności. Każdy taki przypadek zwany jest wariantem procesu. Oprócz tego musimy wiedzieć, jak często dany wariant wystąpił.

nr wariantu	ilość wystąpień	kolejność aktywności
1	2	a,b,c,b,d
2	1	a,c,b,d
3	1	a,c,b,e,d
4	1	a,f,d

Rys. 2.7. Przykład wariantów procesu

Dla poprawy czytelności aktywności często reprezentowane są jako symbole, np. kolejne litery alfabetu, zamiast pełnej nazwy.

2.2.4. Automatyczne odkrywanie procesów biznesowych

Automatyczne odkrywanie procesów biznesowych jest podgrupą i obejmuję techniki przekształcania danych w procesy. Ważne, że proces już istnieje, a my tylko go odkrywamy. Wejściem jest dziennik zdarzeń, a wyjściem jest mapa lub model procesu.

Procesy zaprojektowane nie zawsze są realizowane w praktyce. Ważne jest, żeby proces był oparty na analizie prawdziwych danych, a nie spekulacjach i założeniach. Pozwala na znajdowanie procesu takim, jaki jest, a nie takim, jakim chciano by, żeby był.

Rys. 2.8. Proces rzeczywisty i pierwotnie zakładany

Celem automatycznego odkrywania procesów biznesowych jest zaprojektowanie funkcji - algorytmu, która przekształci dane z dziennika zdarzeń w model procesu [16]. Istnieje wiele algorytmów do odkrywania procesów biznesowych. Wśród najpopularniejszych można wymienić:

- Alpha algorithm [17]
- The ILP Miner [18]
- Heuristic Miner [19]
- Multi-phase Miner [20]
- Inductive Miner [21]

Istnieją 4 powszechnie używane kryteria do określania jakości otrzymanego model. Są to:

- odwzorowanie (*eng. fitness*) - zgodność modelu z dziennikiem zdarzeń
- prostota (*eng. simplicity*) - złożoność i łatwość zrozumienia modelu.
- precyzja (*eng. precision*) - brak zachowań niezwiązanych z logiem, a możliwych w modelu
- generalizacja (*eng. genralization*) - odzwierciedlenie w modelu prawdopodobnych aktywności, mimo że nie znajdują się one w logu.

Koniecznym jest znalezienie balansu między nimi, gdyż często starając się poprawiać model pod kątem jednego kryterium, pogorszy się on pod względem innych. Powstało wiele metryk przedstawiających te kryteria za pomocą wzorów matematycznych [22] [23]. Bardziej szczegółowo wybór metryk omówiono w sekcji 2.4.

Wśród problemów dotykających istniejące algorytmy można wymienić problem ze znajdowaniem aktywności zachodzących równolegle, brak możliwości pomijania aktywności czy reprezentowania duplikatów, nieradzenie sobie z zakłóceniami w logu, tworzenie zbyt skomplikowanych modeli, czy trudność z odwzorowaniem niektórych zachowań. Modele stworzone mogą nie być spójne strukturalnie [24] [25], przez co rozumie się takie modele, w których istnieje taka aktywność, z której nie możemy osiągnąć zdarzenia końcowego lub nie może być ona w żaden sposób osiągnięta ze zdarzenia początkowego. Metody te zazwyczaj oparte są na grafach bezpośrednich następstw (*eng. directly follows graphs*), przez co problemem może być sytuacja, kiedy log jest niekompletny.

Zastosowanie algorytmy genetyczne do automatycznego odkrywania procesów biznesowych może być odpowiedzią na te problemy. Takie podejście pozwala na wyeliminowanie części problemów często dotyczących innych metod. Najważniejszą jednak przewagą algorytmów genetycznych jest pełna dowolność w kwestii generowania modelu pod kątem metryk zdefiniowanych przez użytkownika. Często znalezienie dobrze dopasowanego do logu modelu może być okupione stworzeniem bardzo skomplikowanego modelu - prostota, pozwalającego na wiele zachowań nieobecnych w logu - precyzja. Algorytm ewolucyjny pozwala na dowolnie dużą możliwość manipulacji parametrami, żeby znaleźć oczekiwany balans między wszystkimi metrykami. Możliwe jest też stworzenie nowych własnych metryk, gdyż są one niezależne od samego algorytmu ewolucyjnego. Algorytmy klasyczne mają problem z uzyskanie dobrych rezultatów dla wszystkich metryk i nie mają możliwości zmiany parametrów startowych.

2.3. Ewolucja gramatyczna

2.3.1. Algorytmy ewolucyjne

Algorytmy ewolucyjne [26] są inspirowaną selekcją naturalną metaheurystyką, która używa znanych z ewolucji biologicznej operacji jak mutacja, selekcja czy krzyżowanie do rozwiązywania problemów wyszukiwania i optymizacji. Ich ideą jest zaproponowanie metody przeszukiwania przestrzeni losowych rozwiązań w celu wyszukania najlepszych z nich.

Algorytmy ewolucyjne znajdują zastosowanie w problemach, dla których nie jest konieczna gwarancja znalezienia najlepszego rozwiązania. Cechami wyróżniającymi algorytmy genetyczne na tle innych algorytmów uczenia maszynowego jest istnienie puli rozwiązań zamiast jednego rozwiązywania, co umożliwia szersze przeszukiwanie przestrzeni rozwiązań oraz nieograniczoną i łatwą w zaimplementowaniu paralelizację. Algorytmy te są znacznie bardziej nastawione na globalne eksplorowanie nowych rozwiązań, zamiast szybkie osiągnięcie celu. Z tego względu dobrze nadają się do problemów gdzie istnieje dużo ekstremów, a przestrzeń poszukiwań jest duża. Jako przeciwieństwo, metody oparte na gradiencie w najprostszej znajdują tylko lokalne ekstrema, nawet po modyfikacja takich jak na przykład simulated anneling wciąż nie ma populacji i tak szerokiego przeszukiwania przestrzeni rozwiązań.

Sposób działania algorytmów genetyczny polega na stworzeniu populacji losowych rozwiązań zwanych genotypami lub chromosomami, które kodowane są za pomocą genów reprezentowany przez bity, liczby lub znaki i zapisywane w strukturze łatwo przetwarzalnej przez komputer. Najczęściej jest to lista jednowymiarowa liczb całkowitych. Fenotyp natomiast jest reprezentacją utożsamianą z docelowym programem lub modelem. Genotyp może być równoznaczny z fenotypem, jednak poza prostym przykładami, zazwyczaj są to oddzielne reprezentacje i geny mapowane są na odpowiadające wartości w fenotypie, zwane allelami. Fenotyp jest postacią dla której możliwe jest obliczanie funkcji dopasowania (*eng. fitness function*) pozwalająca ocenić jak dobre jest wygenerowane rozwiązanie. jeśli wciąż nie znaleziono satysfakcjonującego rozwiązania, generowana jest nową populację poprzez mutuje lub krzyżowania głównie choć nie tylko najlepszych chromosomów. Warto zauważyć, że krzyżowanie przeszukuje przestrzeń rozwiązań globalnie, podczas gdy mutacja odpowiada za lokalne wyszukiwanie. Proces ten jest powtarzany do momentu otrzymania satysfakcjonującego rozwiązania. Ten proces jest powtarzany dla każdego elementu populacji. Po sklasyfikowaniu rozwiązań. Po zastosowaniu tych operatorów utworzona zostaje nowa populacja, dla której cały proces jest powtarzany.

2.3.2. Szczegółowe omówienie operatorów i działania algorytmów ewolucyjnych

Tworzenie populacja jest pierwszym krokiem algorytmu ewolucyjnego. Może ono odbywać się kompletnie losowe lub do tworzenia nowych osobników może zostać użyta odpowiednia heurystka.

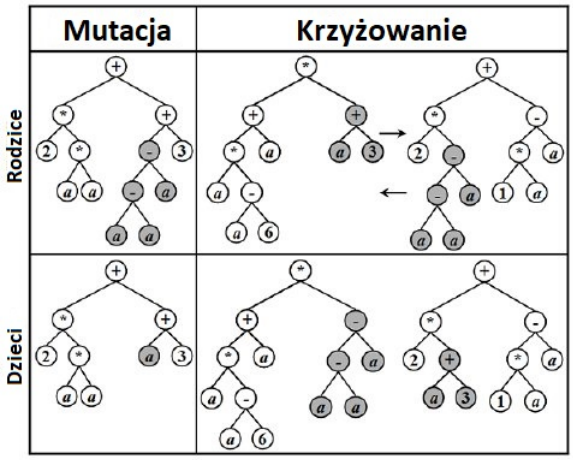
Kolejny krok selekcja sprawia, że przeszukiwanie przestrzeni rozwiązań nie jest kompletnie losowe. Wybierając najlepszej rozwiązanie można skorzystać z metod takich jak:

- Selekcja proporcjonalna - wybieramy losowo rozwiązania z puli wszystkich rozwiązań z warunkiem, że rozwiązania z największą wartością metryk mają największą szansę na bycie zachowanymi w populacji. Jest to najpopularniejsza metoda selekcji i najczęściej umożliwiająca najszybsze znalezienie rozwiązania. Pozwala na elityzm, czyli zachowanie części najlepszych genotypów w przyszłej populacji.

- Selekcja turniejowa - wybieramy podzbiór ze zbioru rozwiązań i zachowujemy w przyszłej najlepsze rozwiązanie z tego podzbioru. Rozwiązanie to pozwala na duży wpływ na presję genetyczną

- zwiększając wielkość podzbioru ograniczamy szansę na wybór z niską wartością metryk. Jest to także metoda, która łatwe zrównoleglenie.

Następnie stosowane są operatory krzyżowania i mutacji. Krzyżowanie jest zamianą meteriału genetycznego, czyli częsci rozwiązania pomiędzy dwoma osobnikami w poulacji tworząc również dwa zmienione osobniku. Mutacja natomist zachodzi w obrębie jednego osobnika.



Rys. 2.9. Mutacja i krzyżowanie

Operatory te nie muszą i zazwyczaj nie są stosowane do każdego osobnika w populacji, a za to jak często powinny być stosowane ustalane jest za pomocą odpowiedniego parametru. Mutacja może być stosowana więcej niż dla danego osobnika w danej generacji

Najczęściej używane operatory krzyżowania to:

- Krzyżowanie punktowe - spośród dwóch genotypów losowo wybieramy jeden punkt, następnie tworzymy dwa nowe genotypy pierwszy z chromosomów na prawo od punktu w pierwszym genotypie i na lewo w genotypie drugim oraz drugi z dwóch pozostałych.

- Krzyżowanie dwupunktowe - spośród dwóch genotypów losowo wybieramy dwa punkty, następnie część pomiędzy tymi punktami jest zamieniana pomiędzy genotypami.

- Krzyżowanie n-punktowe - uogólnienie powyższych krzyżowań dla n punktów.

- Krzyżowanie zamiana w drzewie - genotyp może być reprezentowany jako drzewo, w tej metodzie zamieniamy ze sobą dwa poddrzewa, tworzone są tylko prawidłowe rozwiązania, jednak jest to metoda wymagająca większej ilości obliczeń.

Natomiast operatory mutacji to:

- Mutacja punkowa - dowolna wartość w tablicy zostaje zmieniona na inną losową wartość. Pozostałe produkcje pozostają niezmienione.

- Mutacja zamiana w drzewie - genotyp może być reprezentowany jako drzewo, w tej metodzie tworzone jest nowe poddrzewo, przy tej metodzie tworzone są tylko prawidłowe rozwiązania, jednak jest to metoda wymagająca większej ilości obliczeń.

Żeby uniknąć sytuacji, w której najlepsze rozwiązania zostaną zmodyfikowane możliwe jest zastosowanie elityzmu, który pozwala na zachowanie w kolejnej generacji części najlepszych osobników w populacji niezależnie id wyniku selekcji.

Ostatnim krokiem jest zastąpienie osobników w poprzedniej populacji przez nowych powstałych na skutek zastosowania wcześniej wspomnianych operatorów. Istnieją dwa podejścia, sugerujące zastępowanie całej populacji lub tylko kilku osobników [27].

2.3.3. Ewolucja gramatyczna

Optymalny model procesu biznesowego może mieć różną długość, dlatego chcąc znaleźć taki model potrzebujemy metody, która pozwoli na generowanie rozwiązań o zmiennej długości. Tradycyjne algorytmy genetyczne operują na stałej strukturze i mogą być użyte na przykład, żeby dobrać odpowiednie parametry do istniejącego modelu. W wielu problemach jednak chcemy rozwiązanie o nieznanym rozmiarze, dlatego obecnie najpopularniejsza metoda to programowanie genetyczne [28][29], które pozwala na generowanie rozwiązań o różnym rozmiarze, dzięki ewolucji całej struktury fenotyp, najczęściej reprezentowane jako drzewo.

Oparta na programowaniu genetycznym jest ewolucja genetyczna [30]. Korzysta ona ze standardowych metod ewolucji genetycznej jednak ewoluuje gramatykę w celu znalezienia programu, który najlepiej rozwiązuje problem. Gramatyka najczęściej zapisana jako BNF (sekcja 2.3.4). Rekursywne produkcje pozwalają na tworzenie rozwiązań o różnej długości.

2.3.4. BNF

Gramatyka jest zbiorem zasad opisujących budowę języka. Opisuje syntaktykę języka, czyli sposób łączenia poszczególnych symboli, nie mówiąc nic o znaczeniu poszczególnych słów języka. Odnosi się to także do języków formalnych np. języków programowania. Do opisu takich języków służą gramatyki formalne. Na gramatykę formalną składa się z czwórka G = (T,N,P,S), gdzie

- T - skończony zbiór symboli terminalnych, czyli stałych, symboli które nie mogą być zastąpione innym ani podzielone na mniejsze
- N - skończony zbiór symboli nieterminalnych, czyli zmiennych, czyli symboli które są modyfikowane przy tworzeniu języka
- P - skończony zbiór produkcji, czyli zasad, przekształceń postaci (N∪T)*N(N∪T)* -> (N∪T)*, czyli przynajmniej jednego symbolu nieterminalnego w dowolny zbiór symboli terminalnych i nieterminalnych.

- S - symbol startowy, gdzie S $\in$ N

W informatyce szczególnie szeroko stosowane są gramatyki bezkontekstowe. Pozwalają one na pokazanie w jaki sposób rekurencyjnie tworzony jest język, co jest potrzebne, żeby zrozumieć znaczenie programu. Wiele teorii co pomaga uniknąć wieloznaczności na etapie parsowania. Gramatyka bezkontekstowa jest to gramatyka, w której wszystkie produkcje mają postać:

$$A \rightarrow \alpha,$$

gdzie A jest to pojedynczy symbol nieterminalny, a α to dowolny zbiór symboli terminalnych i nieterminalnych.

Notacja Backusa-Naura (*eng. Backus-Naur Form*)[31][32][33] jest najpopularniejszą notacją używaną do kodowania gramatyk bezkontekstowych. Symbole, które składają się na BNF to:

- ::= - produkcja
- | - lub
- <> - symbole nieterminalne

Używając tych symboli możemy opisywać składnie języka w następujący sposób:

<Nieterminalny1> ::= <Nieterminalny2>Terminany1 | <Nieterminalny3> | Terminany1

Oczywiście możliwe jest też definiowanie rekurencyjnie, co pozwala na tworzenie skomplikowanego języka za pomocą prostych zasad:

<Nieterminalny1> ::= <Nieterminalny2> | <Nieterminalny2><Nieterminalny1>

2.3.5. Tworzenie gramatyki pod kątem ewolucji

Dla każdego problemu można stworzyć nieskończoną ilość poprawnych gramatyk, jednak nie wszystkie z nich będą właściwe do zastosowania w algorytmie ewolucji gramatycznej. Chcąc stworzyć gramatykę, która umożliwi najwydajniejsze rozwiązanie problemu można przyjąć kilka wytycznych [34]:

- Każda rekursywna produkcja powinna mieć co najmniej tyle samo produkcji nierekursywnych inaczej biorąc pod uwagę, że genotyp jest mapowany na fenotyp metodą pseudolosową, prawdopodobieństwo otrzymania rozwiązania, które musi się składać tylko symboli terminalnych będzie zbyt niskie.
- Tworząc gramatykę pod kątem wykorzystania jej w procesie ewolucji ważne jest, żeby ilość produkcji jak najlepiej odzwierciedlała jak często chcemy uzyskać dane rozwiązanie.

- Stosując mutacje oraz krzyżowanie punktowe lub n-punktowe, które podmienia gen w genotypie może dojść do sytuacji, w której dany gen po podmianie zostanie zmapowany na produkcje o innej ilości symboli nieterminalnych przez co kolejne geny, jako że mamy wyprowadzenie lewostronne, będą mapowania niezgodnie z pierwotnym sensem. Żeby to osiągnąć należy ograniczyć ilość symboli nieterminalnych do minimum. Także każdy symbol nieterminalny powinien mieć możliwie tyle samo produkcji oraz jego produkcje powinny tworzyć tyle samo symboli nieterminalnych.
- Stworzona gramatyka powinna umożliwić na tworzenie jak najmniejszej ilości rozwiązań, które nie należą do języka, co prawda tak stworzony program nadal będzie działać i można odrzucić te rozwiązania na etapie parsowania, ale warto oszczędzić niepotrzebnych obliczeń i rozwiązań ten problem już na etapie tworzenia gramatyki.

2.3.6. Omówienia działania algorytmu ewolucji genetycznej

Mechanizmem, który wyróżnia ewolucji genetyczną na tle innych algorytmów ewolucyjnych jest mapowanie genotypu zapisywanego jako jednowymiarowa tablica liczb całkowitych na fenotyp - gramatykę. Przykładem może być genotyp:

[6, 71, 92, 59, 52, 95, 23, 45, 12, 2]

i gramatyka w notacji BNF:

<e> ::= <var>=<math><var>
<math> ::= <math><math> | <var><op>
<op> ::= + | -
<var> ::= a | b | c | d,

której symbolem startowym jest <e>. Wybór kolejnych produkcji odbywa się według zasady:

$$produckja = gen\_jako\_liczba\_cakowita \,\%\, ilosc\_dostepnych\_produkcji$$

Używając wyprowadzenia lewostronnego:

- Aktualne wyrażenie: <e>; Aktualny symbol nieterminalny: <e>; Ilość możliwych produkcji: 1; Wartość genu: 6; 6 mod 1 = 0, więc <e> zostaje zastąpiony przez <var>=<math><var>
- Aktualne wyrażenie: <var>=<math><var>; Aktualny symbol nieterminalny: <var>; Ilość możliwych produkcji: 4; Wartość genu: 71; 71 mod 4 = 3, więc <var> zostaje zastąpiony przez d
- Aktualne wyrażenie: d=<math><var>; Aktualny symbol nieterminalny: <math>; Ilość możliwych produkcji: 2; Wartość genu: 92; 92 mod 2 = 0, więc <math> zostaje zastąpiony <math><math>
- Aktualne wyrażenie: d=<math><math><var>; Aktualny symbol nieterminalny: <math>; Ilość możliwych produkcji: 2; Wartość genu: 59; 59 mod 2 = 1, więc <math> zostaje zastąpiony <var><op>

kontynuując analogicznie, po zastąpienie wszystkich symboli nieterminalnych, ostatecznie otrzymany fenotyp to:

d = a - b + c

Używanie tego prostego mechanizmu, w przypadku bardziej skomplikowanej gramatyki, może generować złożone rozwiązania. W połączeniu z metodami z algorytmów ewolucyjnych, możliwe jest teoretycznie ewoluowanie programów w dowolnym języku programowania, a następnie za pomocą odpowiedniej funkcji dopasowania znalezienie takiego, który rozwiąże dany problem. W kolejnej sekcji omówiono jak dobrać metryki, aby użyć tego mechanizmu do odkrywania procesów biznesowych.

2.4. Metryki

2.4.1. Metryki a funkcja dopasowania

Funkcja dopasowania powinna być możliwie najmniej kosztowna obliczeniowo, dlatego nie warto nadmiernie komplikować, gdyż mimo że wersja lepiej określi jak dobre jest dane rozwiązanie, to obliczenie może stać się wąskim gardłem całego algorytmu w rezultacie pogarszając jego działanie.

Funkcja dopasowania powinna nie może tylko mierzyć jak dobre jest rozwiązanie, dobry przykładem jest tutaj problem układanie planów (*e*ng. timetabling problem), gdzie większość otrzymanych rozwiązań będzie nieprawidłowa, dlatego lepszym pomysłem jest sprawdzanie jak blisko jesteśmy potencjalnie prawidłowego rozwiązania [35] [36].

Ostatecznie funkcja dopasowania w naszym algorytmie jest średnią ważoną metryk wymienionych w sekcji 2.2.4. Oprócz tego dodano kolejną metrykę złożoność. Użytkownik może określić z jakimi wagami wziąć pod uwagę poszczególne metryki.

2.4.2. Dodatkowa metryka - złożoność

Dodatkowa metryka zawiera informacje nie potrzebne do jakość odkrytego modelu, jednak istnieją teoretyczne przesłanki, że powinna wpłynąć pozytywnie na rozwiązania znajdowane przez algorytm. Różni się od nieco podobnych prostoty czy precyzji, tym że nie jest to podejście z perspektywy modelu, a wydajności algorytmu ewolucji i próby zminimalizowanie czasu potrzebnego na znalezienie rozwiązania.

Ideą jest promowanie rozwiązywania prostych problemów w prosty sposób. Chcemy unikać lokalnych maksimów - sytuacji, w której populacja zostanie zdominowana przez zbyt skomplikowane osobniki na wczesnym etapie ewolucji, zamiast tego nagradza sytuacje, w której wraz ze znajdowaniem coraz lepszego rozwiązania, stopień skomplikowanie modelu będzie stopniowo rósł. Prosta analogię można odnaleźć w naturze, w historii Ziemii w pewnym momencie dinozaury wyewoluowała jako najlepiej przystosowane do życia na tej planecie organizmy - lokalne maksimum i blokowały powstawanie mniej skomplikowanych form życia, które w rezultacie wyewoluowały do lepiej przystosowanych organizmów.

Większa złożoność każdego modelu przekłada się na dłuższe obliczanie metryk dla tego modelu, przez co znalezienie ostatecznego rozwiązanie zajmie więcej czasu.

Kolejną zaletą jest to, że korzysta z istniejących już obliczeń, przez co nie zwiększa w istotny sposób czasu każdej iteracji algorytmu, co pozwala na łatwiejsze porównanie działania algorytmu z i bez tej metryki.

2.4.3. Metryki - szczegóły

2.4.3.1. Prostota

Jest to najprostsza z metryk. Celem jest zmniejszenie skomplikowania modelu. Głównym czynnikiem na to wpływającym jest ilość aktywności w modelu. Idealna jest sytuacja, w której model ma tyle samo aktywności ile unikalnych aktywności jest w dzienniku zdarzeń. To nie zawsze jest możliwe, jednak chcąc otrzymać maksymalnie czytelny model powinniśmy do tego dążyć. Stąd też metryk wybrana skupia się na dwóch czynnikach, czyli ilości duplikatów w modelu i ilości brakujących wartości w modelu. Oczywiście znaczenie ma wielkość modelu, dlatego wartości tego musimy odnieść do ilości wszystkich zdarzeń w logu i modelu. Metrykę zapożyczono z [37] i ostatecznie wyrażono wzorem:

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

2.4.3.2. Odwzorowanie

Jest to najbardziej kosztowna obliczeniowo metryka. Pozostałe metryki obliczane są na podstawie informacji uzyskanych podczas obliczania odwzorowanie. Ważnym jest, żeby obliczać dopasowanie częściowo dla każdej aktywności, a nie całego wariantu, co sprawi, że metryka będzie bardziej wrażliwa na zmiany. Możliwe jest zastosowanie prostej metryki zero-jedynkową, która sprawdzałaby czy model zgadza się z wariantem logu, jednak szczególnie w przypadku algorytmu genetycznego nie sprawdzane byłoby o ile bliżej jesteśmy do celu, dopóki losowo nie znaleziony zostałby model idealnie pasujący do wariantu. Metrykę oparto o [38], jednak przy procesie z wieloma aktywnościami, najczęściej części aktywności może zostać dopasowana, co sprawi, że łączny błąd będzie niewielki, dlatego wprowadzono modyfikację i podniesiono otrzymany wynik do potęgi 4, żeby zrobić metrykę bardziej wrażliwą na zmiany w porównianu z innymi metrykami. Ostatecznie metrykę wyrażono wzorem:

$$M_o = (1 - blad\_odwzorowania)^4$$

$$blad\_odwzorowania = \frac{\sum_{ilosc\ procesow\ w\ logu} \frac{blad\ odwzorowania\ logu\ w\ modelu}{minimalna\ dugosc\ najlepszej\ sciezki\ w\ modelu + dugosc\ sciezki\ w\ logu}}{ilosc\ zdarzen\ w\ logu}$$

2.4.3.3. Precyzja

Pozwala na unikanie niewystarczającego dopasowania (*eng. underfitting*). Celem jest uniknięcie tworzenia modeli, dla których możliwe są dowolne zachowania. Możliwe jest osiągnięcie maksymalne wartości pozostałych metryk tworząc jednak bramkę or zawierającą wszystkie aktywności, jednak oczywistym jest, że nie jest to model, który jest pożądany. We wzorze skupiono się na ilości osiągalnych zdarzeń

następujących po danej aktywności, możliwych w modelu. Chcemy żeby poszczególne aktywności mogły poprzedzać tylko te, które poprzedzają w logu. Metrykę oparto o [39], jednak otrzymany wynik do potęgi $\frac{1}{3}$, bo oryginalna metryka zmienia się zbyt łatwo przez co jest nieproporcjonalna do zmian innych metryk i łatwo wpaść w lokalne maksimum, gdzie każda mała zmiana będzie wpływać na ogromną zmianę w metryce. Ostatecznie metrykę wyrażono wzorem:

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

2.4.3.4. Generalizacja

Unikanie nadmiernego dopasowania (*eng. overfitting*) W pierwszej chwili może wydawać się przeciwieństwem precyzji, jednak dobrą wizualizacją działania jest operacja domknięcia znana z dziedziny przetwarzania obrazów cyfrowych. Chodzi o wypełnienie brakujących w dzienniku zdarzeń wariantów, a nie o rozszerzenie na nowe zupełnie niezwiązane. Można to osiągnąć poprzez obliczenie średniej ważonej liczby przejścia w modelu przez dane zdarzenie, co sprawia, że wciąż zachowujemy ścieżki, które są często odwiedzane przez inne warianty, mimo że pozwalają na zachowanie niewidoczne w dzienniku zdarzeń, nie pozwalając na na tworzenie ścieżek, które nie pokrywają się z żadnymi ścieżkami. Wzięto także pierwiastek, gdyż wraz z ilością wykonywanie danej ścieżki wzrasta pewność, że jest ona ok i ta informacja staje się mniej cenna. Metrykę zapożyczono z [37] i ostatecznie wyrażono wzorem:

$$M_g = 1 - \frac{\sum_{zdarzenia\ w\ modelu} \frac{1}{\sqrt{ilosc\ wystapien\ zdarzenia}}}{ilosc\ zdarzen\ w\ logu}$$

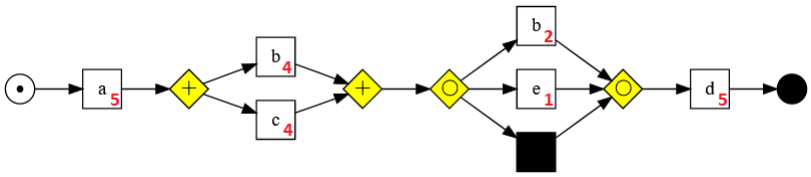
2.4.3.5. Złożoność

Metryka jest powiązana z odwzorowaniem, czyli kiedy odwzorowanie rośnie pozwalamy na tworzenie bardziej skomplikowanego modelu. Złożoność jest wyrażana jako ilość możliwych ścieżek w modelu. Jako, że np. dla bramki AND jest to n! zauważamy, że złożoność rośnie niewspółmiernie szybciej do odwzorowania, dlatego pierwiastek ze złożoności. W przypadku pętli złożoność byłaby nieskończona, dlatego przyjęto że pętla oznacza maksymalnie dwukrotne wykonanie pętli. Ostatecznie metrykę wyrażono wzorem:

$$M_z = 1 - \frac{1}{\sqrt{1 + blad\_odwzorowania * \sqrt{zlozonosc\ modelu}}}$$

2.4.4. Obliczanie metryk

W sekcji 2.2.3 przedstawiono przykład dziennika zdarzeń i warianty procesu uzyskane z niego. Poniżej przedstawiono przykładowy model tego procesu - na czerwono zaznaczono ilość wykonań danej aktywności - oraz zademonstrowano obliczanie metryk:



Rys. 2.10. Model, dla którego obliczane są metryki

2.4.4.1. Obliczanie prostoty

Model osiada jest jeden duplikat - b oraz jedno brakujące zdarzenie - f, więc wartość metryki wynosi:

$$M_{pro} = 1 - \frac{1+1}{6+6} = 0.8333$$

2.4.4.2. Obliczanie odwzorowania

Są trzy możliwość brak zdarzenia w modelu, brak zdarzenia w logu lub zdarzenie w logu różne od zdarzenia w modelu, co może być rozumiane jako brak zdarzenia w logu i modelu, dlatego w takim przypadku błąd wynosi 2, a w pozostałych 1. Nieprawidłowe odwzorowanie można przedstawić na kilka sposobów, jednak nie ma to znaczenia i błąd wynosi 3:

Ścieżka w modelu	a	b	e	d
Ścieżka w logu	a	f		d

Ścieżka w modelu	a		b	e	d
Ścieżka w logu	a	f			d

Rys. 2.11. Nieprawidłowe odwzorowanie

Dla pozostałych wariantów odwzorowanie jest pełne i błąd wynosi 0:

Ścieżka w modelu	a	b	c	e	d
Ścieżka w logu	a	b	c	e	d

Rys. 2.12. Prawidłowe odwzorowanie

Podstawiając wszystkie odwzorowania do wzoru otrzymujemy:

$$blad\_odwzorowania = \frac{2*\frac{0}{5+5}+\frac{0}{4+4}+\frac{0}{5+5}+\frac{3}{4+3}}{22} = 0.0195$$
$$M_o = (1-0.0195)^4 = 0.9243$$

2.4.4.3. Obliczanie precyzji

Potrzebna jest ilość osiągalnych kolejnych zdarzeń dla danej aktywności. Zależy ona od kontekstu w jakim dana aktywność została wywołana np. dla bramki AND możemy jako kolejne wywołać tyle aktywności ile nie zostało jeszcze wywołanych w tej bramce, z wyjątkiem ostatniej i gdzie przechodzimy do kolejnych aktywności spoza bramki. Dla demonstrowanego przykładu wygląda to następująco:

Poprzedzające zdarzenia	Zdarzenia w logu	Ilość zdarzeń w logu	Zdarzenia w modelu	Ilość zdarzeń w modelu
[]	['a']	1	['a']	1
['a']	['b', 'c']	2	['b', 'c']	2
['a', 'b']	['c']	1	['c']	1
['a', 'b', 'c']	['b']	1	['b', 'e', 'd']	3
['a', 'b', 'c', 'b']	['d']	1	['e', 'd']	2
['a', 'b', 'c', 'b', 'd']	['koniec']	1	['koniec']	1
['a', 'c']	['b']	1	['b']	1
['a', 'c', 'b']	['d', 'e']	2	['b', 'd', 'e']	3
['a', 'c', 'b', 'd']	['koniec']	1	['koniec']	1
['a', 'c', 'b', 'e']	['d']	1	['b', 'd']	2
['a', 'c', 'b', 'e', 'd']	['koniec']	1	['koniec']	1

Rys. 2.13. Kolejne zdarzenia

Podstawiając te dane do wzoru otrzymujemy:

$$M_{pre} = (1 - (\frac{1-1}{1} + \frac{2-2}{2} + \frac{1-1}{1} + \frac{3-1}{3} + \frac{2-1}{2} + \frac{1-1}{1} + \frac{1-1}{1} + \frac{3-2}{3} + \frac{1-1}{1} + \frac{2-1}{2} + \frac{1-1}{1}))^{\frac{1}{3}} = 0.9465$$

2.4.4.4. Obliczanie generalizacji

Pewną słabością tej metryki jest to, że wpływa na nią rozmiar dziennika zdarzeń. Jeśli ilość rekordów jest mała, jak w powyższym przykładzie, to generalizacja będzie słaba. Starając się znaleźć najlepszy model używamy zawsze tego samo logu, więc nie wpływa to na prawidłowość rozwiązania. W tym przypadku otrzymujemy:

$$M_g = 1 - \frac{(\frac{1}{\sqrt{5}} + \frac{1}{\sqrt{4}} + \frac{1}{\sqrt{4}} + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{1}} + \frac{1}{\sqrt{5}})}{6} = 0.3997$$

2.4.4.5. Obliczanie złożoności

Najpierw obliczamy złożoności dla poszczególnych bramek, a następnie podstawiamy do wzoru wraz z błędem dopasowania obliczony w sekcji 2.4.4.2.

$zlozonosc\_dla\_bramki\_AND = 2! = 2$

$zlozonosc\_dla\_bramki\_OR = 2! * \frac{2!}{(2!*(2-2)!)} + 1! * \frac{2!}{(1!*(2-1)!)} + 0! * \frac{2!}{(0!*(2-0)!)} = 2 + 2 + 1 = 5$

$zlozonosc = 1 * 2 * 5 * 1 = 10$

$$M_z = 1 - \frac{1}{\sqrt{1+0.0195 * \sqrt{10}}} = 0.9706$$

3. Projekt i implementacja

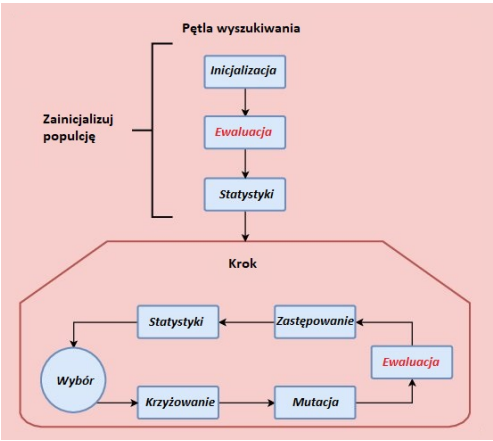
3.1. Wykorzystane technologie

3.1.1. Python 3.8.1

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

3.1.2. PonyGE2

PonyGE2 [40] jest implementacja ewolucji genetycznej w języku Python. Pozwala na łatwa konfiguracje parametrów ewolucji genetycznej oraz możliwość dodania własnych problemów oraz sposobów ewaluacji wyników.



Rys. 3.1. Ogólny schemat działania PonyGE2

3.2. Tworzenie gramatyki procesu biznesowego

Zgodnie z zaprezntowanymi wcześniej zasadmi odnośnie jak poiwnien wygladac midel zdecowydowano sie na W wykorzystanie gramryec postawioneo się zdecydować na soundness, czyli bramki zamknięte, jeden event ckończowyco pozwala na tworzenie, brak bramek ktore nie sa domkniete./ Pozwala

to zredukowac przestrzen roziwazan jednoczesniej tworzac rzowiazania, ktore na pewno nie generuja bledow. Łatwe mapowanie na BPMN. Przy tworzeniu gramatyki procesu biznesowego ważnym jest, żeby znaleźć balans, jeśli chodzi o poziom skomplikowania zaproponowanej gramatyki. W pracy [41] autorzy przeanalizowali składniki języka BPMN pod kątem częstotliwości ich stosowania. z pracy wynika, że najczęściej stosowany 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

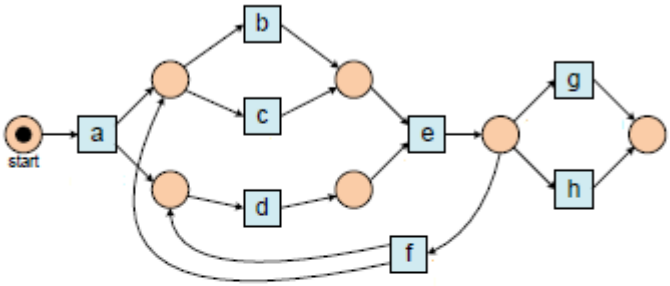
Przykład wygenerowanej gramatyki: and({d}opt({f})and({a}{c})lop(seq(lop({a}){e})))

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

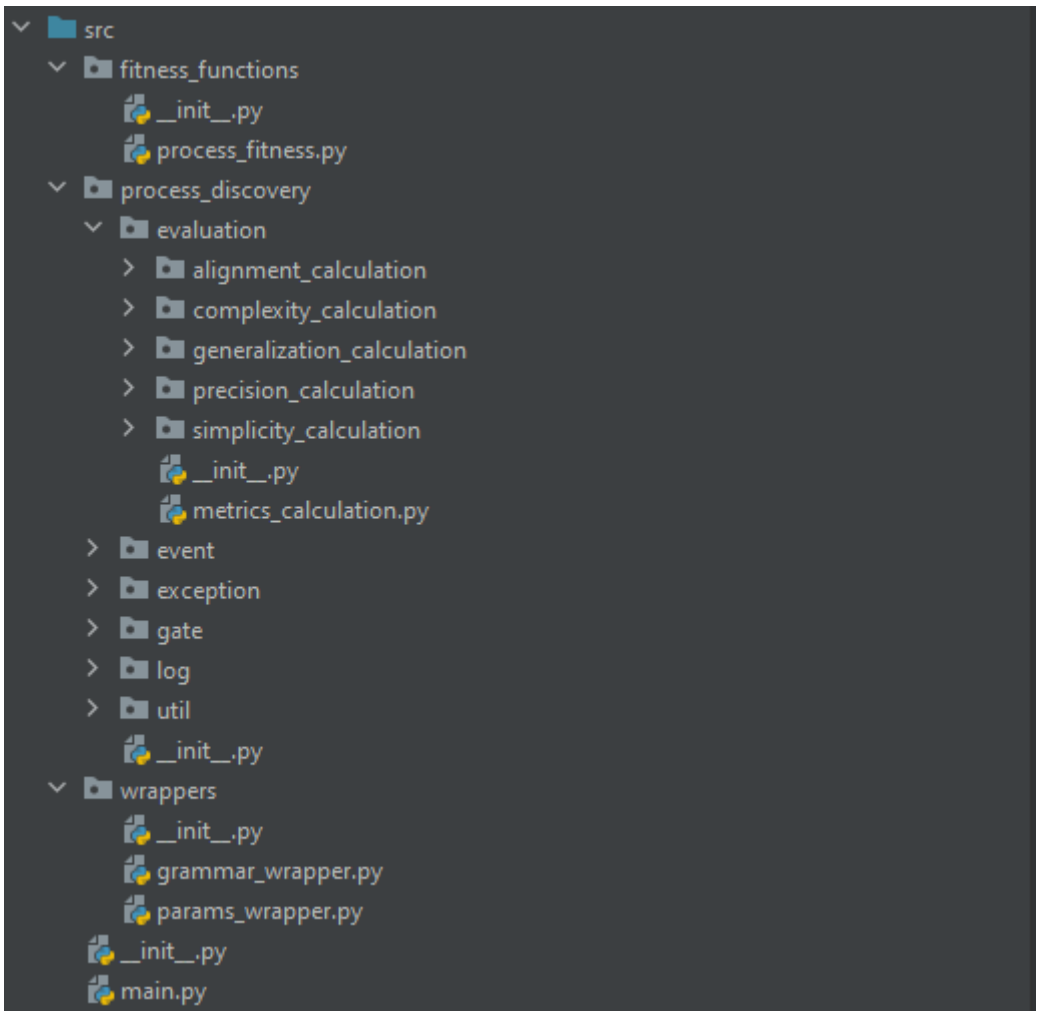
3.3.0.1. Podział na moduły

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 cześć z modułów PonyGE2. Moduły, które nadpisano to params, który zawiera konfigurację aplikacji oraz grammar, gdzie dodano zmiany w jaki sposób parsowana jest podana gramatyka. dorzucic nazwe jakiegos design patternsa

– fitness_functions - zawiera klasę bazowy moduł, gdzie znajduje się bazowa klasa dla obliczania metryk

– process_discovery - moduł zawiera całą logikę obliczenia metryk

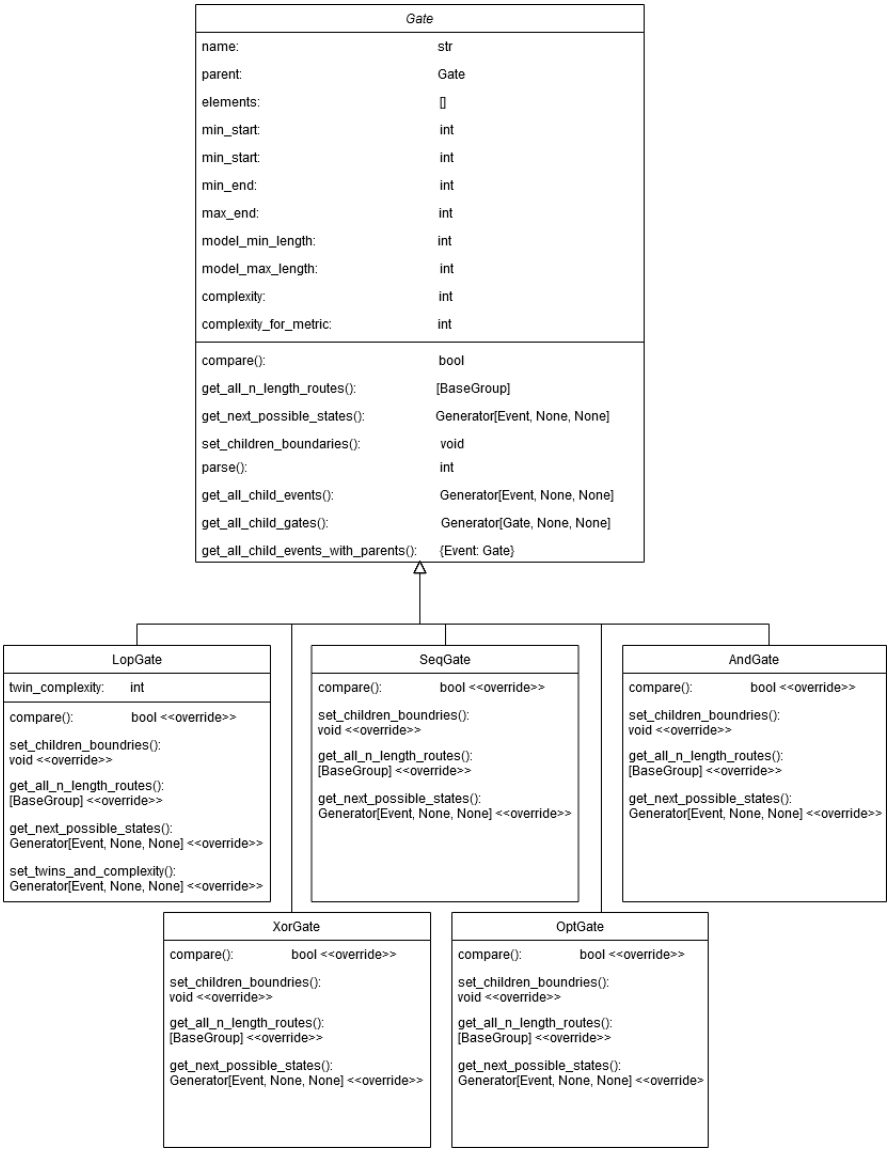


Rys. 3.3. Podział na moduły

3.3.0.2. Model

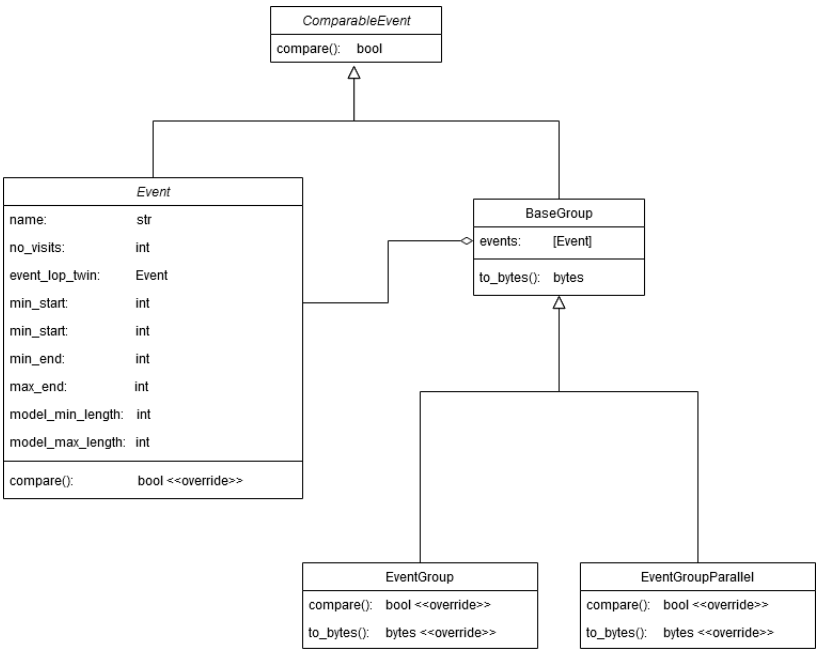
Podział na dwie klasy przydatne na różnym etapie procesowania:

Gate: Gramatyka zostaje sparsowana na to klasa. Pozwala to zastąpić proces w formie ciągu znaków na formę, na której łatwiej będzie nam operować w przyszłość.



Rys. 3.4. Gate UML

EventGroup: Obliczanie metryk dla klasy Gate byłoby utrudnione z uwagi na duża ilość bramek logicznych Takie modularne podejście pozwala na dodanie nowych bramek logiczny bez konieczności zmieniana metody obliczanie dopasowania, która jest najbardziej złożonym algorytmem występującym w programie.

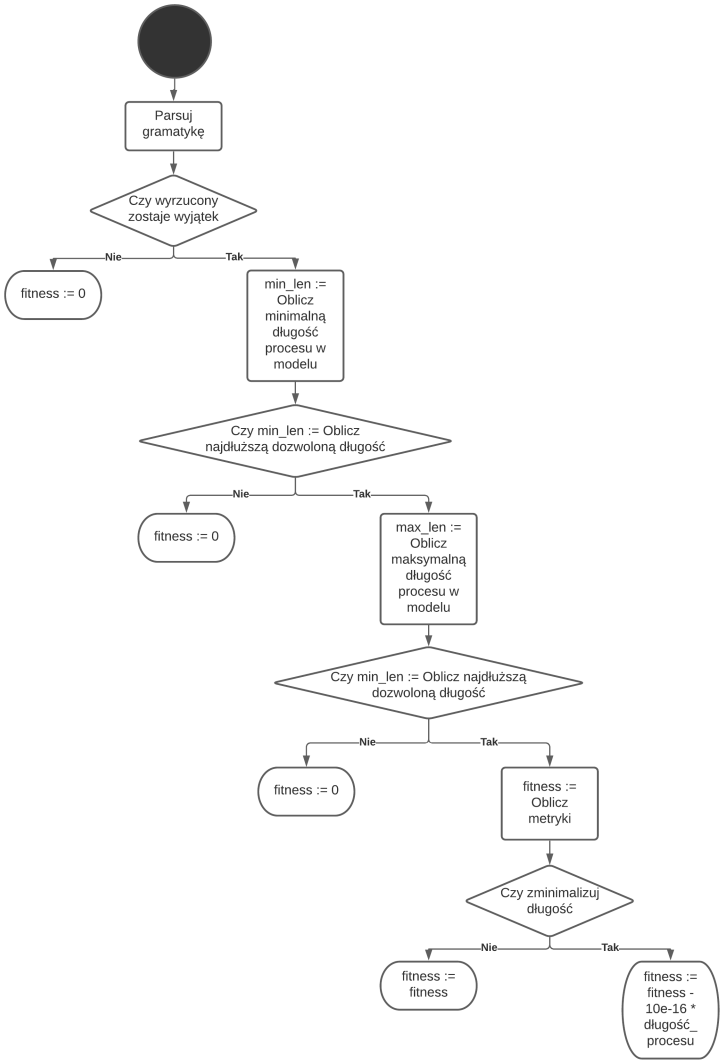


Rys. 3.5. Event UML

3.4. Implementacja

W tej części przedstawiono listingi z pseudokodem opartym na języku Python. Tam gdzie to konieczne pozostawiono słowa kluczowa oraz operatory tego języka.

3.4.0.1. Ogólny schemat blokowy



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 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. To na tym etapie odrzucamy też procesy, które mimo, że gramatyka pozwala na ich stworzenie inie moją sensu z punktu biznesowego, co pozwala na ograniczenie zasobów i nie przeliczanie dalszych rzeczy dla procesow, ktore sa bezwartosciowe. Odrzucamy procesy, ktore

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

   for i w zakresie długość_wyrażenia:
      if wyrażenie[i] == "{":
         zdarzenie := Event(wyrażenia[i + 1])
         dodaj zdarzenie do aktualnie parsowanej bramki
         i += 2
      elif wyrażenie[i] == ")":
         return i+1
      elif i+4 < długość_wyrażenia:
          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(wyrażenie[i+4:])
                if self.name == "seq" or self.name == "lop":
                   if int(wyrażenie[i+2]) <= długpść(gate.elementy):
                      for x in gate.elementy[int(wyrażenie[i+2]):]:
                         self.dodaj_element(x)
                dodaj zdarzenie do aktualnie parsowanej bramki
                i += ilość_przeparsowanych_znaków
             else:
                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:
          wyrzuć wyjątek
```

Listing 3.2. Parser gramatyki

3.4.0.3. Obliczanie metryk

Mając już sparsowany model musimy obliczyć metryki. Najbardziej problematyczną metryką do obliczenia jest dopasowanie. Obliczanie dopasowanie można rozbić na następujące kroki:

- Znalezienie ścieżek o długości n w modelu.

- Przerobienie ścieżek na postać zawierającą BaseGroup.
- Obliczenie dopasowania.

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 znajduję 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 = obilcz_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_średnią_ważoną_metryk
   return najlepszy_wynik
```

Listing 3.3. Obliczanie metryk

3.4.0.4. Obliczanie metryk dla jednego procesu

Ważnym jest, żeby jak najbardziej ograniczyć zbędne obliczenia. W tym celu, co pozwolilo usprawnic caly proces

```
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_osiągnięty and górny_limit_osiągnięty):
      if n >= min(oblicz_maksymalną_dozwoloną_długość(długość_procesu),
                  długość_procesu - minimalny_błąd_dopasowania):
         górny_limit_osiągnięty := True
         n += (-i if i % 2 == 1 else i); i += 1
         continue
      if n <= max(oblicz_minimalną_dozwoloną_długość(długość_procesu),
                  długość_procesu + minimalny_błąd_dopasowania):
         dolny_limit_osiągnięty := 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 lista_ścieżek_do_obliczenia
            posortowane_ścieżki := posortuj lista_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

3.4.0.5. Wyszukiwanie w modelu procesów o określonej długości

Łatwiejszym jest znalezienie wszystkich ścieżek o określonej długości. Najprawdopodobn iejsze jest, ze najlepiej dopasowana sciezka bedzie miala dlugosc jak wejscie w logu dlatego zaczyna od tej dlugosci. Algorytm rekurencyjny. Implementacja różni się w zależności od przeszukiwanej bramki logicznej. Poniżej zaprezentowano przykład dla bramki.

```
def znajdź_wszystkie_ścieżki_długości_n(n, 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_l
            for i in range(lower_limit, upper_limit + 1):
                try:
                    child_all_n_length_routes = elem.get_all_n_length_routes(i, process)
                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

3.4.0.6. Obliczanie dopasowania

Pomysł zaczerpnięty z algorytmu Needlessmann-Wunsch [42], 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.

```
def oblicz_dopasowanie(model, log):
    błąd := {'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
    # Wypelnij osie macierzy właściwymi wartościami
    for j in range(n):
        macierz_rozwiązań[0][j] := błąd['PRZERWA'] * j

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

    ścieżka := znajdź_ściezkę(macierz_rozwiązań, błąd['PRZERWA'], model,
    log, najlepiej_dopasowana_ścieżka_podmodelu)

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

Listing 3.6. Obliczanie dopasowania

3.4.0.7. Znajdowanie ścieżki w modelu

Potrzebne do obliczenia precyzji oraz generalizacji. Z wzgledu na zmiany

```
def znajdź_sciezkę(macierz_rozwiązań, model, log, rozwiązania_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_rozwiązań[i][j] == macierz_rozwiązań[i - 1][j] + długość_grupy_zdarzeń *
                [model_result.append(None) for _ in range(długość_grupy_zdarzeń)]
                macierz_rozwiązań[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_rozwiązań[i][j] == macierz_rozwiązań[i - 1][k] +
                        (długość_grupy_zdarzeń + (j-k) - 2 * długość(processes)) * błąd['PRZERW
                        [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_rozwiązań[i][j] = 0
                        i -= 1; j = k
                        znaleziono_dopasowanie = True
                        break
                if not znaleziono_dopasowanie:
                    if macierz_rozwiązań[i][j] == macierz_rozwiązań[i][j - 1] + błąd['PRZERWA']
                        macierz_rozwiązań[i][j] = 0
                        j -= 1
        else:
            if macierz_rozwiązań[i][j] == macierz_rozwiązań[i - 1][j] + kara:
                model_result.append(None)
                macierz_rozwiązań[i][j] := 0
                i -= 1
            elif macierz_rozwiązań[i][j] == macierz_rozwiązań[i][j - 1] + kara:
                macierz_rozwiązań[i][j] := 0
                j -= 1
            elif macierz_rozwiązań[i][j] == macierz_rozwiązań[i - 1][j - 1]:
                model_result.append(model[i-1])
                log = log.replace(model[i-1].name, "", 1)
                macierz_rozwiązań[i][j] := 0
                i -= 1; j -= 1
    return sciezka
```

Listing 3.7. Znajdowanie ścieżki w modelu

3.4.0.8. Ogólny opis dlaczego ten program jest fajny

Cache

W sytuacji kiedy wiele obliczeń się powtarza można znacząca przyspieszyć czas działania aplikacji poprzez zastosowanie cachowania. W przypadku naszego algorytmu można zauważyć dwa miejsca, w których często dochodzi to powtórzeń: Poprzednio obliczone rozwiązanie może się powtórzyć. W tym wypadku możemy skorzystać z cache genotypów, dostarczane przez bibliotekę PonyGE2. Podczas obliczania dopasowania, które jest najbardziej kosztownym obliczeniem. Ponadto z uwagi na duża ilość obliczeń, żeby ograniczyć rozmiar cache zaimplementowano cachowanie LRU.

3.5. Wybór parametrów algorytmu

Wybór parametrów algorytmu ma ogromny wpływ na jakość i szybkość znalezienia rozwiązania. Jest kilka zasad, którymi należy się kierować przy tym wyborze właśnie. Ilość parametrów wymagana przez ponyGE2 jest duża, dodatkowo tworząc mimo, że starano się ograniczyć możliwość konfiguracji, która nie daje dużo korzyści do minimum tworząc aplikacje dodano kilka innych niezbędnych parametrów. Z tego powodu, poniżej przedstawiono i krótko omówiono niezbędne do działania aplikacji parametry.

Włącza cache:
CACHE: True
CODON_SIZE: 100000
Ilość wątków procesora: *CORES: 4*
CROSSOVER: subtree
CROSSOVER_PROBABILITY: 0.75
DEBUG: False
ELITE_SIZE: 30
GENERATIONS: 100000
MAX_GENOME_LENGTH: 500
GRAMMAR_FILE: process-subtree.bnf
INITIALISATION: PI_grow
INVALID_SELECTION: False
LOOKUP_FITNESS: True
MAX_INIT_TREE_DEPTH: 13
MAX_TREE_DEPTH: 21
MULTICORE: True
MULTI_OBJECTIVE: False
MUTATION: subtree
MUTATION_EVENTS: 1
POPULATION_SIZE: 500
FITNESS_FUNCTION: process_fitness

REPLACEMENT: generational
SAVE_STATE_STEP: 10
SELECTION: tournament
TOURNAMENT_SIZE: 16
VERBOSE: True
MAX_WRAPS: 3
ALIGNMENT_CACHE_SIZE: 32*1024
DATASET: discovered-processes.csv
MAX_ALLOWED_COMPLEXITY_FACTOR: 300
MINIMIZE_SOLUTION_LENGTH: True
RESULT_TOLERANCE_PERCENT: 5
TIMEOUT: 5

Rekomendowane wagi poszczególnych metryk:
WEIGHT_ALIGNMENT: 8
WEIGHT_COMPLEXITY: 2
WEIGHT_GENERALIZATION: 2
WEIGHT_PRECISION: 2
WEIGHT_SIMPLICITY: 2

4. Dyskusja rezultatów

4.1. Przykładowe wyniki

Metoda została przetestowana dla dziennika zdarzeń:

455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	

Rys. 4.1. Dziennik zdarzeń

4.2. Porównanie z innymi algorytmami

4.3. Wyniki w zależności od przyjętych metryk

4.3.1. Złożoność

4.4. Wnioski

5. Podsumowanie

Bibliografia

[1] Thomas H Davenport. *Process innovation: reengineering work through information technology*. Harvard Business Press, 1993.

[2] Michael Hammer i James Champy. „Reengineering the corporation: A manifesto for business revolution". W: *Business Horizons* 36.5 (1993), s. 90–91. ISSN: 0007-6813. DOI: *https://doi.org/10.1016/S0007-6813(05)80064-3*.

[3] Ivar Jacobson, Maria Ericsson i Agneta Jacobson. *The Object Advantage: Business Process Reengineering with Object Technology*. USA: ACM Press/Addison-Wesley Publishing Co., 1994. ISBN: 0201422891.

[4] Hans-Erik Eriksson i Magnus Penker. *Business Modeling With UML: Business Patterns at Work*. 1st. USA: John Wiley i Sons, Inc., 1998. ISBN: 0471295515.

[5] Geary A. Rummler i Alan P. Brache. *Improving performance: how to manage the white space on the organization chart*. Jossey-Bass, 1995.

[6] Wil Aalst. „Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management". W: t. 3098. Sty. 2003, s. 1–65. ISBN: 978-3-540-22261-3. DOI: *10.1007/978-3-540-27755-2_1*.

[7] Nathaniel Palmer. *What is BPM?*

[8] Wil Aalst. „Aalst, W.M.P.: Business process management: a comprehensive survey. ISRN Softw. Eng. 1-37". W: *ISRN Software Engineering* (sty. 2012), ??–?? DOI: *10.1155/2013/507984*.

[9] M. Dumas i in. *Fundamentals of Business Process Management*. Springer Berlin Heidelberg, 2013. ISBN: 9783642331435.

[10] Jan Recker i in. „Business Process Modeling- A Comparative Analysis". W: *Journal of the Association of Information Systems* 10 (kw. 2009). DOI: *10.17705/1jais.00193*.

[11] OMG. *Business Process Model and Notation (BPMN), Version 2.0*. Object Management Group, 2011.

[12] Jan Mendling, H.A. Reijers i Wil Aalst. „Seven Process Modeling Guidelines (7PMG)". W: *Information and Software Technology* 52 (lut. 2010), s. 127–136. DOI: *10.1016/j.infsof.2009.08.004*.

[13] Marc Kerremans. „Market Guide for Process Mining". W: *Gartner* (kw. 2018).

[14] Wil Aalst i in. „Process Mining Manifesto”. W: t. 99. Sierp. 2011, s. 169–194. ISBN: 978-3-642-28107-5. DOI: *10.1007/978-3-642-28108-2_19*.

[15] Wil Aalst. „Process Mining: Overview and Opportunities”. W: *ACM Transactions on Management Information Systems* 3 (lip. 2012), s. 7.1–7.17. DOI: *10.1145/2229156.2229157*.

[16] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016, s. 163–240. ISBN: 978-3-662-49850-7. DOI: *10.1007/978-3-662-49851-4*.

[17] Wil Aalst, A. Weijters i Laura Măruşter. „Workflow Mining: Discovering Process Models from Event Logs”. W: *Knowledge and Data Engineering, IEEE Transactions on* 16 (paź. 2004), s. 1128 –1142. DOI: *10.1109/TKDE.2004.47*.

[18] Jan Martijn Van der Werf i in. „Process Discovery Using Integer Linear Programming”. W: t. 94. Czer. 2008, s. 368–387. ISBN: 978-3-540-68745-0. DOI: *10.1007/978-3-540-68746-7_24*.

[19] A. Weijters, Wil Aalst i Alves Medeiros. *Process Mining with the Heuristics Miner-algorithm*. T. 166. Sty. 2006.

[20] B Dongen i Wil Aalst. „Multi-phase process mining: Aggregating instance graphs into EPCs and Petri nets”. W: *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management* (sty. 2005).

[21] Raji Ghawi. *Process Discovery using Inductive Miner and Decomposition*. Paź. 2016.

[22] Wil M. P. van der Aalst. „Relating Process Models and Event Logs - 21 Conformance Propositions”. W: *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2018 Satellite event of the conferences: 39th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2018 and 18th International Conference on Application of Concurrency to System Design ACSD 2018, Bratislava, Slovakia, June 25, 2018*. Red. Wil M. P. van der Aalst, Robin Bergenthum i Josep Carmona. T. 2115. CEUR Workshop Proceedings. CEUR-WS.org, 2018, s. 56–74.

[23] F. Blum. „Metrics in process discovery”. W: 2015.

[24] B. F. van Dongen, J. Mendling i W. M. P. van der Aalst. „Structural Patterns for Soundness of Business Process Models”. W: *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*. 2006, s. 116–128. DOI: *10.1109/EDOC.2006.56*.

[25] Ahmed Awad i Frank Puhlmann. „Structural Detection of Deadlocks in Business Process Models”. W: t. 7. Grud. 2008, s. 239–250. ISBN: 978-3-540-79395-3. DOI: *10.1007/978-3-540-79396-0_21*.

[26] P. A. Vikhar. „Evolutionary algorithms: A critical review and its future prospects”. W: *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*. 2016, s. 261–265. DOI: *10.1109/ICGTSPICC.2016.7955308*.

[27] D. Noever i Subbiah Baskaran. *Steady-state vs. generational genetic algorithms: A comparison of time complexity and convergence properties*. Lip. 1992.

[28] John R. Koza. *Non-Linear Genetic Algorithms for Solving Problems*. United States Patent 4935877. filed may 20, 1988, issued june 19, 1990, 4,935,877. Australian patent 611,350 issued september 21, 1991. Canadian patent 1,311,561 issued december 15, 1992. 1990.

[29] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992. ISBN: 0262111705.

[30] Conor Ryan, Jj Collins i Michael O Neill. „Grammatical evolution: Evolving programs for an arbitrary language". W: *Lecture Notes in Computer Science Genetic Programming* (1998), 83–96. DOI: *10.1007/bfb0055930*.

[31] J. Backus. „The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference". W: *IFIP Congress*. 1959.

[32] Peter Naur. „A Course of Algol 60 Programming". W: *ALGOL Bull.* Sup 9 (sty. 1961), 1–38. ISSN: 0084-6198.

[33] Donald E. Knuth. „backus normal form vs. Backus Naur form". W: *Communications of the ACM* 7.12 (1964), s. 735–736. ISSN: 0001-0782. DOI: *http://doi.acm.org/10.1145/355588.365140*.

[34] Miguel Nicolau i Alexandros Agapitos. „Understanding grammatical evolution: Grammar design". W: sty. 2018, s. 23–53. DOI: *10.1007/978-3-319-78717-6_2*.

[35] Nichael Lynn Cramer. „A representation for the Adaptive Generation of Simple Sequential Programs". W: *Proceedings of an International Conference on Genetic Algorithms and the Applications*. Red. John J. Grefenstette. Carnegie-Mellon University Pittsburgh PA USA, 1985, s. 183–187.

[36] David Beasley, David R. Bull i Ralph R. Martin. „An Overview of Genetic Algorithms: Part 1, Fundamentals". W: *University Computing* 15.2 (1993), s. 58–69.

[37] 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*.

[38] Wil van der Aalst, Arya Adriansyah i Boudewijn van Dongen. „Replaying history on process models for conformance checking and performance analysis". W: *WIREs Data Mining and Knowledge Discovery* 2.2 (2012), s. 182–192. DOI: *https://doi.org/10.1002/widm.1045*. eprint: *https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1045*.

[39] J. Munoz-Gama i J. Carmona. „Enhancing precision in Process Conformance: Stability, confidence and severity". W: *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. 2011, s. 184–191. DOI: *10.1109/CIDM.2011.5949451*.

[40] Michael Fenton i in. „PonyGE2”. W: *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2017). DOI: *10.1145/3067695.3082469*.

[41] 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 Bellahsène i Michel Léonard. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 465–479. ISBN: 978-3-540-69534-9.

[42] 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*.