AKADEMIA GÓRNICZO – HUTNICZA
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

mgr Dariusz Pałka

Algorytmy genetyczne w automatycznym generowaniu gramatyk bezkontekstowych dla potrzeb syntaktycznej interpretacji wzorców

ROZPRAWA DOKTORSKA

Promotor
Prof. dr hab. Marek Ogiela

Kraków 2006

Autor wyraża swe gorące podziękowania promotorowi pracy prof. dr. hab. Markowi Ogieli, którego uwagi merytoryczne i formalne miały ogromne znaczenie w przygotowaniu niniejszej pracy.

Spis treści

1. Wstęp

Proces wnioskowania (lub wywodzenia) gramatyk (ang. *grammatical inference*), polegający na generowaniu szukanej gramatyki na podstawie przykładowych słów (sentencji) należących do języka określonego przez tę gramatykę, stanowi istotny składnik systemu syntaktycznego rozpoznawania wzorców [24]. Celem tego procesu jest znalezienie reguł syntaktycznych dla nieznanej gramatyki **G** w oparciu o skończony zbiór **S+** słów należących do języka **L(G)** generowanego przez gramatykę **G** i, ewentualnie, przez również skończony, zbiór **S-** słów nienależących do **L(G)**.

Jako że proces analizy syntaktycznej wymaga zadania gramatyki służącej do klasyfikacji wzorców [24], [69], w sytuacji, gdy gramatyka ta nie jest znana *a priori*, ale znane są przykładowe słowa należące do generowanego przez nią języka (i, ewentualnie, przykładowe słowa nienależące do tego języka), gramatyka ta może zostać wydedukowana przez człowieka, lub też znaleziona w sposób automatyczny w procesie wnioskowania gramatyk. Ponieważ często konstrukcja gramatyki przez człowieka, szczególnie dla bardziej złożonych przypadków, jest trudna lub wręcz niemożliwa, w dziedzinie syntaktycznego rozpoznawania wzorców istnieje duże zapotrzebowanie na algorytmy umożliwiające generowanie gramatyk w sposób automatyczny. Wyniki badań nad sposobem automatycznego generowania gramatyk na podstawie przykładów należących do języka stanowią także ważny wkład w rozwój psycholingwistyki, a w szczególności w zrozumienie mechanizmów przyswajania języków naturalnych. Opracowana teoria i uzyskane rezultaty pozwolą prawdopodobnie, na lepsze zrozumienie procesów uczenia się, takich jak te, w których dzieci na podstawie zasłyszanej pewnej skończonej ilości zdań należących do języka są w stanie budować własne zdania poprawne pod względem gramatycznym [61], [62]. Automatyczne generowanie gramatyk na podstawie przykładowych sentencji języka stanowi także ważny wkład w rozwój szeroko rozumianej sztucznej inteligencji. Istnieje dla niego wiele praktycznych zastosowań zarówno w dziedzinie rozpoznawania i klasyfikacji wzorców, jak i w sferze rozumienia przez maszyny języków naturalnych, na przykład przy projektowaniu robotów wykonujących złożone polecenia werbalne czy też tworzeniu programów agentowych komunikujących się z użytkownikiem za pomocą języka naturalnego [5].

Mimo tego, iż podstawy teoretyczne dotyczące wnioskowania gramatyk zostały przedstawione przez Golda [26] już w roku 1967, dopiero pod koniec lat osiemdziesiątych pojawiły się pierwsze działające algorytmy z tego zakresu [28]. Algorytmy te dotyczyły głównie wnioskowania gramatyk regularnych, które stanowią najprostszą klasę gramatyk w hierarchii Chomsky'ego [7], [30]. Język generowany przez taką gramatykę może być rozpoznany za pomocą automatu skończonego [30].

Najogólniej zadanie wnioskowania (generowania) gramatyki regularnej na podstawie przykładów jest zdefiniowane następująco: mając skończony zbiór przykładów pozytywnych (sentencji należących do języka generowanego przez szukaną gramatykę **G**) i skończony zbiór przykładów negatywnych (który w szczególności może być pusty), należy znaleźć gramatykę regularną **G***, równoważną z szukaną gramatyką **G**. Przy czym gramatyki **G*** i **G** uznawane są za równoważne, jeśli generowane przez nie języki są takie same. Ponieważ gramatyki regularne mogą być reprezentowane za pomocą różnych metod, takich jak zbiór produkcji, wyrażenia regularne, deterministyczne automaty skończone (**DAS**) czy też niedeterministyczne automaty skończone (**NAS**), istotny wpływ na algorytm generowania gramatyki ma wybór jej reprezentacji. W większości prac poświęconych wnioskowaniu gramatyk regularnych jako sposób reprezentacji gramatyki przyjmowany jest **DAS**. Na ogół jest to podyktowane następującymi cechami **DAS**: ich sposób działania jest łatwy do zrozumienia; dla każdej gramatyki regularnej istnieje unikalny **DAS** o minimalnej liczbie stanów; istnieją efektywne algorytmy o wielomianowym czasie wykonywania pozwalające znaleźć minimalny **DAS** oraz sprawdzić, czy dwa automaty **DAS** są sobie równoważne lub czy język reprezentowany przez pewien automat **DAS** jest nadzbiorem języka reprezentowanego przez inny automat **DAS**. Wszystkie te algorytmy są często wykorzystywane w różnych sposobach wnioskowania gramatyk regularnych.

Wnioskowanie gramatyk regularnych stanowi trudny problemem, bowiem gramatyka taka nie może być poprawnie zidentyfikowana wyłącznie na podstawie pozytywnych przykładów języka [26]. Co więcej, nie istnieje efektywny algorytm pozwalający zidentyfikować **DAS** o minimalnej liczbie stanów dla dowolnie wybranego zbioru przykładów pozytywnych i negatywnych [58]. Aby algorytm identyfikujący **DAS** mógł być efektywny, wymaga on dostarczenia dodatkowych informacji w procesie uczenia, których przykładem jest spełnienie przez zbiór uczący pewnych dodatkowo założonych kryteriów. Informacje takie mogą mieć także postać odpowiedzi na pytania zadawane przez algorytm identyfikujący **DAS** udzielanych przez 'nauczyciela' znającego poszukiwaną gramatykę.

Oncina i Garcia wykazali, że możliwe jest jednoznaczne zidentyfikowanie **DAS** na podstawie charakterystycznego zbioru przykładów, czyli takiego, który zawiera informacje o wszystkich stanach i przejściach w szukanym **DAS** [52]. Parekh, Nichitiu i Honavar zaprezentowali algorytm znajdujący **DAS** w czasie wielomianowym na podstawie etykietowanego zbioru przykładów oraz odpowiedzi 'nauczyciela' na zadawane pytania [60]. Algorytm ten bazuje na algorytmie przedstawionym przez Angluina, w którym proces uczenia jest sterowany przez najmniejszego adekwatnego 'nauczyciela' (ang. *minimal adequate teacher*). 'Nauczyciel' ten potrafi odpowiadać na pytania czy, dana sentencja (słowo) należy do języka generowanego przez poszukiwaną

gramatykę oraz czy dany **DAS** jest równoważny z poszukiwanym [3], [58]. Generowanie (wnioskowanie) gramatyki może zostać także sformułowane jako problem wyszukiwania właściwego automatu skończonego w przestrzeni wszystkich automatów skończonych. Ponieważ przestrzeń ta jest nieskończona, aby proces wyszukiwania mógł być skuteczny, konieczne jest jej ograniczenie. Jednym ze sposobów takiego ograniczenia przeszukiwanej przestrzeni jest jej zawężenie do automatów powstałych przez łączenie stanów w prefiksowym automacie drzewiastym (**PAD)** (ang. *prefix tree automaton*) zbudowanym na bazie przykładów należących do szukanego języka. Mimo że tak powstała podprzestrzeń jest skończona, to ilość należących do niej elementów stanowi wykładniczą funkcję ilości stanów początkowego **PAD**, co powoduje, że bezpośrednie przeszukiwanie tej przestrzeni jest rozwiązaniem niepraktycznym, głównie ze względu na złożoność czasową [58].

Dodatkowo pojawia się pytanie, czy tak ograniczona przestrzeń automatów skończonych zawiera poszukiwany minimalny **DAS**? Jak to zostało pokazane w pracach [14] i [59], jeśli zbiór przykładów pozytywnych jest strukturalnie kompletny, to powstała podprzestrzeń zawiera szukany minimalny **DAS**. Przy czym zbiór przykładów pozytywnych **S+** jest strukturalnie kompletny, jeśli dla każdego przejścia w szukanym **DAS** istnieje przynajmniej jedno słowo należące do **S+,** dla którego występuje to przejście, oraz dla każdego stanu akceptującego w szukanym **DAS** istnieje przynajmniej jedno słowo należące do **S+,** dla którego osiągany jest ten stan.

Mimo że podprzestrzeń powstała przez łączenie stanów **PAD** zbudowanego na bazie strukturalnie kompletnego zbioru przykładów zawiera poszukiwany minimalny **DAS,** to jej przeszukiwanie za pomocą standardowych algorytmów, takich jak na przykład przeszukiwanie wszerz, jest nieefektywne ze względu na jej rozmiar. Istnieją jednak metody pozwalające na bardziej efektywne przeszukiwanie podprzestrzeni automatów skończonych w oparciu o pewne dodatkowe informacje dostępne dla algorytmu wyszukiwania. Jedną z tych metod jest dwukierunkowe przeszukiwanie z zapytaniami o przynależność (ang. *bi-directional search using membership queries*) przedstawione przez Pharekh'a i Honavara [57], [59]. Metoda ta zakłada, że dostępny jest strukturalnie kompletny zbiór przykładów pozytywnych **S+** oraz 'nauczyciel' odpowiadający na pytanie, czy dany ciąg (słowo) należy do szukanego języka, czy też nie. Jeśli natomiast nie jest dostępny 'nauczyciel' znający gramatykę, ale dostępny jest zbiór przykładów negatywnych, możliwe jest przeszukiwanie podprzestrzeni przy pomocy algorytmu przeszukiwania w głąb z nawrotami [52]. Algorytm ten pozwala znaleźć **DAS** zgodny z zadanym zbiorem przykładów **S** w czasie wielomianowym. Dodatkowo, jeśli **S** jest nadzbiorem zbioru charakterystycznego, wtedy algorytm ten gwarantuje znalezienie kanonicznej reprezentacji szukanego **DAS**. Przez zbiór charakterystyczny składający się z przykładów pozytywnych **S+** i negatywnych **S-**

rozumie się taki zbiór, w którym **S+** jest strukturalnie kompletny, a **S-** zapobiega łączeniu dowolnych dwóch stanów **PAD** skonstruowanego na bazie **S+**, które nie są sobie równoważne (dokładna definicja znajduje się w pracy [52]).

Innym sposobem przeszukiwania podprzestrzeni automatów skończonych jest zastosowanie przeszukiwania losowego bazującego na technice algorytmów genetycznych. Algorytmy genetyczne, dla których inspiracją jest darwinowska teoria doboru naturalnego, pozwalają na ukierunkowane przeszukiwanie losowe przestrzeni rozwiązań [4], [27], [44].

Dupont w swojej pracy [14] zaprezentował, w jaki sposób można zastosować algorytmy genetyczne do przeszukiwania podprzestrzeni automatów skończonych powstałej na bazie **PAD**. W podejściu tym zakłada się, że dostępny jest zbiór przykładów pozytywnych **S+** oraz zbiór przykładów negatywnych **S-**. W oparciu o przykłady pozytywne **S+** tworzony jest **PAD,** a następnie przez losowe łączenie jego stanów konstruowana jest populacja początkowa. Funkcja dopasowania (celu) dla danego automatu skończonego jest zależna od dwóch parametrów: ilości stanów tego automatu oraz ilości błędnie zakwalifikowanych przykładów należących do **S-**. Automaty posiadające mniej stanów oraz źle klasyfikujące mniejszą ilość przykładów z **S-** uzyskują większą wartość funkcji dopasowania. Poprzez selekcję proporcjonalną do wartości funkcji dopasowania wybierana jest pula egzemplarzy stanowiąca bazę dla następnego pokolenia. Następnie egzemplarze te są poddawane operacji strukturalnego krzyżowania i mutacji, w wyniku czego powstaje następne pokolenie automatów skończonych. Po określonej ilości powtórzeń tego cyklu najlepszy automat (czyli taki, dla którego funkcja dopasowania osiąga największą wartość) traktowany jest jako rozwiązanie problemu. Mimo, że takie podejście nie gwarantuje, iż znalezione rozwiązanie będzie szukanym minimalnym **DAS**, to wyniki empiryczne pokazują, że znajdowane są **DAS** ze stosunkową małą ilością stanów oraz błędnie klasyfikujące tylko niewielką ilość przykładów należących do **S-**.

Innym algorytmem pozwalającym na znalezienie minimalnego **DAS** w oparciu o odpowiedzi udzielane przez najmniejszego adekwatnego 'nauczyciela' jest algorytm **L*** [3], [58]. W odróżnieniu od wcześniej przedstawionych algorytmów, **L*** nie przeszukuje podprzestrzeni automatów skończonych zbudowanej na bazie **PAD**, lecz tworzy poszukiwany **DAS** w oparciu o odpowiedzi na pytania zadawane 'nauczycielowi'. Dodatkowo 'nauczyciel' musi zapewnić kontrprzykłady pozwalające odróżnić poszukiwany **DAS** od aktualnie znalezionego przez algorytm. **L*** gwarantuje zbieżność w procesie poszukiwania **DAS**, a jego złożoność czasowa jest wielomianowa względem rozmiaru kanonicznej reprezentacji szukanego **DAS** oraz długości najdłuższego kontrprzykładu dostarczonego przez 'nauczyciela'.

Odmiennym podejściem do problemu generowania gramatyk na podstawie przykładów języka jest zastosowanie w tym procesie sztucznych sieci neuronowych. Zaletą tego podejścia w stosunku do technik symbolicznych opisanych wcześniej jest jego pewna odporność na błędy pojawiające się w przykładach uczących oraz możliwość zastosowania stosunkowo małych zbiorów uczących [38], [39]. Najczęściej używanym do celów generowania gramatyk typem sieci neuronowych są rekurencyjne sieci neuronowe (**RSN)**. Istnieje wiele prac, których przedmiotem badań było określenie przydatności w procesie generowania gramatyk na podstawie pozytywnych i negatywnych przykładów języka różnych rodzajów architektury **RSN** [8], [15], [16], [39], [63].

Istotnym problemem pojawiającym się w przypadku zastosowania sieci neuronowych do generowania gramatyk jest problem reprezentacji znalezionej gramatyki. W przypadku wytrenowanej sieci neuronowej gramatyka jest reprezentowana poprzez strukturę sieci oraz przez wagi dla poszczególnych neuronów, jednakże taka reprezentacja jest o wiele trudniejsza do interpretacji niż reprezentacja za pomocą **DAS** czy też za pomocą produkcji gramatyki. Stało się to motywacją do badań nad możliwością tworzenia deterministycznych automatów skończonych w oparciu o wytrenowane rekurencyjne sieci neuronowe [25], [51], [58]. Opracowana przez Gilesa metoda [25] wykorzystująca technikę klasteryzacji polega na podziale zakresu wyjściowego dla każdego neuronu na q jednakowych przedziałów, co pozwala mapować stan N neuronów na q^N stanów **DAS**. Tak utworzony **DAS** jest następnie minimalizowany (na przykład przy użyciu procedury przedstawionej w pracy [30]). Wyniki badań pokazują, że **DAS** otrzymany na bazie wytrenowanej rekurencyjnej sieci neuronowej jest dobrym przybliżeniem poszukiwanego **DAS**.

Mimo iż opracowano wiele algorytmów pozwalających na automatyczne wnioskowanie gramatyk regularnych, ich zastosowanie dla potrzeb syntaktycznej analizy i klasyfikacji wzorców jest ograniczone. Ograniczenia te dla większości przedstawionych algorytmów wynikają z konieczności zapewnienia dodatkowych informacji, bądź to w postaci odpowiedniej struktury danych (zbiór strukturalnie kompletny, zbiór charakterystyczny), bądź też poprzez dostęp do wiedzy 'nauczyciela' znającego poszukiwaną gramatykę. Dodatkowo ograniczenia języków regularnych powodują, iż często nie są one wystarczające dla potrzeb analizy i klasyfikacji wzorców w zastosowaniach praktycznych, takich jak, na przykład, modelowanie języków naturalnych, opis składni większości języków programowania czy też zastosowanie do celów szeroko rozumianego rozpoznawania i klasyfikacji wzorców, w tym, w szczególności, do rozpoznawania zmian chorobowych na podstawie różnych typów obrazów medycznych [45], [46], [73]. W przypadkach, dla których gramatyki regularne okazują się niewystarczające, konieczne jest zastosowanie gramatyk bezkontekstowych **GBK**, które stanowią następny poziom abstrakcji w hierarchii języków Chomsky'ego.

Ponieważ klasa gramatyk regularnych stanowi podzbiór klasy gramatyk bezkontekstowych, więc wszelkie teoretyczne ograniczenia w możliwościach wnioskowania gramatyk regularnych na podstawie przykładów języka przenoszą się na klasę gramatyk bezkontekstowych. Dodatkowe utrudnienie w zagadnieniu wnioskowania gramatyk bezkontekstowych stanowi fakt, że wiele problemów decyzyjnych dotyczących **GBK** jest nierozstrzygalnych. Przykłady takich nierozstrzygalnych problemów to: czy dane dwie gramatyki **G1** i **G2** są równoważne; czy gramatyka bezkontekstowa **G1** jest bardziej ogólna niż **G2** (tzn., czy język **L(G2)** zawiera się w języku **L(G1)**) oraz czy istnieje sentencja (słowo) wspólna dla danych dwóch języków **L(G1)** i **L(G2)** [30]. Powoduje to, że dokładne wygenerowanie szukanej gramatyki bezkontekstowej na ogół nie jest możliwe, istnieje natomiast kilka praktycznych metod heurystycznych pozwalających znajdować przybliżenia szukanej gramatyki.

Jednym ze sposobów generowania gramatyk bezkontekstowych na podstawie przykładów jest zastosowanie algorytmu GRIDS opracowanego przez Langleya [36]. Algorytm ten bazuje na reprezentacji gramatyki bezkontekstowej za pomocą rekursywnej sieci przejść (ang. *recursive transitions networks*) i jest podobny do techniki łączenia stanów dla problemu znajdowania **DAS**. Na początku w algorytmie tym na podstawie przykładów pozytywnych tworzona jest zdegenerowana jednopoziomowa sieć przejść akceptująca dokładnie słowa należące do zbioru uczącego. Jest ona odpowiednikiem prefiksowego automatu drzewiastego występującego w problemie znajdowania **DAS** dla gramatyk regularnych. Następnie algorytm znajduje powtarzające się wzorce występujące w początkowej sieci przejść i reprezentuje je jako nowe podsieci odpowiadające frazom w szukanym języku. Proces wyboru wzorców, które mają być reprezentowane jako podsieci, sterowany jest poprzez miary, takie jak, na przykład, złożoność podsieci, co pozwala na wybór najlepszej z możliwych alternatyw. Proces tworzenia podsieci jest kontynuowany do momentu, w którym dalszy podział wywołuje zwiększenie złożoności całej sieci. Następnie podobne podsieci są ze sobą łączone, co powoduje, że język rozpoznawany przez sieć przejść staje się bardziej ogólny. Jeśli dostępne są przykłady negatywne, stosuje się je do sterowania procesem łączenia tak, aby uniknąć nadmiernej generalizacji sieci.

Inny sposób generowania gramatyk bezkontekstowych opiera się na zastosowaniu rekurencyjnych sieci neuronowych w połączeniu z zewnętrznym stosem. Takie połączenie, zwane *recurrent neural network push down automata* **NNPDA,** pozwala jednak tylko na znajdowanie stosunkowo prostych deterministycznych gramatyk bezkontekstowych [12]. W celu poprawienia efektywności generowania gramatyk przez **NNPDA,** Das i pozostali badali wpływ dostarczenia dodatkowych informacji dla potrzeb tego modelu [13]. Informacje te były dwojakiego typu: pierwsze dotyczyły tylko zbioru przykładów uczących, natomiast drugie stanowiły częściową

wiedzę na temat szukanego automatu. Badania wykazały, że trenowanie przyrostowe, w którym przykłady o mniejszej długości są reprezentowane wcześniej niż przykłady dłuższe, znacząco zmniejsza czas uczenia **NNPDA**. Dodatkowo, jeśli dostępny jest 'nauczyciel' potrafiący dla przykładów negatywnych podać, który znak słowa powoduje wystąpienie stanu błędu, informacja ta może zostać użyta do zatrzymania procesu uczenia **NNPDA** dla dalszych znaków występujących w przykładzie negatywnym. Zastosowanie częściowej wiedzy na temat szukanego automatu pozwala natomiast na wstępne ustalenie wag dla sieci, co także zwiększa szybkość uczenia dla tej sieci [13].

Odmienne podejście do generowania gramatyk bezkontekstowych opiera się na zastosowaniu algorytmów genetycznych. Przykładem takiego rozwiązania jest przedstawiony przez Lankhorsta schemat uczenia dla niedeterministycznego automatu ze stosem na podstawie etykietowanego zbioru przykładów [37]. W metodzie zaprezentowanej przez Lankhorsta w chromosomie reprezentowanym przez ciąg bitów o stałej długości kodowana jest informacja o stałej, z góry określonej ilości przejść szukanego automatu. W procesie ewolucji stosowane są standardowe operacje krzyżowania i mutacji. Funkcja przystosowania (dopasowania) dla danego chromosomu, reprezentującego niedeterministyczny automat ze stosem, zależna jest zarówno od ilości poprawnie sklasyfikowanych przykładów, jak i od wielkości użytego stosu. Taka konstrukcja funkcji dopasowania pozwala preferować w procesie ewolucji prostsze automaty, używające mniejszego stosu. Jednak istotna słabość tej metody wynika z tego, że ilość przejść dla szukanego automatu jest z góry określona i zdeterminowana stałą długością chromosomu. Dlatego też kluczowym staje się wybór dotyczący maksymalnej ilości przejść w automacie, który musi zostać dokonany przed rozpoczęciem procesu ewolucji. Zbyt mała ilość ogranicza zdolność rozpoznawania języka przez szukany automat, natomiast zbyt duża znacznie zwiększa koszt czasowy i pamięciowy algorytmu klasyfikacji dla danego automatu. Problem ten może zostać rozwiązany poprzez zastosowanie techniki programowania genetycznego, w którym, w odróżnieniu od klasycznego podejścia stosowanego w algorytmach genetycznych, chromosom nie jest kodowany przez ciąg o stałej długości, lecz poprzez drzewo o zmiennej wielkości. Skuteczność takiego podejścia dla potrzeb generowania gramatyk bezkontekstowych była testowana przez autora, a wyniki przedstawiono w pracach [53], [54]. Jak zostało to wykazane, zastosowanie techniki programowania genetycznego w procesie generowania gramatyk bezkontekstowych pozwala na znajdowanie gramatyk zgodnych z zadanymi przykładami, przynajmniej dla stosunkowo prostych gramatyk. Mimo iż przedstawiona technika wymaga dalszych usprawnień, zanim będzie mogła zostać zastosowana w praktycznych zadaniach, takich jak, na przykład, generowanie gramatyk bezkontekstowych na potrzeby rozpoznawania i klasyfikacji obrazów medycznych, to otrzymane wyniki są na tyle obiecujące, iż pozwoliły sformułować tezę o następującej treści:

Możliwe jest opracowanie nowych ewolucyjnych algorytmów programowania genetycznego pozwalających na dokonanie automatycznej generacji gramatyk bezkontekstowych wykorzystywanych w zadaniach inteligentnej analizy i interpretacji wybranych wzorców.

Celem niniejszej pracy jest próba udowodnienia postawionej tezy, czyli opracowanie algorytmów bazujących na podejściu programowania genetycznego pozwalających na automatyczne generowanie gramatyk bezkontekstowych oraz sprawdzenie ich przydatności i ograniczeń dla celów analizy i interpretacji wybranych wzorców. W celu wykazania niniejszej tezy zostały przeprowadzone badania, które polegały na:

- sprawdzeniu skuteczności, różnych typów operatorów krzyżowania (takich jak krzyżowanie standardowe, krzyżowanie zachowujące rozmiar oraz krzyżowanie homologiczne) dla celów wnioskowania gramatyk bezkontekstowych;
- opracowaniu operatorów mutacji zapobiegających nadmiernemu wzrostowi generowanych gramatyk;
- opracowaniu oraz weryfikacji nowego modelu ewolucji, w którym nie występuje podział na pokolenia (ewolucja ciągła) charakterystyczny dla programowania genetycznego.

Prezentacja przeprowadzonych badań oraz uzyskanych wyników poprzedzona zostanie krótkim wprowadzeniem do teorii języków formalnych oraz omówieniem możliwości zastosowania języków formalnych dla potrzeb analizy i interpretacji wzorców. Następnie omówiona zostanie ogólna technika programowania genetycznego będącego rozszerzeniem podejścia ewolucyjnego stosowanego w algorytmach genetycznych. W kolejnym rozdziale przedstawiona zostanie koncepcja zastosowania podejścia ewolucyjnego opartego na technice programowania genetycznego dla potrzeb znajdowania gramatyk bezkontekstowych. Opisane zostaną, opracowane przez autora, algorytmy ewolucyjne pozwalające na automatyczne generowanie gramatyk bezkontekstowych na podstawie pozytywnych i negatywnych przykładów języka. W rozdziale piątym zaprezentowana zostanie implementacja oraz funkcjonalność systemu opracowanego przez autora dla potrzeb ewolucyjnego generowania gramatyk bezkontekstowych. W kolejnym rozdziale przedstawione zostanie porównanie wyników uzyskanych przy zastosowaniu różnych wariantów algorytmów opracowanych przez autora dla potrzeb generowania gramatyk bezkontekstowych. Pokazane zostanie, czy i w jakim stopniu algorytmy te są użyteczne w zastosowaniach praktycznych, takich jak

generowanie gramatyk dla języków opisujących zmiany uwidaczniane na pewnych rodzajach obrazów medycznych. Podsumowanie uzyskanych rezultatów oraz ocena stopnia realizacji założonego celu pracy, jak również nasuwające się wnioski dotyczące dalszych badań stanowić będą treść rozdziału siódmego zamykającego całą pracę.

2. Gramatyki i języki formalne

W rozdziale tym przedstawione zastaną podstawowe definicje i pojęcia z zakresu teorii języków formalnych, dotyczące w szczególności gramatyk ciągowych, które będą wykorzystywane w dalszej części pracy. Następnie omówiona zostanie możliwość zastosowania języków formalnych dla potrzeb analizy i interpretacji wzorców, a zwłaszcza do interpretacji obrazów medycznych [24], [46].

Definicja 2.1

Alfabetem V nazywamy dowolny skończony zbiór symboli.

Definicja 2.2

Słowem (lub łańcuchem) ***w*** nad alfabetem V nazywamy każdy skończony zbiór zestawionych ze sobą symboli należących do alfabetu V.

Definicja 2.3

Złożeniem (konkatenacją) dwóch łańcuchów jest łańcuch powstały przez wypisanie najpierw pierwszego, a następnie drugiego z nich bez żadnego ich rozdzielenia.

Definicja 2.4

Długością słowa *w* nazywamy liczbę symboli tworzących to słowo i oznaczamy ją przez |w|.

Definicja 2.5

Słowem pustym, oznaczanym symbolem ***ε***, nazywamy łańcuch nie posiadający żadnego symbolu. Tak więc długość słowa pustego $|\varepsilon| = 0$. Słowo puste stanowi element neutralny dla operacji składania, tj. $\varepsilon w = w\varepsilon = w$ dla każdego słowa *w*.

Definicja 2.6

Słownikiem V^+ nazywamy zbiór wszystkich niepustych słów nad alfabetem V.

Definicja 2.7

Słownikiem rozszerzonym nazywamy zbiór $V^* = V^+ \cup \{\varepsilon\}$.

Definicja 2.8

Gramatykę formalną G definiujemy jako czwórkę $G = (V_N, V_T, SP, STS)$, gdzie:

V_N – skończony zbiór symboli nieterminalnych

V_T – skończony zbiór symboli terminalnych (zakłada się, iż zbiory V_N i V_T są rozłączne)

SP – zbiór produkcji, przy czym każda produkcja ma postać $S_1 \rightarrow S_2$,

gdzie $S_1 \in (V_N \cup V_T)^+ - V_T^+$, $S_2 \in (V_N \cup V_T)^*$

STS – symbol startowy gramatyki, gdzie $STS \in V_N$

Definicja 2.9

Relacja bezpośredniej wyprowadzalności $\Rightarrow$ pomiędzy łańcuchami z $(V_N \cup V_T)^*$ dla danej gramatyki G. Jeśli $S_1 \rightarrow S_2$ jest produkcją należącą do zbioru SP gramatyki G, a α i β są dowolnymi łańcuchami z $(V_N \cup V_T)^*$, to $\alpha S_1 \beta \Rightarrow \alpha S_2 \beta$. Mówimy, że $\alpha S_2 \beta$ jest bezpośrednio wyprowadzalny z $\alpha S_1 \beta$ dla gramatyki G.

Definicja 2.10

Relacja wyprowadzalności =*> pomiędzy łańcuchami z $(V_N \cup V_T)^*$ dla danej gramatyki G. Jeśli $\alpha_1, \alpha_2, ..., \alpha_m$ są łańcuchami z $(V_N \cup V_T)^*$, $m \geq 1$ oraz $\alpha_1 \Rightarrow \alpha_2$, $\alpha_2 \Rightarrow \alpha_3$, ..., $\alpha_{m-1} \Rightarrow \alpha_m$ to wtedy α_1 =*>α_m. Mówimy, że α_m jest wyprowadzalny z α_1 dla gramatyki G.

Definicja 2.11

Językiem generowanym przez gramatykę G nazywamy zbiór:

$L(G) = \{ w: w \in V_T^* \text{ i } STS =*> w \}$

a zatem słowo *w* należy do L(G), jeśli *w* składa się wyłącznie z symboli terminalnych oraz *w* jest wyprowadzalne z STS.

2.1. Hierarchia gramatyk według Chomsky'ego

W zależności od postaci produkcji gramatyki można podzielić na cztery kategorie:

Typ 0 – gramatyki nieograniczone (struktur frazowych)

Dla tego typu gramatyk nie ma ograniczenia na postać produkcji, to znaczy dozwolone są produkcje o postaci $\alpha \rightarrow \beta$, gdzie α i β są dowolnymi łańcuchami symboli danej gramatyki, przy czym $\alpha \neq \varepsilon$.

Gramatyki tego typu są na ogół zbyt ogólne do zastosowań praktycznych, tzn. w ogólnym przypadku problem, czy dane słowo może być wygenerowane przez określoną gramatykę tego typu, jest nierozstrzygalny.

Gramatyki nieograniczone pozwalają opisywać języki rekurencyjnie przeliczalne [30].

Typ 1 – gramatyki kontekstowe

Produkcje gramatyk tego typu mają postać $\alpha_1 A \alpha_2 \rightarrow \alpha_1 b \alpha_2$, gdzie $A \in (V_N \cup V_T)^*$, $b \in (V_N \cup V_T)^* - \{\varepsilon\}$, $\alpha_1, \alpha_2 \in (V_N \cup V_T)^*$, przy czym $| \alpha_1 A \alpha_2| < |\alpha_1 b \alpha_2|$, czyli $|A| < |b|$.

Typ 2 – gramatyki bezkontekstowe

W przypadku gramatyk bezkontekstowych wszystkie produkcje mają postać: $A \rightarrow b$, gdzie $A \in V_N$, $b \in (V_N \cup V_T)^*$.

Typ 3 – gramatyki regularne

Dla gramatyk prawostronnie regularnych (liniowych) produkcje mogą mieć postać $A \rightarrow wB$ lub $A \rightarrow w$, gdzie $A, B \in V_N$, natomiast w jest słowem (być może pustym) składającym się wyłącznie z symboli terminalnych.

W przypadku gramatyk lewostronnie liniowych produkcje mogą mieć następującą postać: $A \rightarrow Bw$ lub $A \rightarrow w$, gdzie A, B i w są określone jak powyżej.

Gramatyki lewostronnie liniowe i prawostronnie liniowe zwane są gramatykami regularnymi.

Hierarchia gramatyk Chomsky'ego skonstruowana jest w ten sposób, iż zbiory języków generowanych przez gramatyki wyższego typu są zawarte w zbiorach języków generowanych przez gramatyki niższego typu. Prawdziwe są następujące twierdzenia:

Twierdzenie 2.1

Klasa języków generowanych przez gramatyki typu 3 – czyli zbiorów regularnych jest ściśle zawarta w klasie języków bezkontekstowych (języków generowanych przez gramatyki typu 2).

Bez dowodu.

Twierdzenie 2.2

Klasa języków bezkontekstowych nie zawierających łańcucha pustego jest ściśle zawarta w klasie języków kontekstowych (generowanych przez gramatyki typu 1).

Bez dowodu.

Twierdzenie 2.3

Klasa języków kontekstowych jest ściśle zawarta w klasie zbiorów rekurencyjnie przeliczalnych (opisywanych przez gramatyki typu 0)

Dowody tych twierdzeń wynikają z konstrukcji poszczególnych typów gramatyk i są omówione w pracy [30].

Bez dowodu.

2.2. Analizatory syntaktyczne dla gramatyk bezkontekstowych

Podstawowym zadaniem analizatora syntaktycznego (parsera) jest określenie, czy zadany ciąg symboli terminalnych tworzących słowo *w* jest wyprowadzalny z symbolu startowego przy użyciu produkcji dla zadanej gramatyki.

Istnieją dwa podstawowe sposoby takiej analizy:

- zstępujący (ang. top – down) – w tym przypadku proces analizy rozpoczyna się od symbolu startowego, który przekształcany jest przy użyciu produkcji gramatyk tak, aby otrzymać wejściowe słowo (ciąg symboli terminalych); przykładem zastosowania takiego typu analizy jest parser LL;
- wstępujący (ang. bottom – up) – parser rozpoczyna analizę od słowa wejściowego, które przy użyciu produkcji gramatyki próbuje przekształcić tak, aby otrzymać symbol startowy; przykładami analizatorów syntaktycznych tego typu są:
  - CYK (Cocke-Younger-Kasami) parser,
  - Tomita parser,
  - parsery typu LR (takie jak SLR, LALR, GLR).

Jednym z najczęściej stosowanych typów analizatorów składniowych dla gramatyk bezkontekstowych są analizatory klasy LALR(1). Wynika to przede wszystkim z ich efektywności, z dość dużej ogólności (wiele języków bezkontekstowych posiada gramatykę klasy LALR(1)) oraz dostępności narzędzi służących do automatycznego generowania tego typu analizatorów, takich jak YACC czy Bison.

Aby możliwe było zastosowanie analizatora syntaktycznego klasy LALR, zadana gramatyka musi być deterministyczna w tym sensie, że podczas analizy wejściowego ciągu symboli można zawsze jednoznacznie określić, którą produkcję gramatyki należy zastosować, podglądając tylko jeden kolejny symbol wejściowy [1].

Zastosowanie analizatora syntaktycznego klasy LALR(1) wiąże się z pewnymi trudnościami; podstawowa z nich polega na tym, że nie wszystkie języki bezkontekstowe mogą zostać wygenerowane przy użyciu gramatyki klasy LALR(1). Jako że gramatyka ta musi być jednoznaczna dla wielu zastosowań praktycznych, takich jak na przykład syntaktyczny opis niektórych języków programowania (na przykład C++), gramatyki klasy LALR(1) są niewystarczające. Jednak nawet jeśli dany język bezkontekstowy posiada gramatykę klasy LALR(1), to jej znalezienie (czyli przekształcenie dowolnej gramatyki bezkontekstowej tak, aby spełniała ograniczenia klasy LALR(1)) może być skomplikowane i czasochłonne. Stanowi to istotną trudność, szczególnie w przypadku, gdy gramatyki generowane są w sposób automatyczny w procesie wnioskowania gramatyk. Rozwiązaniem tych problemów może być zastosowanie bardziej ogólnego analizatora syntaktycznego klasy GLR (Generalized LR) lub analizatora bazującego na algorytmie CYK, zamiast analizatora klasy LALR(1).

Algorytm analizatora syntaktycznego GLR stanowi rozszerzenie algorytmu LR, w którym stos analizatora jest zastąpiony przez graf zawierający wszystkie możliwe konfiguracje stosu, które mogłyby wystąpić dla parsera LR [40], [41], [64]. Każda taka konfiguracja traktowana jest następnie jako oddzielny, potencjalny analizator LR. Ponieważ algorytm analizatora syntaktycznego GLR musi operować na bardziej złożonych strukturach danych niż algorytm LR, jego efektywność nawet dla deterministycznych gramatyk jest o rząd wielkości mniejsza niż w przypadku efektywnego algorytmu LALR (takiego jak, na przykład, analizator wygenerowany przy użyciu generatora Bison) [41]. Poprawę skuteczności algorytmu analizatora syntaktycznego można uzyskać poprzez zastosowanie rozwiązania hybrydowego, które polega na dynamicznym przełączaniu pomiędzy algorytmem GLR i LALR(1) na poziomie interpretacji poszczególnych symboli wejściowych. Takie hybrydowe podejście zostało na przykład zastosowane w generatorze analizatorów syntaktycznych Elkhound [40], [41], wykorzystanym w badaniach zaprezentowanych w dalszej części pracy.

2.3. Zastosowanie języków formalnych do celów analizy i interpretacji wzorców

Metody syntaktyczne stanowią alternatywę w dziedzinie rozpoznawania i interpretacji wzorców dla metod bazujących na podejściu decyzyjno–teoretycznym. W przypadku tego podejścia na podstawie zadanego zbioru wzorców wybierany jest zbiór

miar charakterystycznych, zwanych cechami. Następnie rozpoznawanie nieznanego wzorca, czyli przypisanie go do odpowiedniej klasy wzorców, dokonywane jest zazwyczaj na podstawie pewnego podziału przestrzeni cech. Podział taki może odbywać się na podstawie rozważań natury geometrycznej przez wprowadzenie pewnej metryki (metody minimalno–odległościowe), przez zdefiniowanie operacji analogicznych do operacji używanych w metodach aproksymacji funkcji (metody aproksymacyjne) lub też na bazie rachunku prawdopodobieństwa (metody statystyczne) [24], [69]. Jednak już w latach sześćdziesiątych okazało się, że w przypadku wielu problemów z dziedziny rozpoznawania wzorców metody bazujące na podejściu decyzyjno–teoretycznym są mało skuteczne. Problem ten dotyczy wzorców, a w szczególności obrazów, które są albo bardzo złożone, albo dla których liczba klas jest bardzo duża, co powoduje znaczne zwiększenie wymiaru przestrzeni cech użytych do klasyfikacji wzorców [20]. Przykładami takich problemów, dla których metody decyzyjno–teoretyczne okazują się mało skuteczne, są: problem identyfikacji twarzy i odcisków palców, rozpoznawanie mowy, identyfikacja znaków pisma ideograficznego (na przykład pisma chińskiego) [24] czy też problem diagnostyki zmian chorobotwórczych występujących na różnego rodzaju obrazach medycznych [45], [71], [73]. W takich przypadkach znacznie efektywniejsze jest zastosowanie w procesie identyfikacji wzorców metod syntaktycznych. W podejściu syntaktycznym złożony wzorzec (taki jak, na przykład, obraz) dzielony jest na prostsze podwzorce, które próbuje się rozpoznać metodami całościowymi, traktując je jako niezależne, a następnie, na podstawie rozpoznanych składowych pierwotnych i relacji pomiędzy nimi, dokonuje się rozpoznania wzorca na podstawie przynależności do pewnego języka formalnego [20].

W przypadku zastosowania metod syntaktycznych do celów analizy obrazów należy najpierw wyodrębnić składowe pierwotne oraz relacje pomiędzy nimi. Najczęściej spotykanym sposobem wyodrębnienia składowych pierwotnych jest segmentacja obrazu, na przykład metodami prostego progowania histogramowego, wykrywania krawędzi z zastosowaniem operatorów gradientowych czy zastosowania dedykowanych dla poszczególnych rodzajów zobrazowań procedur segmentacji.

Następnym niezwykle ważnym etapem jest identyfikacja relacji pomiędzy składowymi pierwotnymi. Ponieważ składowe pierwotne traktujemy jako niezależne obrazy, więc, aby cały obraz mógł zostać prawidłowo złożony z podobrazów, a tym samym aby mógł zostać prawidłowo rozpoznany, konieczna jest prawidłowa identyfikacja relacji pomiędzy tymi podobrazami [69]. Do reprezentacji obrazów używane są trzy podstawowe grupy metod formalnych:

Pierwszą stanowią metody ciągowe – w ich przypadku obraz reprezentowany jest jako ciąg. Oznacza to, iż między składowymi pierwotnymi występuje tylko jeden rodzaj relacji – konkatenacji (złożenia) kolejnego elementu. Przykładami języków

ciągowych są: języki oparte na kodach łańcuchowych Freemana, języki opisu obrazów Shawa [66] oraz języki opisu cech kształtów Jakubowskiego [20]. Mimo iż metody ciągowe stanowią najprostszą klasę metod syntaktycznego opisu wzorców, są one wystarczające dla wielu zastosowań praktycznych, w tym też, na przykład, do opisu różnego rodzaju schorzeń na podstawie zobrazowań medycznych [47], [50]. Badania prezentowane w niniejszej rozprawie będą w całości oparte na tej właśnie metodzie reprezentacji wzorców.

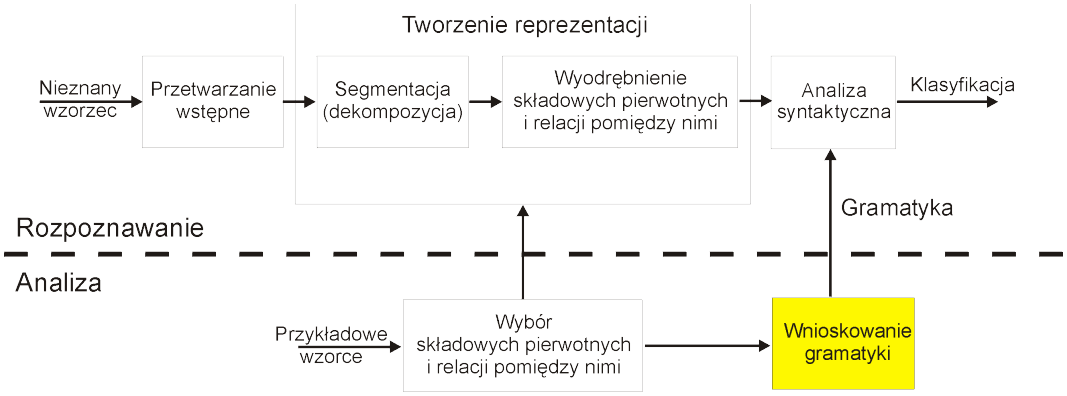
Drugą klasą języków służących do reprezentacji obrazów są języki drzewowe. W ich przypadku obraz (wzorzec) reprezentowany jest jako drzewo złożone z podobrazów. W odróżnieniu od języków ciągowych języki drzewowe pozwalają na opis obrazów wieloobiektowych. Do najczęściej spotykanych zastosowań tych języków należy analiza scen i analiza faktur.

Trzecią, najbardziej ogólną klasę stanowią metody grafowe – w tym przypadku obraz (wzorzec) reprezentowany jest za pomocą grafu. Ponieważ jednak graf jest formalizmem bardziej ogólnym niż drzewo, języki tego typu charakteryzują się większą ogólnością niż języki drzewowe. Przykładem takich formalizmów są gramatyki grafowe klasy ETPL(k) [17], [18], [19], które pozwalają również na rozpoznawanie obrazów zniekształconych [67], [68].

Syntaktyczne metody rozpoznawania wzorców opierają się na wspólnym założeniu, że możliwe jest zdefiniowanie pewnego mechanizmu generującego odpowiednie reprezentacje (ciągowe, drzewowe lub grafowe) dla rozpoznawanych wzorców. Mechanizmem tym jest gramatyka formalna G (odpowiednio: ciągowa, drzewowa lub grafowa), zaś zbiór wszystkich reprezentacji obrazów generowanych przez nią traktowany jest jako pewien język formalny L(G) [24], [69].

Posiadając odpowiednią reprezentację wzorca (na przykład obrazu), można za pomocą analizatora syntaktycznego sprawdzić, czy wzorzec ten należy do poszukiwanej klasy wzorców określonej za pomocą pewnej gramatyki **G,** czyli czy dana reprezentacja należy do pewnego języka formalnego.

Na rysunku 2.1 przedstawiony został schemat blokowy kompletnego systemu syntaktycznego rozpoznawania wzorców [24].



Rysunek 2.1. Schemat blokowy systemu syntaktycznego rozpoznawania wzorców

Przedstawiony system składa się z dwóch podstawowych części funkcjonalnych: części analizującej znane, przykładowe, wzorce oraz części rozpoznającej, bazującej na wynikach analizy, której zadaniem jest klasyfikacja nieznanych wzorców. Zadaniem części analizującej systemu jest wybór na podstawie przykładowych wzorców składowych pierwotnych oraz relacji pomiędzy nimi. Następnie na podstawie dokonanego wyboru w procesie wnioskowania gramatyk, znajdowana jest właściwa gramatyka będąca podstawą dla analizy syntaktycznej nieznanych wzorców. Opracowanie efektywnych algorytmów bazujących na podejściu programowania genetycznego i pozwalających na efektywne wnioskowania gramatyk stanowi zasadniczą część niniejszej rozprawy. Po określeniu składowych pierwotnych, relacji pomiędzy nimi oraz gramatyki będącej podstawą klasyfikacji wzorców możliwe jest przejście od analizy do właściwego procesu rozpoznawania wzorców.

Proces ten składa się z następujących etapów:

a) przetwarzanie wstępne – na etapie tym dokonywana jest filtracja sygnału i usunięcie z niego szumów tak, aby na wyjściu z tego etapu otrzymać wzorce charakteryzujące się dobrą jakością [70];

b) segmentacja – w etapie tym wzorzec podlega podziałowi na mniejsze podwzorce tak, aby w następnym etapie mogły zostać zidentyfikowane składowe pierwotne wchodzące w ich skład;

c) wyodrębnienie składowych pierwotnych i relacji pomiędzy nimi – etap ten ma na celu identyfikację składowych pierwotnych występujących w poszczególnych podwzorcach i relacji pomiędzy nimi;

d) analiza syntaktyczna – w etapie tym następuje właściwa analiza syntaktyczna zadanego wzorca. Sprawdzane jest, czy utworzona w poprzednim etapie struktura zbudowana ze składowych pierwotnych i relacji pomiędzy nimi należy do języka generowanego przez zadaną gramatykę, czy też nie. Tak jak to zostało opisane

wcześniej, sprawdzenie takie odbywa się na ogół za pomocą analizatora syntaktycznego (parsera).

Często w przypadku analizy syntaktycznej obrazów etap tworzenia reprezentacji wymaga wykonania dodatkowych czynności. Na przykład w przypadku analizy cyfrowych obrazów medycznych etap tworzenia reprezentacji przedstawiony w pracy [46] składa się z: segmentacji i filtracji, szkieletyzacji analizowanych struktur, analizy rzeczywistych i weryfikacji pozornych odgałęzień szkieletowych, wygładzenia szkieletu przez uśrednienie jego elementów oraz zastosowania transformacji prostującej, przekształcającej zewnętrzny kontur badanych struktur w przestrzeni dwuwymiarowej do postaci dwuwymiarowych wykresów szerokości uwidaczniających kontury wyprostowanych organów. Dopiero tak uzyskane wykresy szerokości, po wyodrębnieniu składowych pierwotnych i relacji między nimi (w tym przypadku relacja konkatenacji), stanowią dane wejściowe dla modułu generującego gramatyki i modułu właściwej analizy syntaktycznej.

W rozdziale tym przedstawiono podstawowe definicje dotyczące gramatyk i języków formalnych, które będą stosowane w dalszej części pracy. Omówiony został też ogólny schemat systemu syntaktycznego rozpoznawania wzorców z uwzględnieniem roli podsystemu wnioskowania (generowania) gramatyk. W następnym rozdziale przedstawiona zostanie pokrótce idea programowania genetycznego, natomiast w rozdziale czwartym, w oparciu o formalizm przedstawiony powyżej, omówione będzie zastosowanie podejścia bazującego na programowaniu genetycznym dla celów wnioskowania gramatyk bezkontekstowych na podstawie pozytywnych i negatywnych przykładów języka.

3. Programowanie genetyczne jako przykład formalizmów ewolucyjnych

Rozdział ten zawiera krótkie omówienie podstaw programowania genetycznego, które stanowi szczególną odmianę algorytmów genetycznych charakteryzującą się zastosowaniem kodowania drzewiastego do reprezentacji programu podlegającego ewolucji. Przedstawiony opis dotyczyć będzie standardowej postaci programowania genetycznego, czyli takiej, jaką przedstawił Koza w pracy [33]. Natomiast sposób zaadaptowania programowania genetycznego dla celów generowania gramatyk na podstawie przykładów języka zostanie szerzej omówiony w rozdziale 4.

3.1. Algorytmy genetyczne – wprowadzenie

Algorytmy genetyczne (GA) stanowią rodzaj algorytmów przeszukujących przestrzeń alternatywnych rozwiązań zadanego problemu w celu znalezienia najlepszych rozwiązań ([4], [29], [27], [44]). Sposób działania algorytmów genetycznych oraz terminologia służąca do ich opisu inspirowane są zjawiskiem ewolucji biologicznej oraz genetyką [29].

W przypadku algorytmów ewolucyjnych, których szczególną odmianę stanowią algorytmy genetyczne, zadany do rozwiązania problem definiuje środowisko, w którym istnieje pewna populacja osobników reprezentujących potencjalne rozwiązania zadanego problemu. Każdy z osobników należących do populacji posiada przypisany pewien zbiór informacji stanowiący jego genotyp, który jest podstawą utworzenia fenotypu, czyli zestawu cech podlegających ocenie środowiska. Wartość liczbowa tej oceny nazywana jest przystosowaniem (lub dopasowaniem) osobnika i stanowi miarę poprawności rozwiązania zadanego problemu przez określonego osobnika.

Środowisko, które jest zdefiniowane przez problem podlegający rozwiązaniu, może zostać przedstawione jako pewna funkcja przystosowania; odwzorowuje ona fenotyp na przystosowanie osobnika. Ponieważ jednak fenotyp jest zakodowany w genotypie (a w niektórych schematach ewolucji genotyp jest wprost tożsamy z fenotypem), przez funkcję przystosowania można rozumieć bezpośrednio odwzorowanie genotypu na przystosowanie osobnika.

Genotyp danego osobnika złożony jest z jednego lub większej ilości chromosomów, przy czym przynajmniej jeden z tych chromosomów zawiera informacje o fenotypie osobnika. Natomiast chromosom składa się z poszczególnych genów. W przypadku typowych algorytmów genetycznych chromosom osobnika złożony jest ze stałej ilości genów, które mogą posiadać wartość zero lub jeden. W takim wypadku

proponowane rozwiązanie, czyli genotyp osobnika, może zostać przedstawione jako ciąg bitowy o określonej, stałej długości. Wadą takiego rozwiązania jest to, że długość ciągu bitowego musi zostać zadana *a priori*. Ma to istotne znaczenie zwłaszcza w przypadku problemów, dla których postać rozwiązania nie jest z góry znana. Przyjęcie zbyt małej długości ciągu bitowego może spowodować, że nie będzie on w stanie przedstawić poszukiwanego rozwiązania, natomiast przyjęcie zbyt dużej długości znacznie zwiększa czas potrzebny na znalezienie rozwiązania.

Działanie algorytmu ewolucyjnego oparte jest na powtarzającym się cyklu, w którym wykonywane są: ocena przystosowania osobników, selekcja osobników, operacje genetyczne oraz tworzenie nowego pokolenia osobników. Selekcja osobników polega na wyborze z populacji bieżącego pokolenia osobników, które utworzą pulę rodzicielską stanowiącą bazę dla nowego pokolenia. Odbywa się ona w sposób losowy, uwzględniający jednak wartości funkcji przystosowania. W ten sposób osobniki o większej wartości przystosowania mają większe szanse wyboru do puli rodzicielskiej (czyli większą wartość oczekiwaną liczby kopii w tej puli). Zatem średnia wartość funkcji przystosowania osobników wybranych do puli rodzicielskiej jest na ogół większa niż średnia wartość przystosowania osobników w bieżącym pokoleniu, z którego zostały one wybrane.

Następnie osobniki, które znalazły się w populacji rodzicielskiej poddawane są działaniu operatorów genetycznych, do których należą:

- **Reprodukcja** – wybrany osobnik jest bezpośrednio kopiowany z populacji rodzicielskiej do populacji tworzącej nowe pokolenie. Operator ten działa na pojedynczym osobniku rodzicielskim.
- **Krzyżowanie** – operator działający na wielu (na ogół dwóch) osobnikach rodzicielskich. Wynikiem jego działania jest wygenerowanie jednego lub wielu osobników, których chromosomy powstają przez złączenie odpowiednich fragmentów chromosomów należących do osobników rodzicielskich. W wyniku działania tego operatora mogą powstawać nowe osobniki, o większym przystosowaniu niż osobniki rodzicielskie.
- **Mutacja** – podobnie jak reprodukcja jest to operator działający na pojedynczym osobniku rodzicielskim. Jego działanie polega na losowych zmianach genotypu osobnika. W ten sposób mogą powstawać osobniki o większym przystosowaniu niż osobnik przed mutacją.

Osobniki powstałe w wyniku działania operatorów genetycznych tworzą nowe pokolenia i cały cykl ocena – selekcja – operacje genetyczne – tworzenie nowego pokolenia jest powtarzany aż do czasu spełnienia warunków zakończenia, takich, jak

znalezienie odpowiednio dokładnego rozwiązania lub przekroczenie zadanej ilości tworzonych pokoleń.

3.2. Programowanie genetyczne

3.2.1. Rozwój algorytmów programowania genetycznego

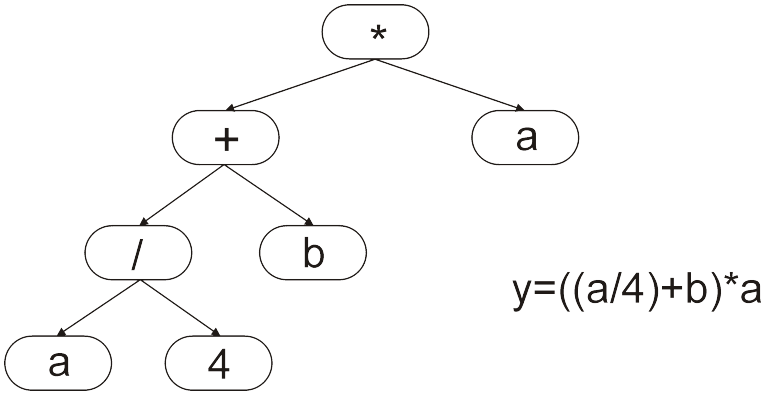
Programowanie genetyczne (ang. *Genetic Programming* **GP**) stanowi szczególną odmianę algorytmów genetycznych, w której genotyp osobnika reprezentuje pewien program podlegający ocenie środowiska. Mimo iż technika programowania genetycznego upowszechniła się i znalazła liczne zastosowania praktyczne właściwie dopiero w przeciągu ostatnich 10 – 15 lat, jej początki sięgają roku 1958 [2], kiedy to Friedberg [22], a następnie w rok później Friedberg, Dunham i North [23] zaprezentowali możliwość automatycznego modyfikowania programów tak, aby jak najlepiej rozwiązywały zadany problem. Następna, bardzo znacząca z punktu widzenia historii rozwoju programowania genetycznego, jest praca autorstwa Fogel, Owens i Walsh z roku 1966. Została w niej przedstawiona idea programowania ewolucyjnego, w którym do rozwiązania postawionego problemu używa się automatów skończonych podlegających ewolucji. W roku 1975 Holland przedstawił algorytmy genetyczne charakteryzujące się użyciem do reprezentacji chromosomu ciągów binarnych o stałej długości oraz zastosowaniem operacji krzyżowania. W 1985 roku Cramer jako pierwszy zastosował operację krzyżowania poddrzew dla potrzeb ewolucji w języku TB (*tree based*). Cztery lata później Koza zaprezentował w swojej pracy [34] zastosowanie języka LISP w połączeniu z reprezentacją programu za pomocą drzewa do rozwiązania zadanych problemów, takich jak regresja symboliczna (ang. *symbolic regression*) czy problem planowania. Praca ta wprowadziła podstawy tego, co obecnie uznawane jest za paradygmat programowania genetycznego, który dokładniej Koza opisał w pracy [33] z roku 1992.

3.2.2. Podstawy działania programowania genetycznego

Reprezentacja programu

Jak zostało to już wcześniej wspomniane, cechą charakterystyczną standardowej postaci GP jest reprezentacja programów podlegających ewolucji za pomocą struktur drzewiastych. Przykład takiej reprezentacji programu znajduje się na rysunku 3.1. Przedstawione na nim drzewo reprezentuje wyrażenie (które jest rozumiane jako określony program) $y = ((a/4)+b)*a$, gdzie a oraz b są pewnymi zmiennymi. W

przedstawionym drzewie wierzchołki z etykietami *a*, *b* oraz *4* są liśćmi, natomiast wierzchołki +, / oraz * są wierzchołkami wewnętrznymi.



Rysunek 3.1. Reprezentacja przykładowego programu za pomocą drzewa stosowana w standardowej postaci programowania genetycznego

W standardowej postaci programowania genetycznego program, który reprezentowany jest za pomocą drzewa, składa się z jednego lub większej ilości wierzchołków (węzłów). Wierzchołki te pochodzą z dwóch rozłącznych zbiorów: zbioru funkcji $\mathcal{F}$ oraz zbioru terminali $\mathcal{T}$. Przy czym wierzchołki wewnętrzne pochodzą z pierwszego z tych zbiorów, do którego należą funkcje posiadające jeden lub więcej argumentów – czyli wierzchołki reprezentujące te funkcje posiadają jednego lub więcej synów (reprezentujących ich argumenty). Natomiast liście drzewa reprezentującego program wybierane są ze zbioru $\mathcal{T}$, w skład którego wchodzą wartości stałe, zmienne oraz funkcje bezargumentowe[1].

Jeśli potraktuje się wszystkie elementy zbioru $\mathcal{T}$ jako funkcje bezargumentowe, to każdy wierzchołek w drzewie reprezentującym program należy do pewnego zbioru funkcji $\mathcal{C} = \mathcal{F} \cup \mathcal{T}$. W takim przypadku obszar poszukiwań (ang. *search space*) jest zbiorem wszystkich możliwych kompozycji elementów $\mathcal{C}$. Zbiór ten musi być domknięty i wystarczający do rozwiązania danego problemu. Domknięcie oznacza w tym przypadku, iż każdy element (funkcja) ze zbioru $\mathcal{C}$ musi akceptować jako swoje argumenty dowolne inne elementy (funkcje) należące do zbioru $\mathcal{C}$. Zapewnia to, że dowolny program zbudowany z funkcji należących do zbioru $\mathcal{C}$ nie będzie zawierał błędów[2]. Jednym ze sposobów zapewnienia domknięcia stosowanym w GP jest ograniczenie argumentów i wartości zwracanych przez funkcje do typów zgodnych ze sobą. Natomiast to, że zbiór $\mathcal{C}$ jest wystarczający do rozwiązania problemu, oznacza, iż funkcje wchodzące w jego skład pozwalają na zbudowanie programu rozwiązującego

1 Wartości stałe oraz zmienne można traktować jako szczególnym przypadek funkcji bezargumentowych.

2 Chodzi tutaj o błędy składniowe i błędy wykonania (ang. *run time errors*).

zadany problem. Oczywiście to, jaki zbiór C jest wystarczający, zależy od problemu, który ma zostać rozwiązany.

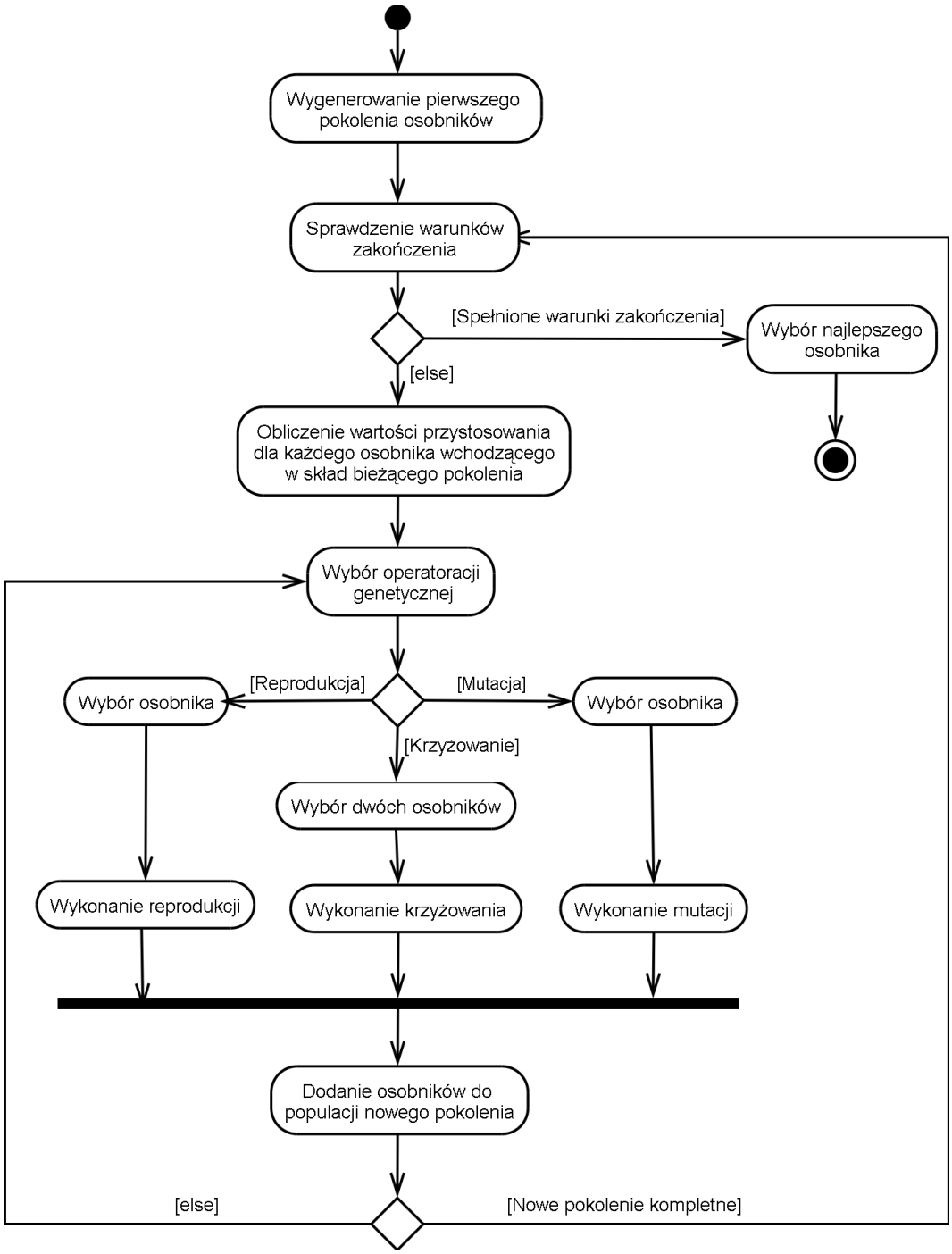
Wyznaczenie tego zbioru stanowi etap wstępny, poprzedzający rozwiązanie zadanego problemu za pomocą programowania genetycznego. Właściwy wybór tego zbioru jest niezwykle istotny dla całego procesu programowania genetycznego - z jednej strony, w zbiorze tym nie powinny znajdować się funkcje, które nie są potrzebne do rozwiązania zadanego problemu, gdyż zwiększenie ilości funkcji zwiększa czas potrzebny do znalezienia poprawnego rozwiązania (programu), z drugiej jednak strony, jeśli rozwiązanie zadanego problemu wymaga funkcji, których nie ma w zbiorze C, to poprawne rozwiązanie nigdy nie zostanie znalezione. Odpowiedni wybór zbioru C stanowi jeden z największych problemów, na jakie napotyka się w przypadku korzystania z techniki programowania genetycznego.

Schemat działania GP

Ponieważ programowanie genetyczne stanowi pewną odmianę algorytmów ewolucyjnych, więc sposób jego działania odpowiada przedstawionemu wcześniej ogólnemu sposobowi działania algorytmów ewolucyjnych i składa się z następujących faz:

1) Utworzenie pierwszego pokolenia. Początkowe osobniki (programy) tworzone są w sposób losowy z elementów zbioru funkcji $\mathcal{F}$ i zbioru terminali $\mathcal{T}$.
2) Obliczenie wartości przystosowania. Każdy program należący do bieżącego pokolenia jest wykonywany i, na podstawie osiągniętych przez niego rezultatów, obliczana jest dla niego wartość funkcji przystosowania.
3) Utworzenie nowego pokolenia. Wybierane są najlepsze osobniki z bieżącego pokolenia, które poddawane są działaniu następujących operatorów genetycznych w celu utworzenia nowego pokolenia programów:
   - reprodukcji,
   - krzyżowania,
   - mutacji.
4) Następnie fazy 2 i 3 są powtarzane tak długo, aż osiągnięta zostanie założona ilość pokoleń lub zostanie znaleziony program spełniający ustalone warunki na wartość przystosowania, czyli taki, który rozwiązuje zadany problem z określoną dokładnością.

Na rysunku 3.2 przedstawiony został diagram czynności obrazujący sposób działania programowania genetycznego.



Rysunek 3.2. Diagram czynności przedstawiający sposób działania programowania genetycznego

Poniżej znajduje się krótkie omówienie czynności przedstawionych na diagramie.

Tworzenie pokolenia początkowego

Programy wchodzące w skład pierwszego pokolenia tworzone są w sposób losowy z funkcji i terminali należących odpowiednio do zbiorów $\mathcal{F}$ i $\mathcal{T}$. Budowane są drzewa reprezentujące programy, których wierzchołki wewnętrzne należą do zbioru funkcji $\mathcal{F}$, natomiast ilość synów dla tych wierzchołków jest określona przez liczbę

argumentów poszczególnych funkcji. Do utworzenia liści w generowanych drzewach wykorzystywane są elementy zbioru terminali $\mathcal{T}$.

Istnieją różne sposoby tworzenia drzew z elementów zbiorów $\mathcal{F}$ i $\mathcal{T}$; trzy podstawowe metody, które przedstawił Koza, to: *full*, *grow* oraz *ramped half and half*. Metoda ***full*** polega na losowym wybieraniu wierzchołków ze zbioru $\mathcal{F}$ aż do osiągnięcia zadanej głębokości, a następnie na wyborze wierzchołków ze zbioru terminali $\mathcal{T}$. Wszystkie powstałe w ten sposób drzewa mają taką samą, z góry zadaną, wysokość. Metoda ***grow*** różni się od metody *full* tym, iż do czasu osiągnięcia zadanej głębokości drzewa wierzchołki wybierane są ze zbioru $\mathcal{C}$, a nie ze zbioru $\mathcal{F}$. Powstałe w ten sposób drzewa mogą mieć różną wysokość (jednak nie większą niż maksymalna zadana). Metoda ***ramped half and half*** stanowi kombinację metod *full* i *grow.* Polega ona na utworzeniu jednakowej ilości drzew o wysokości między 2 a określoną maksymalną wysokością drzewa. Jeśli na przykład określona maksymalna wysokość wynosi 6 to 1/5 (20%) drzew będzie miała zadaną wysokość 2, 1/5 – wysokość 3 itd. aż do wysokości 6. Następnie dla każdej wysokości 50% drzew tworzonych jest za pomocą metody *full,* a 50% za pomocą metody *grow.* Metoda *ramped half and half* zapewnia uzyskanie dużej różnorodności kształtów i wielkości drzew wchodzących w skład wygenerowanej populacji.

Porównanie wydajności działania GP dla przedstawionych metod generowania pokolenia początkowego zawarte zostało w pracy [33], a zaprezentowane wyniki pokazują, iż zastosowanie metody *ramped half and half* daje największe prawdopodobieństwo znalezienia poprawnego rozwiązania na dalszym etapie ewolucji.

Ocena osobników

Po utworzeniu wszystkich osobników (programów) wchodzących w skład danego pokolenia – przez wygenerowanie pokolenia początkowego lub też w drodze operacji genetycznych dla następnych pokoleń – zostają one poddane ocenie środowiska. W tym celu każdy program jest wykonywany[3] i na podstawie rezultatów jego działania za pomocą zdefiniowanej funkcji przystosowania obliczane jest przystosowanie poszczególnych programów. Warto podkreślić, iż etap wykonywania i oceny programów stanowi na ogół najbardziej czasochłonną fazę działania algorytmu GP, natomiast właściwe zdefiniowanie funkcji przystosowania (dopasowania) jest, poza odpowiednim wyborem elementów wchodzących w skład zbiorów $\mathcal{F}$ i $\mathcal{T}$, jedną z największych trudności przy korzystaniu z GP.

3 Standardowa wersja programowania genetycznego, którą zaproponował Koza, używa języka LISP jako języka w którym zapisywane są poszczególne programy podlegające ewolucji, pozwala to bezpośrednio interpretować drzewo jako program podlegający wykonaniu.

Selekcja osobników

Po etapie obliczenia wartości przystosowania dla poszczególnych programów wybierane są najlepsze z nich, które następnie poddawane są działaniu operatorów genetycznych w celu utworzenia następnego pokolenia programów. W programowaniu genetycznym stosowane są dwa podstawowe sposoby wyboru najlepszych osobników: selekcja proporcjonalna (ang. *fitness proportionate selection*) oraz selekcja turniejowa (ang. *tournament selection*).

W przypadku selekcji proporcjonalnej, jak to zostało przedstawione w pracy [33], prawdopodobieństwo, że osobnik s_i zostanie wybrany, jest proporcjonalne do:

$$\frac{f(s_i(t))}{\sum_{j=1}^{M} f(s_j(t))} \tag{3.1}$$

gdzie:

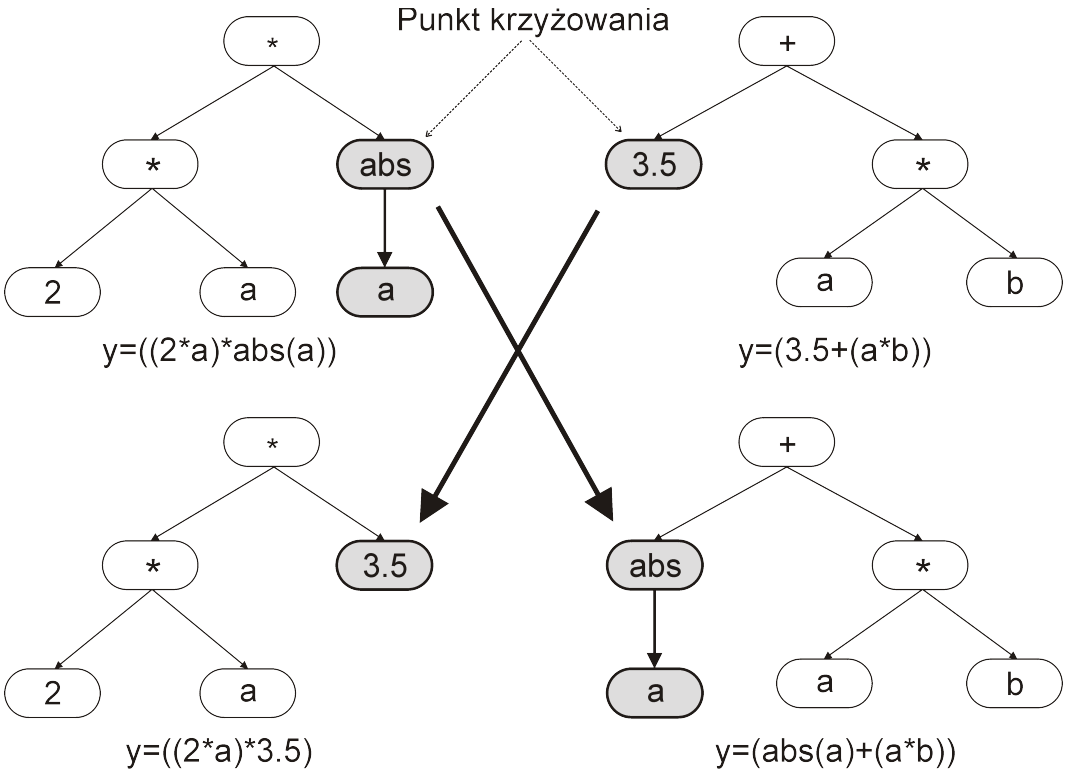
$f(s_i(t))$ – przystosowanie osobnika s_i w pokoleniu t

M – ilość osobników w pokoleniu (rozmiar populacji)

t – numer pokolenia

W selekcji turniejowej za każdym razem wybieranych jest z populacji losowo n osobników, z których osobnik o największej wartości przystosowania przechodzi do następnego pokolenia. Wartość n może być dowolną liczbą całkowitą większą od 1 i mniejszą od ilości osobników w populacji.

W celu utworzenia nowego pokolenia wybrane osobniki poddawane są działaniu jednego z operatorów genetycznych: reprodukcji, krzyżowaniu oraz mutacji. **Reprodukcja,** zwana też klonowaniem, jest operacją polegającą na bezpośrednim skopiowaniu wybranego osobnika do populacji następnego pokolenia. **Krzyżowanie,** inaczej rekombinacja, jest operacją działającą na dwóch osobnikach A i B. Polega ona na losowym wyborze punktu krzyżowania dla każdego z osobników, a następnie na wymianie pomiędzy osobnikami poddrzew, których korzeniami są wybrane punkty krzyżowania. Zasada działania operacji krzyżowania przedstawiona została na rysunku 3.3. **Mutacja** polega na losowym wyborze wierzchołka w drzewie reprezentującym program i zamianie poddrzewa, którego korzeniem jest wybrany punkt, na nowe, losowo wygenerowane, poddrzewo.



Rysunek 3.3. Przykład działania operacji krzyżowania dla dwóch drzew reprezentujących programy

Zakończenie procesu

Po utworzeniu nowego pokolenia programów sprawdzane są warunki zakończenia procesu ewolucji, takie jak osiągnięcie odpowiedniej wartości przystosowania najlepszego programu czy też osiągnięcie zadanej ilości generowanych pokoleń. W przypadku, gdy warunki te są spełnione, proces programowania genetycznego jest przerywany, a jako znalezione rozwiązanie zwracany jest osobnik (program) posiadający największą wartość przystosowania.

W niniejszym rozdziale zaprezentowane zostały podstawy działania programowania genetycznego, stanowiącego szczególny rodzaj algorytmów genetycznych, w którym chromosom o zmiennej długości kodowany jest za pomocą struktury drzewiastej. Przedstawiony powyżej opis dotyczy wersji standardowej programowania genetycznego, czyli takiej, jaką przedstawił Koza w swojej pracy [33]. W rozdziale następnym omówiony zostanie sposób zastosowania przedstawionej tutaj techniki programowania genetycznego dla potrzeb ewolucyjnego generowania (wnioskowania) gramatyk bezkontekstowych na podstawie przykładów języka.

4. Zastosowanie podejścia ewolucyjnego do generowania gramatyk bezkontekstowych

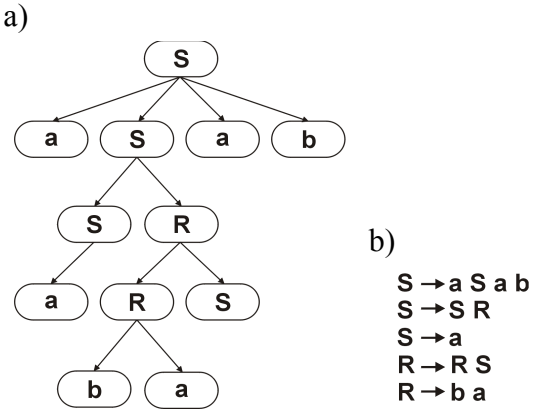
Rozdział ten stanowi opis zastosowania techniki programowania genetycznego, przedstawionego w rozdziale poprzednim, do celów ewolucyjnego generowania gramatyk bezkontekstowych na bazie przykładów języka.

4.1. Reprezentacja gramatyki bezkontekstowej za pomocą drzewa

Jak już zostało to zasygnalizowane w poprzednim rozdziale, w przypadku zastosowania podejścia ewolucyjnego, opartego o technikę programowania genetycznego, istotny jest odpowiedni sposób zakodowania programu w postaci chromosomu. W przypadku generowania gramatyki rolę programu, który podlega ewolucji, pełni zapis produkcji gramatyki. Spośród różnych możliwych postaci reprezentacji produkcji gramatyki, takich jak Postać Normalna Chomsky'ego (**CNF**), Postać Normalna Greibacha (**GNF**) czy Postać Backusa–Naura (**BNF**), wybrana została ta ostatnia. Decyzja taka podyktowana została tym, iż postać BNF jest najbardziej ogólna z przedstawionych, co z reguły umożliwia zapis gramatyki za pomocą mniejszej ilości produkcji i symboli nieterminalnych niż w przypadku zastosowania zapisu o postaci CNF lub GNF. Zaś mniejsza ilość produkcji i symboli nieterminalnych w zapisie gramatyki wpływa bezpośrednio na zmniejszenie obszaru poszukiwań [74].

Aby zakodować produkcje gramatyki bezkontekstowej w chromosomie, zastosowana została reprezentacja w postaci drzewa. W drzewie tym korzeń reprezentuje symbol startowy gramatyki (STS), zaś wierzchołki – symbole terminalne i nieterminalne, przy czym symbole terminalne reprezentowane są jako liście drzewa.

Na rysunku 4.1 przedstawiono reprezentację pewnej gramatyki bezkontekstowej za pomocą drzewa oraz odpowiadającą jej reprezentację w postaci BNF.



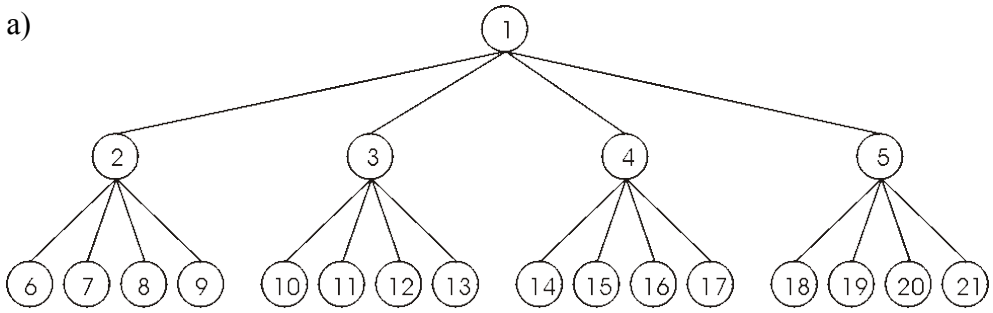
Rysunek 4.1. Reprezentacja produkcji przykładowej gramatyki bezkontekstowej a) za pomocą drzewa b) w postaci BNF

4.2. Wizualizacja struktury drzewa

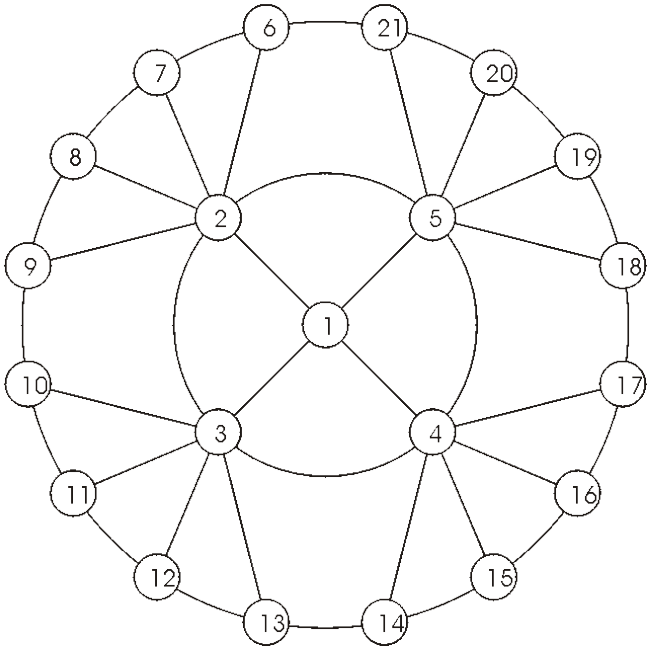
Dla celów wizualizacji struktury drzew reprezentujących produkcje gramatyk zaadoptowana została metoda zaproponowana w pracy [11], polegająca na prezentacji drzewa na siatce kołowej (ang. *circular grid*). Metoda ta pozwala na przedstawienie struktury drzew charakteryzujących się znacznym stopniem złożoności na stosunkowo niewielkiej powierzchni rysunku.

Na rysunku 4.2 jako przykład pokazane zostało pełne 4–drzewo o wysokości 3 oraz jego reprezentacja na siatce kołowej.

a)



b)



Rysunek 4.2. Pełne 4-drzewo a) standardowy sposób przedstawienia, b) przedstawienie na siatce kołowej

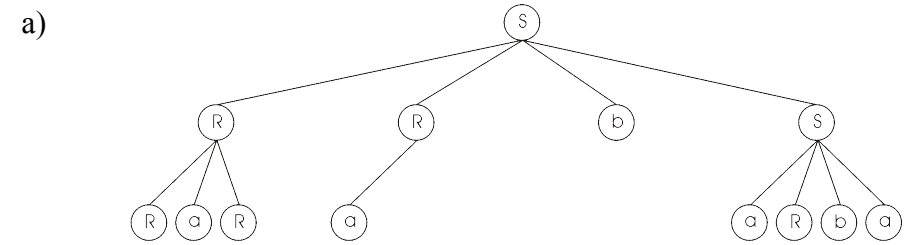
Metoda prezentacji drzewa na siatce kołowej stosowana jest w celu przedstawienia samej jego struktury, dla której nie są istotne etykiety poszczególnych wierzchołków;

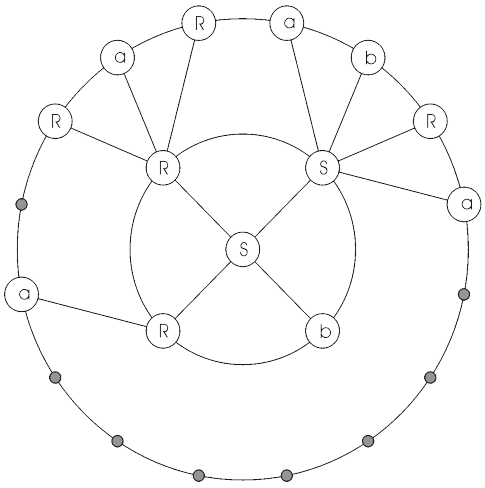
b) c)

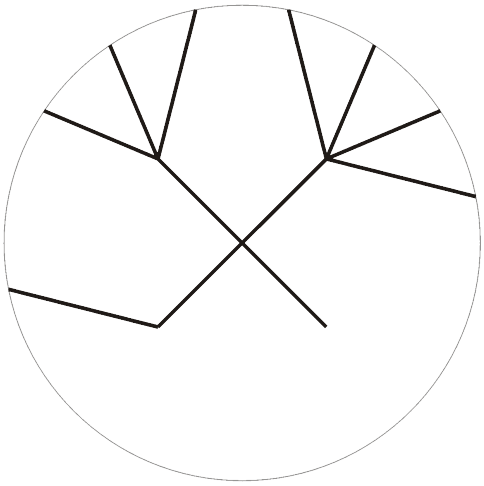
dlatego też na ogół wierzchołki są pomijane, a przedstawia się jedynie krawędzie drzewa.

Na rysunku 4.3 znajduje się przykład drzewa reprezentującego produkcje pewnej gramatyki a) oraz jego przedstawienie na siatce kołowej z zaznaczeniem wierzchołków i krawędzi b) a także samych krawędzi c).

a)







Rysunek 4.3. Drzewo reprezentujące produkcje pewnej gramatyki a) standardowy sposób przedstawienia drzewa, b) przedstawienie na siatce kołowej z zaznaczeniem wierzchołków krawędzi drzewa, c) przedstawienie na siatce kołowej samych krawędzi drzewa

W dalszej części niniejszej pracy dla celów wizualizacji struktury drzew reprezentujących produkcje gramatyki używana będzie prezentacja krawędzi drzewa na siatce kołowej.

4.3. Funkcja dopasowania

Tak jak zostało to omówione w rozdziale 3, funkcja dopasowania (przystosowania) jest miarą zdolności rozwiązania zadanego problemu przez poszczególne osobniki biorące udział w procesie ewolucji. Przez zdolność rozwiązania

problemu w przypadku generowania gramatyk rozumiana jest poprawność klasyfikacji syntaktycznej zadanych przykładowych wzorców przez wygenerowaną gramatykę.

Dla celów przedstawionych w dalszej części pracy badań zaproponowano dwa rodzaje funkcji dopasowania: funkcję podstawową f_{NORM} oraz funkcję rozszerzoną f_{EXT}. W obydwu tych przypadkach wartość funkcji dopasowania zależy od liczby zaakceptowanych przez wygenerowaną gramatykę przykładów pozytywnych oraz liczby odrzuconych przez nią przykładów negatywnych. Różnica pomiędzy tymi funkcjami polega na tym, iż w przypadku funkcji podstawowej f_{NORM} dla przykładów pozytywnych rozpatrywane jest tylko to, czy przykład został zaakceptowany przez wygenerowaną gramatykę, czy też nie, natomiast dla funkcji rozszerzonej $f_{EXT,}$, w przypadku, gdy przykład pozytywny nie jest poprawnie rozpoznany, bierze się pod uwagę także to, ile symboli wejściowych z tego przykładu zostało poprawnie rozpoznanych.

Wartość podstawowej funkcji dopasowania f_{NORM} dla gramatyki g_i wyraża się następującym wzorem:

$$f_{NORM}(g_i)=\begin{cases} f(g_i) \;\; dla\, f(g_i)<0{,}5 \\ 1-\dfrac{q}{2\cdot Q}\, dla\, f(g_i)=0{,}5 \end{cases} \tag{4.1}$$

gdzie:

$$f(g_i)=\frac{\sum_{k=1}^{N} f_k(g_i)}{2\cdot N} \tag{4.2}$$

$f(g_i)$ – funkcja dopasowania dla przykładów pozytywnych

$f_k(g_i)$ – funkcja dopasowania k-tego przykładu pozytywnego – osiąga wartość 1, jeśli k-ty przykład pozytywny został zaakceptowany przez gramatykę, i wartość 0 jeśli przykład został odrzucony

N – całkowita ilość przykładów pozytywnych

q – ilość zaakceptowanych (niepoprawnie sklasyfikowanych) przykładów negatywnych

Q – całkowita ilość przykładów negatywnych

Natomiast wartość rozszerzonej funkcji dopasowania f_{EXT} dla gramatyki g_i wyraża się następującym wzorem:

$$f_{EXT}(g_i)=\begin{cases} f(g_i) \;\; dla\, f(g_i)<0{,}5 \\ 1-\dfrac{q}{2\cdot Q}\, dla\, f(g_i)=0{,}5 \end{cases} \tag{4.3}$$

gdzie:

$$f(g_i)=\frac{\sum_{k=1}^{N} f_k(g_i)}{N} \tag{4.4}$$

$$f_k(g_i)=\frac{l_k}{2\cdot L_k} \tag{4.5}$$

$f(g_i)$ – funkcja dopasowania dla przykładów pozytywnych

$f_k(g_i)$ – funkcja dopasowania k-tego przykładu pozytywnego

N – całkowita ilość przykładów pozytywnych

q – ilość zaakceptowanych (niepoprawnie sklasyfikowanych) przykładów negatywnych

Q – całkowita ilość przykładów negatywnych

l_k – ilość poprawnie rozpoznanych symboli terminalnych w k-tym przykładzie pozytywnym

L_k – całkowita ilość symboli terminalnych w k-tym przykładzie pozytywnym

Obydwie funkcje dopasowania zostały skonstruowane w taki sposób, że uwzględniają poprawność rozpoznania dla przykładów negatywnych tylko w sytuacji, gdy wszystkie przykłady pozytywne zostały poprawnie zaklasyfikowane (tzn. wartość funkcji $f(g_i)$ jest równa 0.5). Rozwiązanie takie zapobiega przejęciu decydującej roli w sterowaniu procesem ewolucji przez przykłady negatywne szukanego języka. Jest to konieczne, ponieważ statystycznie znacznie łatwiej jest wygenerować gramatykę, która odrzuci wszystkie przykłady negatywne, niż taką, która zaakceptuje wszystkie przykłady pozytywne szukanego języka.

W badaniach przedstawionych w dalszej części pracy, funkcja rozszerzona będzie wykorzystywana w przypadku zastosowania w procesie ewolucji gramatyk parsera Bison, natomiast w przypadku zastosowania parserów Elkhound i CYKP używana będzie funkcja podstawowa.

4.4. Entropia

Dla celów określenia stopnia zróżnicowania osobników wchodzących w skład populacji wykorzystana została entropia oparta na wartości funkcji przystosowania.

Zastosowany w niniejszej rozprawie sposób określenia entropii stanowiącej miarę nieuporządkowania układu dla potrzeb programowania genetycznego wprowadzony został przez J.P. Rosca [65]. Idea tego rozwiązania jest następująca: jeśli unikalne wartości funkcji dopasowania osobników wchodzących w skład populacji ponumerujemy od 1 do k, to p_i (gdzie $1 \leq i \leq k$) określamy jako stosunek ilości osobników posiadających i–tą wartość funkcji przystosowania do ilości wszystkich osobników wchodzących w skład populacji. Na przykład, jeśli populacja liczy 100 osobników i dla 90 spośród nich wartość funkcji przystosowania wynosi 0.5, natomiast dla 10 pozostałych wynosi 0.8 – czyli istnieją dwie unikalne wartości funkcji przystosowania (k = 2) – to wartość $p_1 = 90/100$, a wartość $p_2 = 10/100$.

Entropia S zdefiniowana jest następująco:

$$S = -\sum_{i=1}^{k} p_i \cdot \log_2(p_i) \tag{4.6}$$

Wzrost entropii, oznaczający przejście układu do stanu mniej uporządkowanego, może być spowodowany wzrostem ilości różnych wartości funkcji przystosowania występujących w populacji (wzrost k) lub też zmianą rozkładu ilości osobników posiadających określoną wartość funkcji przystosowania. W przypadku takiej zmiany entropia jest tym większa, im rozkład ten jest bardziej równomierny (czyli mniej uporządkowany). Na przykład, jeśli dla całej populacji liczącej 100 osobników istnieją tylko dwie unikalne wartości funkcji przystosowania, to entropia będzie najmniejsza jeżeli pierwszą z tych wartości będzie posiadał tylko jeden osobnik, a pozostałe 99 osobników będzie posiadać drugą z wartości. Entropia będzie największa przy rozkładzie równomiernym, czyli wówczas, gdy 50 osobników będzie posiadać pierwszą z unikalnych wartości funkcji przystosowania, a pozostałe 50 – drugą. Najmniejszą możliwą wartością, jaką może osiągać entropia oparta na funkcji przystosowania, jest wartość 0, i występuje ona w przypadku, gdy wszystkie osobniki mają taką samą wartość funkcji przystosowania.

4.5. Operacje genetyczne

4.5.1. Krzyżowanie

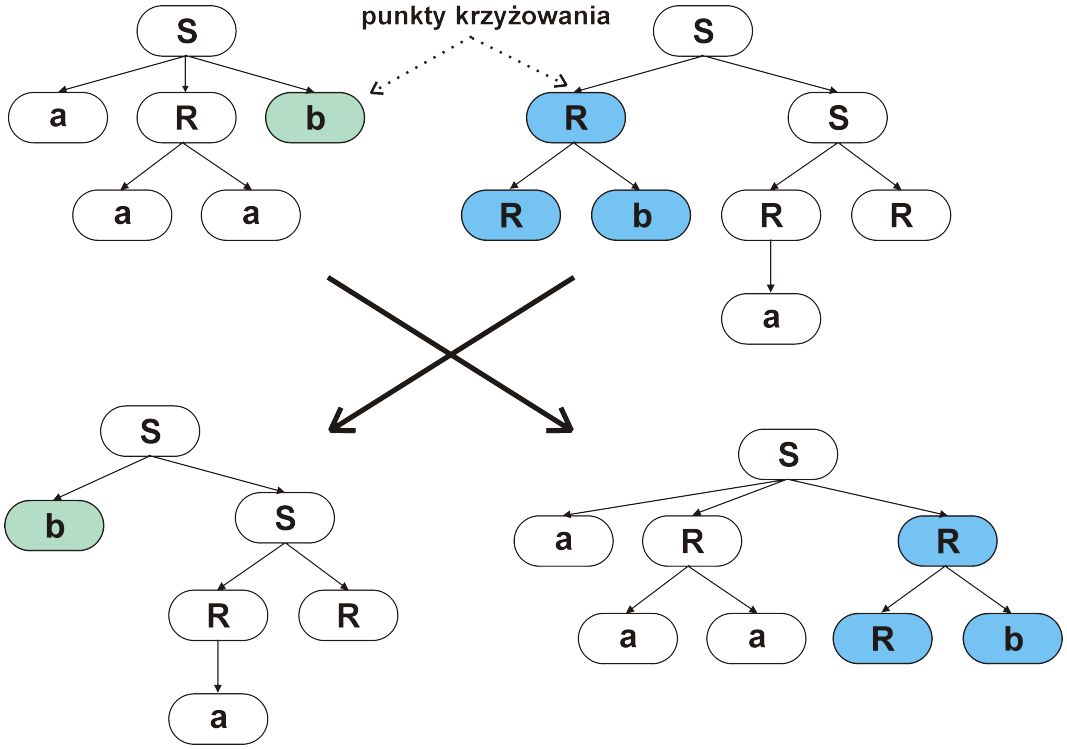
Podstawową operację genetyczną stosowaną w procesie ewolucji jest operacja krzyżowania dla wybranej pary osobników rodzicielskich (w naszym przypadku gramatyk). W badaniach przedstawionych w dalszej części pracy porównana została efektywność trzech rodzajów operacji krzyżowania:

- standardowego
- zachowującego rozmiar (ang. *size fair crossover*)

– homologicznego (ang. *homologous crossover*).

W operacji krzyżowania standardowego dla każdego z osobników rodzicielskich punkt krzyżowania w drzewie reprezentującym gramatykę wybierany jest w sposób losowy. Następnie pomiędzy osobnikami rodzicielskimi (gramatykami) wymieniane są poddrzewa, których korzeniem jest wybrany losowo punkt krzyżowania, w wyniku czego z pary osobników rodzicielskich powstaje para osobników (gramatyk) potomnych.

Przykład działania operacji krzyżowania standardowego przedstawiony został na rysunku 4.4.



Rysunek 4.4. Przykład działania operacji krzyżowania standardowego dla gramatyk

Operacja krzyżowania zachowującego rozmiar, podobnie jak to zostało omówione w pracy [35], odbywa się tak, jak krzyżowanie standardowe, z tą różnicą, że punkt krzyżowania dla drugiego z rodziców nie jest wybierany w sposób w pełni losowy, lecz jego wybór jest uzależniony od ilości wierzchołków w w poddrzewie wybranym dla pierwszego z rodziców, którego korzeniem jest pierwszy punkt krzyżowania. Wybór punktu krzyżowania dla drugiego z rodziców odbywa się następująco: najpierw obliczany jest rozmiar każdego poddrzewa dla drugiego z rodziców, a następnie wszystkie poddrzewa, których rozmiar jest większy niż $2 * w - 1$, są pomijane. Indeksy poddrzew, które pozostały (czyli tych, których ilość wierzchołków

jest większa od 1 i mniejsza od 2 * *w* – 1), są przypisywane do jednego z trzech zbiorów P_1, P_2 lub P_3. Przy czym do zbioru P_1 przypisywane są indeksy poddrzew o ilości wierzchołków mniejszej niż *w*, do P_2 – indeksy poddrzew o ilości wierzchołków równej *w*, natomiast do zbioru P_3 – o ilości wierzchołków większej niż *w*.

Wprowadzamy następujące oznaczenia:

n_1 – ilość elementów w zbiorze P_1 (czyli ilość poddrzew dla drugiego rodzica o mniejszej ilości wierzchołków niż w wybranym poddrzewie dla pierwszego rodzica)

n_2 – ilość elementów w zbiorze P_2

n_3 – ilość elementów w zbiorze P_3

W przypadku, gdy dla drugiego rodzica nie ma poddrzew o większej lub mniejszej ilości wierzchołków niż ilość wierzchołków w poddrzewie wybranym dla pierwszego (czyli, gdy n_1 lub n_3 jest równe zero), wybór dokonywany jest tylko spośród poddrzew drugiego rodzica o takiej samej ilości wierzchołków jak w poddrzewie wybranym dla pierwszego (o ile takie istnieją, czyli $n_2 > 0$). W takim wypadku prawdopodobieństwo wybrania danego poddrzewa wyraża się wzorem:

$$p = \frac{1}{n_2} \tag{4.7}$$

W przypadku gdy n_1 i n_3 są większe od zera, wybór poddrzewa odbywa się ze wszystkich poddrzew o indeksie k, gdzie:

$$k \in P1 \cup P2 \cup P3$$

Przy czym prawdopodobieństwo wybrania poddrzewa o ilości wierzchołków w_k jest proporcjonalne do wartości funkcji h zdefiniowanej następująco:

$$h(w_k) = 1 - \frac{|w - w_k|}{w} \tag{4.8}$$

Funkcja ta osiąga wartość maksymalną 1 dla w_k równego w oraz wartość minimalną 1/w dla w_k równego 1 oraz w_k równego 2 * w – 1. Takie podejście jest rozszerzeniem rozwiązania zaproponowanego w pracy [35], w którym

prawdopodobieństwo wybrania poddrzew o ilości wierzchołków różnej od *w* jest niezależne od ilości wierzchołków.

Prawdopodobieństwo wybrania k-tego poddrzewa o ilości wierzchołków mniejszej niż *w* (czyli dla k należącego do zbioru P1) wynosi:

$$p_{-}(k)=\frac{h(w_k)}{\sum_{k\in P1} h(w_k)+\sum_{k\in P2} h(w_k)+c\cdot\sum_{k\in P3} h(w_k)}=\frac{h(w_k)}{\sum_{k\in P1} h(w_k)+n_2+c\cdot\sum_{k\in P3} h(w_k)} \quad (4.9)$$

gdzie:

w_k – ilość wierzchołków k-tego poddrzewa

c – stała dobrana tak, aby wartość oczekiwana $E\{(w - w_k)\}$ była równa zero

Prawdopodobieństwo wybrania k-tego poddrzewa o ilości wierzchołków większej niż *w* i mniejszej niż 2**w* (czyli k należącego do zbioru P3) wynosi:

$$p_{+}(k)=\frac{c\cdot h(w_k)}{\sum_{k\in P1} h(w_k)+\sum_{k\in P2} h(w_k)+c\cdot\sum_{k\in P3} h(w_k)}=\frac{c\cdot h(w_k)}{\sum_{k\in P1} h(w_k)+n_2+c\cdot\sum_{k\in P3} h(w_k)} \quad (4.10)$$

Natomiast prawdopodobieństwo wybrania k-tego poddrzewa o ilości wierzchołków równej *w* (czyli k należącego do zbioru P2) wynosi:

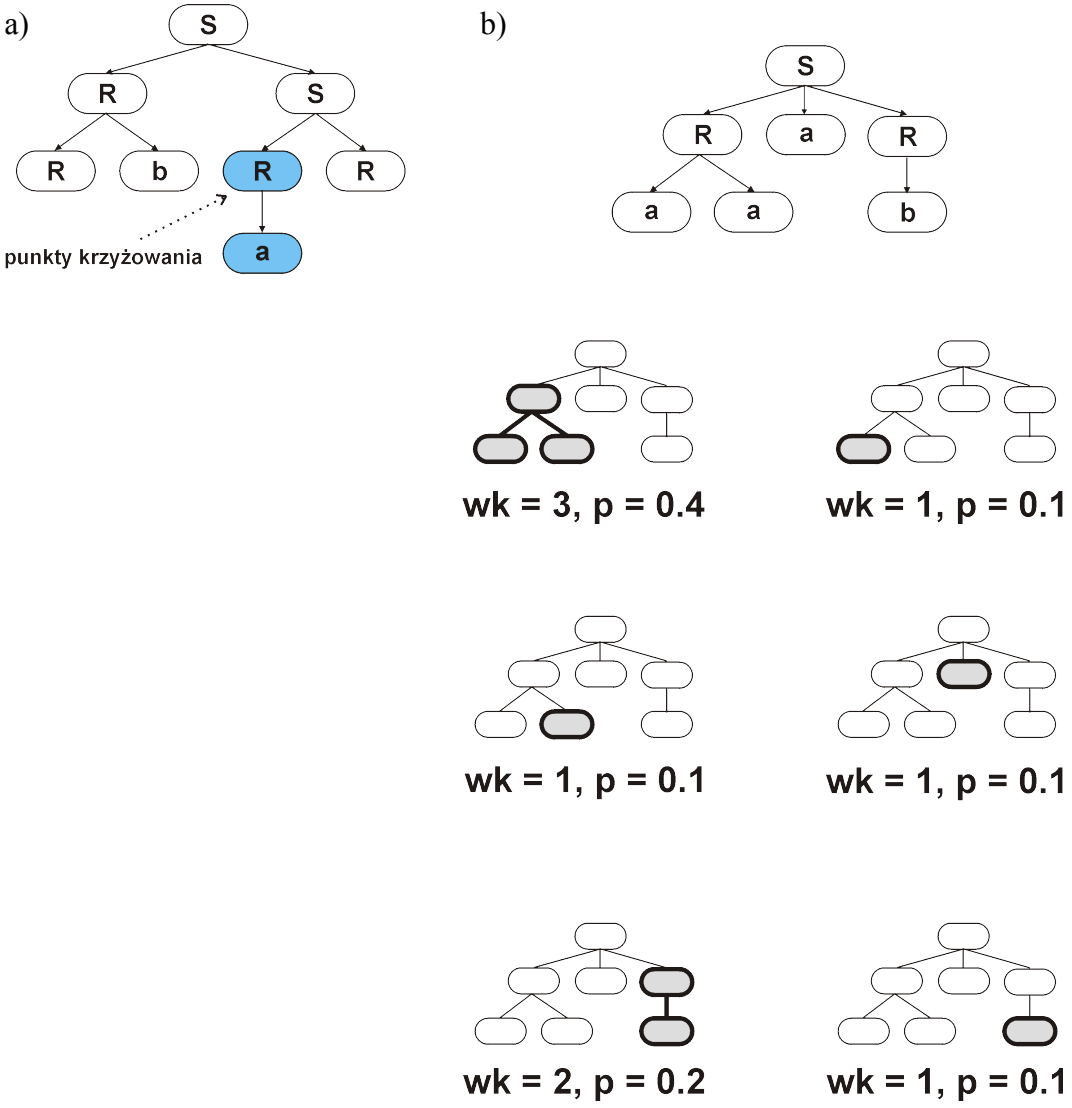
$$p_0(k)=\frac{h(w_k)}{\sum_{k\in P1} h(w_k)+\sum_{k\in P2} h(w_k)+c\cdot\sum_{k\in P3} h(w_k)}=\frac{1}{\sum_{k\in P1} h(w_k)+n_2+c\cdot\sum_{k\in P3} h(w_k)} \quad (4.11)$$

Wartość stałej c, która jest wyznaczona tak, aby wartość oczekiwana $E\{(w - w_k)\}$ była równa zero, wynosi:

$$c=\frac{\sum_{k\in P1} (w-w_k)\cdot h(w_k)}{\sum_{k\in P3} (w_k-w)\cdot h(w_k)} \quad (4.12)$$

Ponieważ dla operacji krzyżowania zachowującej rozmiar wartość oczekiwana $E\{(w - w_k)\}$ jest równa zero, średnio statystycznie ilość wierzchołków w wymienianych poddrzewach dla obydwóch rodziców jest taka sama.

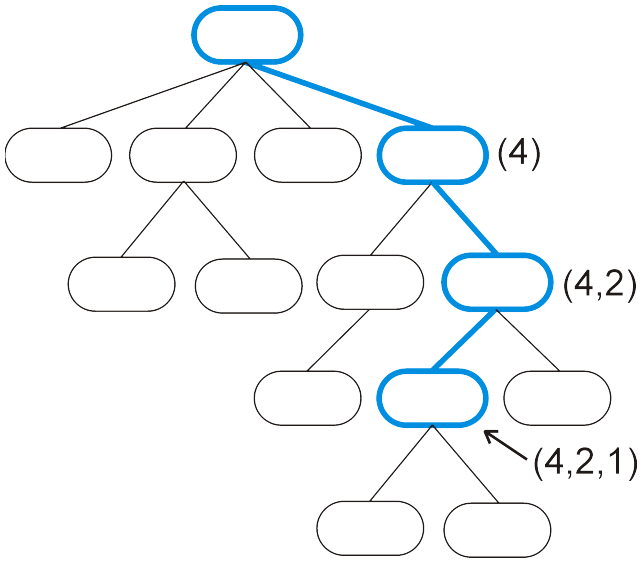
Na rysunku 4.5 przedstawiony został przykład działania operacji krzyżowania zachowującego rozmiar.



Rysunek 4.5. Operacja krzyżowania zachowującego rozmiar a) pierwszy rodzic z zaznaczonym losowo wybranym punktem krzyżowania, b) drugi rodzic oraz zaznaczone dla niego poddrzewa, których korzeniami są dozwolone punktach krzyżowania z podaną ilością wierzchołków wk oraz prawdopodobieństwem wyboru p

Operacja krzyżowania homologicznego działa podobnie do operacji krzyżowania zachowującego rozmiar. Różnica polega na tym, że wybór poddrzewa dla drugiego rodzica nie odbywa się w sposób losowy z prawdopodobieństwem zależnym od funkcji $h(w_k)$, lecz dokonywany jest spośród poddrzew o takiej samej ilości wierzchołków, jak w wybranym poddrzewie dla pierwszego rodzica (czyli takich, dla których funkcja $h(w_k)$ ma wartość 1) w taki sposób, aby różnica w pozycjach korzeni obydwóch poddrzew była jak najmniejsza. Pozycja wierzchołka, podobnie jak w pracy [10], określana jest jako n-tka współrzędnych $T = (k_1, k_2,..., k_n)$, gdzie n jest głębokością wierzchołka w drzewie, natomiast k_i określa którą, krawędź, licząc od lewej strony do prawej dla danego wierzchołka, należy wybrać na poziomie i, aby dojść do danego wierzchołka. Na

przykład, współrzędne (4,2,1) oznaczają wierzchołek, który zostanie osiągnięty po wybraniu na pierwszym poziomie (dla korzenia) czwartej krawędzi, na drugim poziomie – drugiej i na trzecim – pierwszej krawędzi, co zostało pokazane na rysunku 4.6.



Rysunek 4.6. Współrzędne przykładowych wierzchołków w drzewie

Jeśli pozycja T korzenia poddrzewa dla pierwszego rodzica jest równa $(t_1, t_2,..., t_n)$, natomiast pozycja S korzenia poddrzewa dla drugiego rodzica jest równa $(s_1, s_2,..., s_m)$, to różnica pozycji jest tym większa, im dla mniejszego p (gdzie p jest większe lub równe 1 i mniejsze lub równe max(n,m)[4]) pozycje t_p i s_p się różnią. Tak więc różnica pozycji poddrzew jest tym mniejsza, im różnica w ścieżce prowadzącej od korzeni drzew do korzeni wybranych poddrzewa występuje dalej od korzenia drzewa. Na przykład, jeśli pozycja korzenia wybranego poddrzewa dla pierwszego rodzica wynosi T_1 = (4,2,1,2), a dla drugiego rodzica wybrane są dwa poddrzewa o współrzędnych odpowiednio: S_1 = (4,3,1,2) oraz S_2 = (4,2,5,2,2), to różnica współrzędnych T_1 i S_1 jest większa niż T_1 i S_2, ponieważ w pierwszym przypadku różnica występuje na drugiej pozycji (czyli dla p = 2, ponieważ w tym przypadku $t_1 = s_1 = 4$, natomiast $t_2 = 2$, a $s_2 = 3$), natomiast w drugim przypadku na trzeciej pozycji (czyli dla p = 3, ponieważ w tym przypadku $t_1 = s_1 = 4$ i $t_2 = s_2 = 2$, natomiast $t_3 = 1$, a $s_3 = 5$). W takim przypadku w operacji krzyżowania homologicznego zostałoby wybrane poddrzewo, dla którego różnica współrzędnych korzenia jest mniejsza, czyli to, którego współrzędna korzenia wynosi S_2. W przypadku poddrzew, których różnica współrzędnych występuje na tej samej pozycji, wybierane jest to, dla którego różnica ta jest mniejsza.

4 Dla p większego od n lub m przyjęto, iż t_p (lub odpowiednio s_p) jest równe zero.

Jak to zostało wcześniej przedstawione, podstawowa operacja krzyżowania przenosi fragment kodu z jednego opisu produkcji gramatyki do drugiego. Ma to na celu utworzenie z pary osobników rodzicielskich pary osobników potomnych, które mogą mieć lepszą zdolność do rozwiązania problemu niż osobniki rodzicielskie (miarą tej zdolności jest wartość funkcji dopasowania). Jednak ponieważ w przypadku standardowej operacji krzyżowania miejsce do którego przenoszony jest fragment kodu, jest przypadkowe, często zdarza się, że taka operacja ma działanie destrukcyjne polegające na tym, iż fragment kodu (produkcji gramatyk), który poprawnie działał w pierwotnej pozycji po przeniesieniu do nowej pozycji przestaje być poprawny [35]. Wynika to z faktu, iż działanie programu (opisu gramatyki) w dużej mierze zależy od kontekstu, w jakim się znajduje. Zaletą operatora krzyżowania homologicznego w stosunku do standardowego operatora krzyżowania jest to, iż zachowuje on kontekst wymienianych fragmentów kodu (poddrzew). Tym samym zmniejsza destrukcyjne skutki działania operacji krzyżowania.

Skuteczność działania poszczególnych operatorów krzyżowania oraz ich wpływ na wzrost rozmiaru drzew reprezentujących gramatyki przedstawione zostaną w rozdziale 6. Porównanie tych operatorów zawarte zostało w pracy [56].

4.5.2. Mutacja

Celem zastosowania operacji mutacji w procesie ewolucji gramatyk jest wprowadzenie dodatkowego mechanizmu pozwalającego na modyfikację istniejących i dodawanie nowych produkcji do istniejącej populacji gramatyk.

Operacja mutacji polega na losowym wyborze wierzchołka reprezentującego symbol nieterminalny w drzewie opisującym gramatykę, a następnie na wykonaniu jednej ze zdefiniowanych operacji modyfikacji tego wierzchołka:

a) zmiana kolejności dwóch synów (dzieci) – w ramach jednej produkcji następuje zamiana ze sobą dwóch losowo wybranych symboli (*transwersja*);

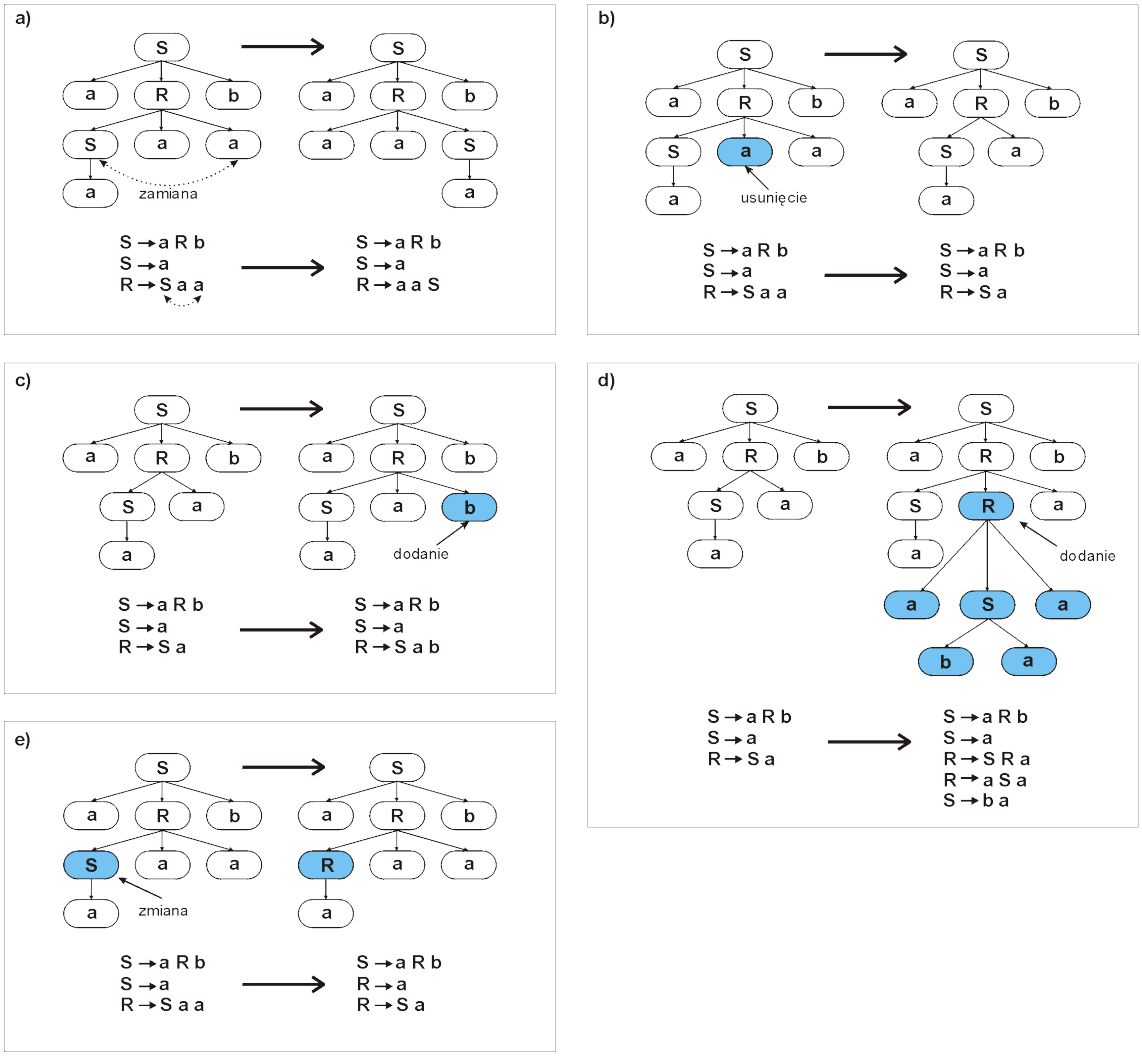
b) usunięcie jednego z synów – usuwany jest losowo wybrany symbol po prawej strony produkcji (*delecja*);

c) dodanie losowego symbolu terminalnego do prawej strony produkcji (*addycja terminalna*);

d) dodanie losowego symbolu nieterminalnego do prawej strony produkcji – wstawiany jest wierzchołek reprezentujący symbol nieterminalny, a następnie generowane jest losowe poddrzewo (reprezentujące nowe produkcje), którego korzeniem jest wstawiony wierzchołek (*addycja nieterminalna*);

e) zmiana wybranego symbolu nieterminalnego – zmieniany jest wybrany symbol nieterminalny znajdujący się po lewej stronie produkcji na inny wybrany losowo (*tranzycja niterminala*).

Na rysunku 4.7 przedstawione zostały przykłady działania poszczególnych rodzajów operacji mutacji.



Rysunek 4.7. Przykład działania różnych rodzajów operacji mutacji a) zamiana kolejności symboli w prawej stronie produkcji, b) usunięcie jednego z symboli po prawej stronie produkcji, c) dodanie symbolu terminalnego d) dodanie symbolu nieterminalnego wraz z poddrzewem reprezentującym nowe produkcje, e) zmiana wybranego symbolu nieterminalnego z lewej strony produkcji na inny

Porównanie efektywności działania poszczególnych operatorów mutacji w procesie ewolucji gramatyk przedstawione zostanie w części badawczej niniejszej pracy.

4.6. Przebieg ewolucji

W pracy przedstawione i porównane zostaną dwa modele ewolucji gramatyk: model klasyczny, odpowiadający standardowemu schematowi działania programów

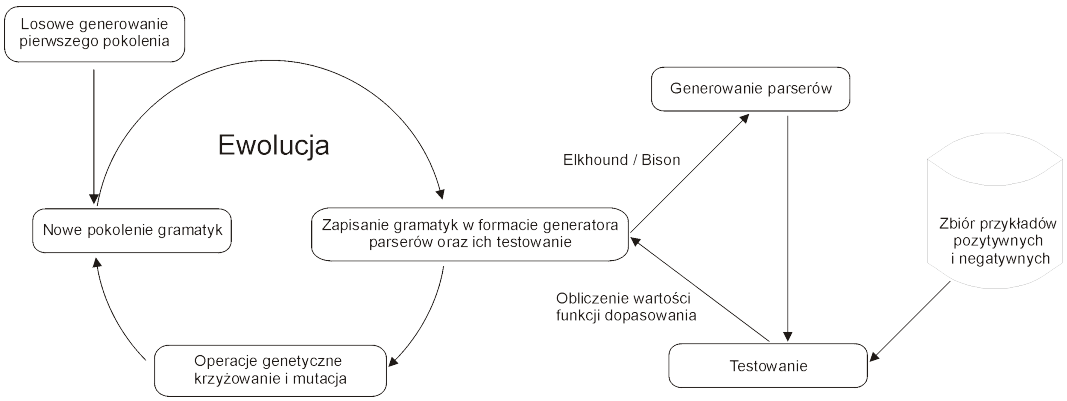
genetycznych przedstawionemu w rozdziale 3.2, w którym występuje wyraźny podział na poszczególne pokolenia gramatyk, oraz opracowany przez autora model ewolucji 'ciągłej', w którym gramatyki testowane są za każdym razem za pomocą tylko jednego z przykładów języka i w którym nie występuje podział na poszczególne pokolenia.

4.6.1. Klasyczny model ewolucji

Schemat klasycznej ewolucji gramatyk podobny jest do przedstawionego w pracach [53] oraz [54]:

1) W pierwszym etapie ewolucji generowane jest początkowe pokolenie gramatyk
2) Wygenerowane gramatyki tłumaczone są na format akceptowany przez wybrany generator analizatorów składniowych (parserów): Bison, Elkhound lub CYKP. Następnie, w przypadku użycia generatorów parserów Bison lub Elkhound, za ich pomocą tworzony jest kod źródłowy analizatora syntaktycznego w języku C (w przypadku generatora Bison) lub C++ (w przypadku generatora Elkhound) i tak otrzymany parser w postaci źródłowej jest kompilowany za pomocą kompilatora języka C++ lub C do kodu wykonywalnego. Natomiast w przypadku analizatora syntaktycznego CYKP plik zawierający opis gramatyki wykorzystywany jest jako plik wejściowy dla analizatora syntaktycznego.
3) Wygenerowane analizatory syntaktyczne są sprawdzane na testowym zbiorze przykładów pozytywnych i negatywnych szukanego języka. Na tej podstawie dla każdej wygenerowanej gramatyki g_i obliczana jest wartość funkcji dopasowania $f(g_i)$.
4) Sprawdzane są warunki zakończenia procesu ewolucji, takie jak, na przykład, osiągnięcie maksymalnej założonej ilości pokoleń lub wartości funkcji dopasowania dla najlepszej znalezionej gramatyki. W przypadku spełnienia tych warunków proces ewolucji zostaje zakończony.
5) Na podstawie wartości funkcji przystosowania wybierane są gramatyki stanowiące bazę dla nowego pokolenia. Następnie, z zadanym prawdopodobieństwem, poddawane są one operacji krzyżowania oraz mutacji. Tak powstałe gramatyki weryfikowane są pod kątem spełnienia założonych ograniczeń, takich jak maksymalna wysokość drzewa opisującego gramatykę, maksymalna ilość symboli w prawej stronie produkcji, maksymalna ilość symboli nieterminalnych w prawej stronie produkcji itp. Jeśli weryfikacja danej gramatyki zakończy się powodzeniem, zostaje ona dołączona do puli tworzącej nowe pokolenie.
6) Proces ewolucji powraca do etapu 2 i kontynuowany jest do momentu osiągnięcia któregoś z warunków zakończenia opisanych w etapie 4.

Rysunek 4.8 przedstawia schematycznie podstawowy proces ewolucji gramatyk.



Rysunek 4.8. Proces ewolucji gramatyk

Opisany powyżej proces ewolucji gramatyk odpowiada klasycznemu schematowi działania algorytmów genetycznych, czy też programowania ewolucyjnego [4], [33], w którym istnieje ścisły podział poszczególnych pokoleń podczas ewolucji. Jeśli wybór osobników, które mają wejść w skład następnego pokolenia odbywa się w sposób losowy proporcjonalny do wartości funkcji dopasowania (na przykład w reprodukcji ruletkowej czy też turniejowej), to może się zdarzyć, że osobnik z największą wartością funkcji dopasowania nie zostanie promowany do następnego pokolenia. Aby temu zapobiec, stosuje się pewne modyfikacje procesu selekcji, na przykład polegające na tym, że określona ilość (na przykład 10%) osobników z największymi wartościami funkcji dopasowania zawsze przechodzi do następnego pokolenia. Jednak wadą takiego rozwiązania jest możliwość zdominowania nowego pokolenia przez osobniki z największymi wartościami funkcji dopasowania. Dodatkową słabością przedstawionej tutaj metody ewolucyjnego znajdowania gramatyk jest jej brak odporności na błędy występujące w przykładach szukanego języka. Ponieważ wartość funkcji dopasowania jest zawsze wyliczana na podstawie wszystkich przykładów pozytywnych i, w przypadku poprawnego ich rozpoznawania przez gramatykę, wszystkich przykładów negatywnych, więc jeśli nawet tylko w jednym z przykładów pozytywnych występuje błąd (taki, iż należy on do języka generowanego przez szukaną gramatykę), to dla gramatyki takiej funkcja dopasowania nie osiągnie maksymalnej możliwej wartości, czyli w przypadku rozważanych tutaj funkcji wartości 1. Jeśli natomiast zostanie znaleziona gramatyka, dla której funkcja dopasowania osiągnie wartość maksymalną, to gramatyka ta będzie niepoprawnie rozpoznawać błędny przykład.

4.6.2. Model ewolucji ciągłej

Rozwiązaniem przedstawionych powyżej problemów jest opracowany przez autora schemat ewolucji 'ciągłej', w którym nie występuje podział na poszczególne pokolenia, zaś test gramatyki odbywa się za każdym razem tylko dla jednego przykładu języka (pozytywnego lub negatywnego) – w tym przypadku nie jest obliczana funkcja dopasowania bazująca na wszystkich przykładach.

Przebieg ewolucji 'ciągłej' składa się z następujących etapów:

1) Na początku, podobnie jak to miało miejsce w przypadku klasycznego sposobu ewolucji, generowana jest początkowa populacja gramatyk.
2) Dla każdej wygenerowanej gramatyki przypisywana jest pewna początkowa wartość *poprawności rozpoznania przykładów*. W przypadku prezentowanych w dalszej części pracy badań wartość początkowa *poprawności rozpoznania przykładów* wynosiła 2, chyba że została ona jawnie określona inaczej.
3) Wszystkie gramatyki wchodzące w skład początkowej populacji tłumaczone są na format akceptowany przez generator parserów. Następnie, przy użyciu tego generatora, gramatyki tłumaczone są na postać kodu źródłowego w języku C lub C++. Na koniec kod źródłowy kompilowany jest do kodu programu wykonywalnego.
4) W sposób losowy wybierana jest jedna z gramatyk, a następnie jeden z przykładów języka (pozytywny lub negatywny). Wybrana gramatyka testowana jest dla tego przykładu i w przypadku poprawnego rozpoznania wartość *poprawności rozpoznania przykładów* zwiększana jest o pewną wartość v_+, natomiast w przypadku błędnego rozpoznania jest zmniejszana o v_- [5]. Jeśli wartość *poprawności rozpoznania przykładów* spadnie poniżej zera, dana gramatyka jest usuwana z populacji, a na jej miejsce wstawiana jest nowa powstała przez operacje krzyżowania i mutacji dwóch losowo wybranych gramatyk z pozostałej puli. Następnie gramatyka tłumaczona jest na format akceptowany przez generator analizatorów syntaktycznych i generowany jest kod źródłowy, który kompilowany jest do kodu programu wykonywalnego. Dla tej nowo powstałej gramatyki przypisywana jest początkowa wartość *poprawności rozpoznania przykładów*.
5) Opcjonalnie, co określoną ilość razy zakończenia etapu 4, obliczana jest sumaryczna wartość *poprawności rozpoznania przykładów* dla całej populacji i, na jej podstawie, zmieniane są wartości v_+ oraz v_-.

5 Początkowo wartości v_+ oraz v_- wynoszą 1, jednak w trakcie ewolucji są one dopasowywane dynamicznie tak, aby zapewnić jak największą efektywność procesu ewolucji. Zostanie to dokładniej omówione w dalszej części pracy.

6) Co określoną ilość razy przeprowadzenia etapu 4 testowane są warunki zakończenia. Jeśli któryś z nich jest spełniony, proces ewolucji ulega przerwaniu, a jeśli nie, to ponownie wykonywany jest krok 4. Warunki zakończenia mogą mieć różną postać, na przykład możliwe jest uzależnienie zakończenia procesu ewolucji od wartości funkcji dopasowania (liczonej tak, jak w przypadku ewolucji klasycznej) dla osobnika (gramatyki) o największej wartość *poprawności rozpoznania przykładów,* lub dla dowolnego osobnika należącego do populacji. Możliwe jest także uzależnienie zakończenia procesu ewolucji od osiągnięcia zadanych wartości przez v_+ i v_-.

Dodatkową różnicą w stosunku do ewolucji klasycznej jest podział całej populacji liczącej n osobników na k rozłącznych podzbiorów, z których każdy ma określone własne parametry ewolucji. W prezentowanych w niniejszej rozprawie badaniach parametry ewolucji dla poszczególnych podzbiorów różniły się maksymalną wysokością generowanych drzew reprezentujących produkcje gramatyk (parametr **`InitialMaxTreeDepth`**) oraz maksymalną dopuszczalną wysokością powstających w procesie ewolucji drzew reprezentujących produkcje gramatyk (parametr **`MaxTreeDepth`**). Takie zróżnicowanie parametrów ewolucji pozwala na równoczesne poszukiwanie gramatyk o różnym stopniu złożoności poprawnie rozpoznających zadane przykłady języka. Stanowi to przewagę w stosunku do klasycznego modelu ewolucji, w którym parametry ewolucji są jednakowe dla całej populacji gramatyk, co wymaga podjęcia *a priori* decyzji o maksymalnej dopuszczalnej wysokości drzew reprezentujących produkcje gramatyk. Wybranie zbyt małej wartości dla dopuszczalnej wysokości drzew może spowodować, że w procesie ewolucji nie uda się znaleźć żadnej gramatyki poprawnie rozpoznającej zadane przykłady języka (i jednocześnie takiej, dla której wysokość drzewa reprezentującego produkcje jest nie większa niż zadana wartość), natomiast wybranie zbyt dużej wartości powoduje, że powstające gramatyki cechują się niepotrzebnie dużą ilością produkcji, co wpływa niekorzystnie na czas ewolucji oraz utrudnia interpretację znalezionych rozwiązań. Sprawdzenie efektywności modelu ciągłej ewolucji gramatyk i porównanie go z podejściem klasycznym przedstawione zostanie w rozdziale 6.

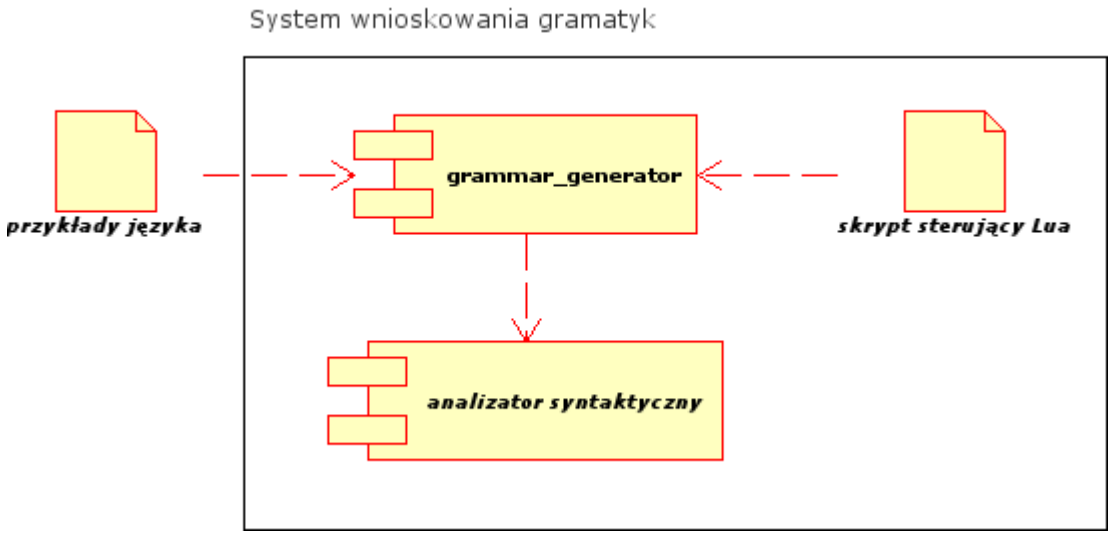
W rozdziale tym zaprezentowany został sposób zastosowania podejścia ewolucyjnego opartego na bazie programowania genetycznego dla celów generowania gramatyk bezkontekstowych na podstawie przykładów języka. Przedstawione zostały podstawowe operacje ewolucyjne – krzyżowanie oraz mutacja – a także sposób pomiaru zgodności wygenerowanych gramatyk z poszukiwanym językiem bezkontekstowym za pomocą funkcji dopasowania. Omówiono różne warianty operacji krzyżowania, których

porównanie, na podstawie osiągniętych wyników, zostanie przedstawione w dalszej części niniejszej pracy. Przedstawiono również dwa modele ewolucji gramatyk: model klasyczny, zgodny ze schematem działania programowania genetycznego przedstawionego w rozdziale 3.2, oraz zaproponowany przez autora model ewolucji 'ciągłej'.

5. Opis systemu

Niniejszy rozdział zawiera ogólny opis implementacji systemu służącego do wnioskowania gramatyk bezkontekstowych na podstawie przykładów języka opisanego w poprzednim rozdziale. Dla celów prezentacji modelu systemu zastosowany został język UML (ang. *Unified Modeling Language*) [6].

Ogólny diagram komponentów przedstawiający prezentowany system wnioskowania gramatyk przedstawiony został na rysunku 5.1.



Rysunek 5.1. Ogólny diagram komponentów przedstawiający system wnioskowania gramatyk

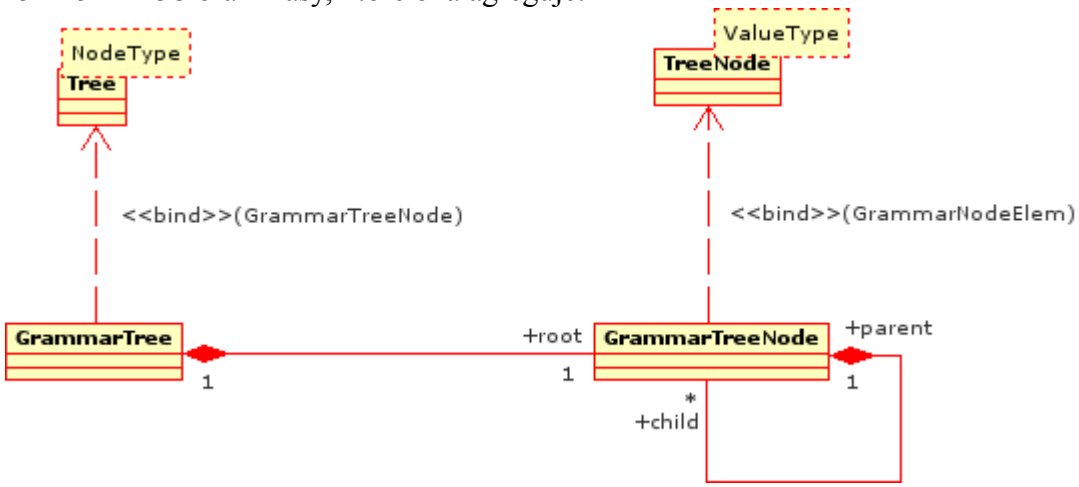
Podstawowy komponent systemu, którym jest plik wykonywalny o nazwie 'grammar_generator', powiązany jest w ramach systemu wnioskowania gramatyk ze skryptem kontrolującym jego działanie ('skrypt sterujący Lua') oraz z analizatorem syntaktycznym, przy użyciu którego testowane są gramatyki generowane w procesie ewolucji. Z komponentem 'grammar_generator' powiązany jest także zewnętrzny komponent 'przykłady języka', który zawiera zbiór przykładów pozytywnych i negatywnych, języka dla którego ma zostać znaleziona gramatyka.

Komponent 'grammar_generator', będący podstawową częścią systemu wnioskowania gramatyk opartego na bazie programowania genetycznego, został zaimplementowany w języku C++ z osadzonym interpreterem języka Lua [31], [75]. Do połączenia interpretera języka Lua z kodem programu w języku C++ została użyta biblioteka '*luabind*' [77]. Zastosowanie osadzonego interpretera miało na celu ułatwienie dokonywania modyfikacji w algorytmach wysokiego poziomu, takich jak algorytm generowania pierwszego pokolenia gramatyk, algorytm selekcji osobników czy też algorytm określający przebieg ewolucji gramatyk. Rozwiązanie takie pozwoliło na

uniknięcie konieczności ponownej kompilacji całości lub części systemu po każdorazowym wprowadzeniu zmian na poziomie, algorytmów wysokiego poziomu co miało istotne znaczenie z punktu widzenia prac badawczych prowadzonych z wykorzystaniem prezentowanego tutaj systemu.

Natomiast obiektowe struktury danych używane w systemie (na przykład struktura drzewa reprezentującego produkcje gramatyki) oraz algorytmy niskiego poziomu (na przykład algorytm tłumaczenia struktury drzewa reprezentującego produkcje gramatyki na postać opisu gramatyki akceptowaną przez generator analizatorów syntaktycznych) zostały utworzone w języku C++, co było podyktowane z jednej strony potrzebą optymalizacji pod kątem szybkości działania oraz zajętości pamięci, a z drugiej strony brakiem konieczności wprowadzania częstych zmian w tej części systemu. Przy tworzeniu systemu wykorzystano częściowo kod źródłowy opracowany wcześniej dla potrzeb środowiska programowania genetycznego GPSource [55].

Podstawową klasą wchodzącą w skład komponentu 'grammar_generator', znajdującą się na najwyższym poziomie abstrakcji, jest klasa `GrammarTree` reprezentująca gramatykę bezkontekstową za pomocą jej zbioru produkcji. Diagram klasowy (ang. *class diagram*) przedstawiony na rysunku 5.2 pokazuje klasę `GrammarTree` oraz klasy, które ona agreguje.



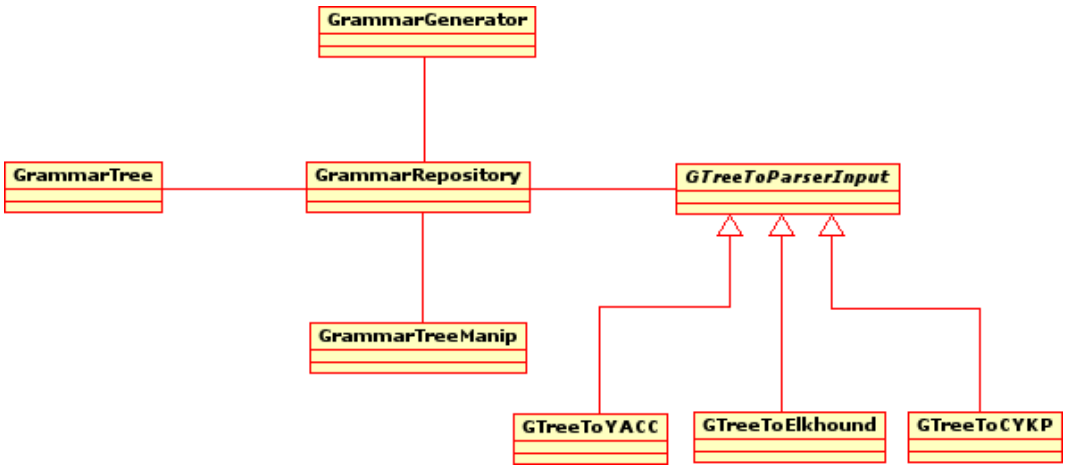
Rysunek 5.2. Diagram klas przedstawiający klasę GrammarTree oraz klasy z nią powiązane

Obiekty klasy `GrammarTreeNode` reprezentują wierzchołki drzewa produkcji gramatyki. Przechowują one informacje o nazwie symbolu oraz o tym, czy jest to symbol terminalny, czy nieterminalny gramatyki[6]. W przypadku, gdy obiekt klasy `GrammarTreeNode` reprezentuje symbol nieterminalny, może on agregować inne obiekty klasy `GrammarTreeNode`, tworząc z nimi relację rodzic – syn (ang. parent –

6 Informacja o nazwie symbolu i jego typie przechowywana jest w polach 'name' oraz 'terminal' klasy `GrammarNodeElem` będącej wartością parametru `ValueType` szablonu `TreeNode` dla klasy `GrammarTreeNode`.

child). Obiekt rodzicielski klasy `GrammarTreeNode` (reprezentujący symbol nieterminalny) wraz z uporządkowanym zbiorem obiektów klasy `GrammarTreeNode`, które bezpośrednio agreguje, reprezentują jedną produkcję gramatyki bezkontekstowej. Przy czym obiekt rodzicielski traktowany jest jako symbol nieterminalny znajdujący się w lewej stronie produkcji, natomiast jego synowie jako symbole terminalne i nieterminalne znajdujące się w prawej stronie produkcji. Obiekt klasy `GrammarTree` reprezentującej drzewo produkcji gramatyki agreguje jeden obiekt klasy `GrammarTreeNode` (typu nieterminalnego) reprezentujący korzeń drzewa (ang. *root*), który traktowany jest jako symbol startowy gramatyki (STS). Więcej informacji na temat sposobu reprezentacji gramatyki za pomocą drzewa produkcji podano wcześniej w rozdziale 4. Klasą pomocniczą wykorzystywaną do definiowania gramatyki wspólnie z `GrammarTree` jest `GrammarRepository`. Klasa ta przechowuje zbiory nazw symboli terminalnych (V_T) oraz nieterminalnych (V_N) wchodzących w skład wszystkich produkcji gramatyk.

Na diagramie 5.3 przedstawione zostały klasy, korzystające z klasy `GrammarRepository`.



Rysunek 5.3. Diagram przedstawiający klasy powiązane z klasą GrammarRepository

Poza omówioną wcześniej klasą `GrammarTree`, z `GrammarRepository` powiązane są następujące klasy:

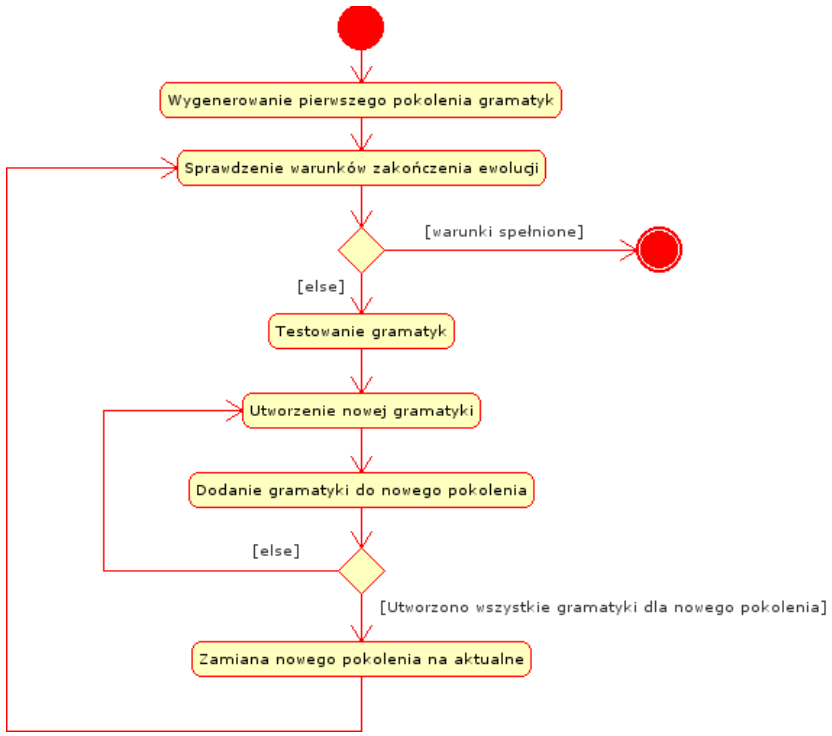
- `GrammarGenerator` – klasa służąca do tworzenia nowych drzew reprezentujących produkcje gramatyk metodami *grow* oraz *full*;
- `GrammarTreeManip` – klasa realizująca operacje genetyczne krzyżowania i mutacji dla drzew reprezentujących produkcje gramatyk;
- `GTreeToParserInput` – klasa zapewniająca tłumaczenie gramatyki z reprezentacji drzewa produkcji (przechowywanej w klasie `GrammarTree`) na

postać pliku wejściowego akceptowanego przez parser lub generator parserów. `GTreeToParserInput` jest klasą abstrakcyjną nieposiadającą implementacji. Natomiast z klasy tej dziedziczą trzy klasy: `GTreeToYACC`, `GTreeToElkhound` oraz `GTreeToCYKP`, które zapewniają tłumaczenie drzewa produkcji odpowiednio na formaty: generatora parserów YACC oraz Bison, generatora parserów Elkhound oraz parsera CYK. Rozdzielenie interfejsu od konkretnej implementacji klasy tłumaczącej przez utworzenie klasy abstrakcyjnej `GTreeToParserInput` pozwala na łatwe rozszerzanie systemu o nowe parsery lub generatory parserów. W celu wykorzystania nowego parsera lub generatora parserów wystarczy stworzyć klasę dziedziczącą z `GTreeToParserInput` i realizującą tłumaczenie z reprezentacji drzewa na format pliku źródłowego nowego parsera.

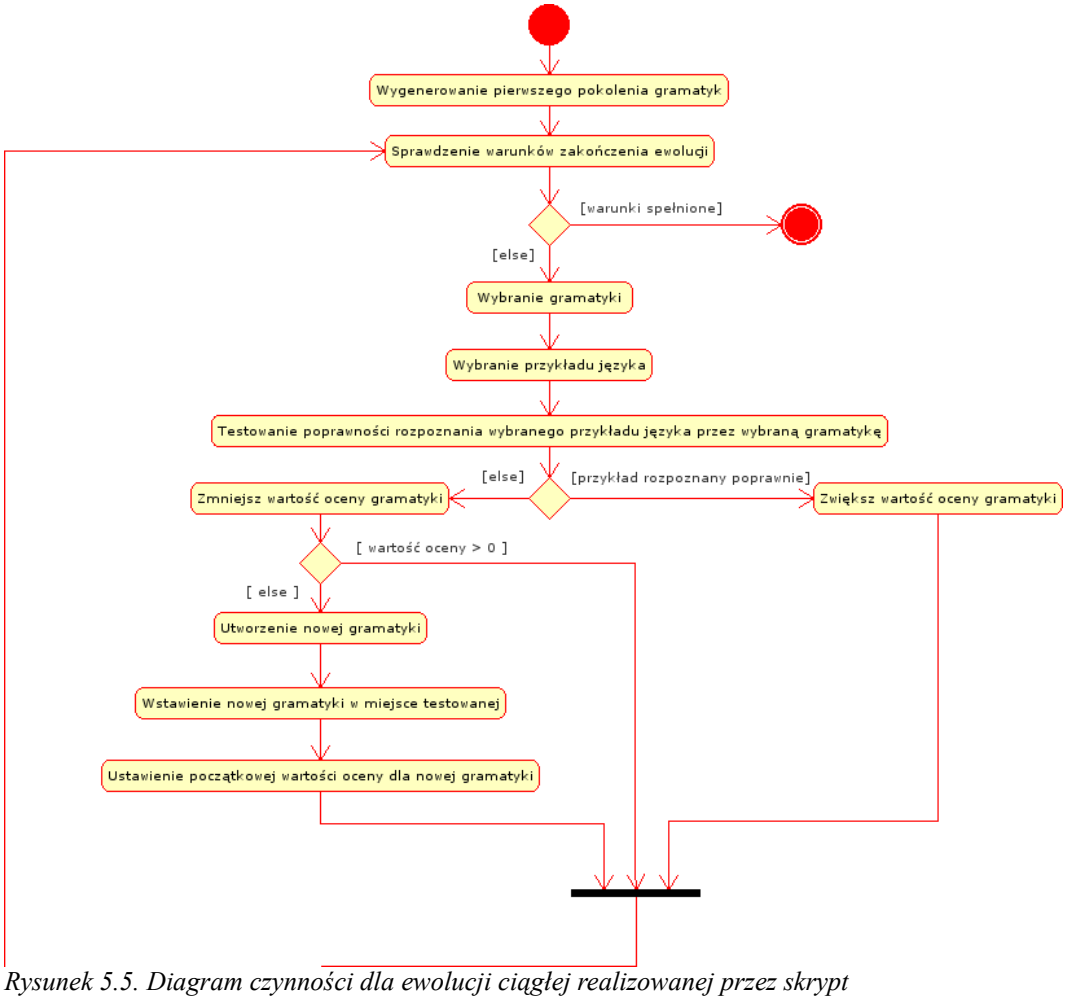
Proces ewolucji gramatyk kontrolowany jest przez skrypt napisany w języku Lua. W przedstawionej tutaj wersji systemu wnioskowania gramatyk istnieją dwa alternatywne skrypty sterujące działaniem komponentu 'grammar_generator':

- evolution_classical.lua – skrypt realizujący klasyczną ewolucję gramatyk, przedstawioną w rozdziale 4.6.1;
- evolution_continuous.lua – skrypt realizujący ewolucję ciągłą gramatyk, opisaną w rozdziale 4.6.2

Diagramy czynności (ang. *activity diagrams*) przedstawiające działanie tych skryptów zostały przedstawione odpowiednio na rysunkach: 5.4 – dla skryptu evolution_classical.lua i 5.5 – dla skryptu evolution_continuous.lua.

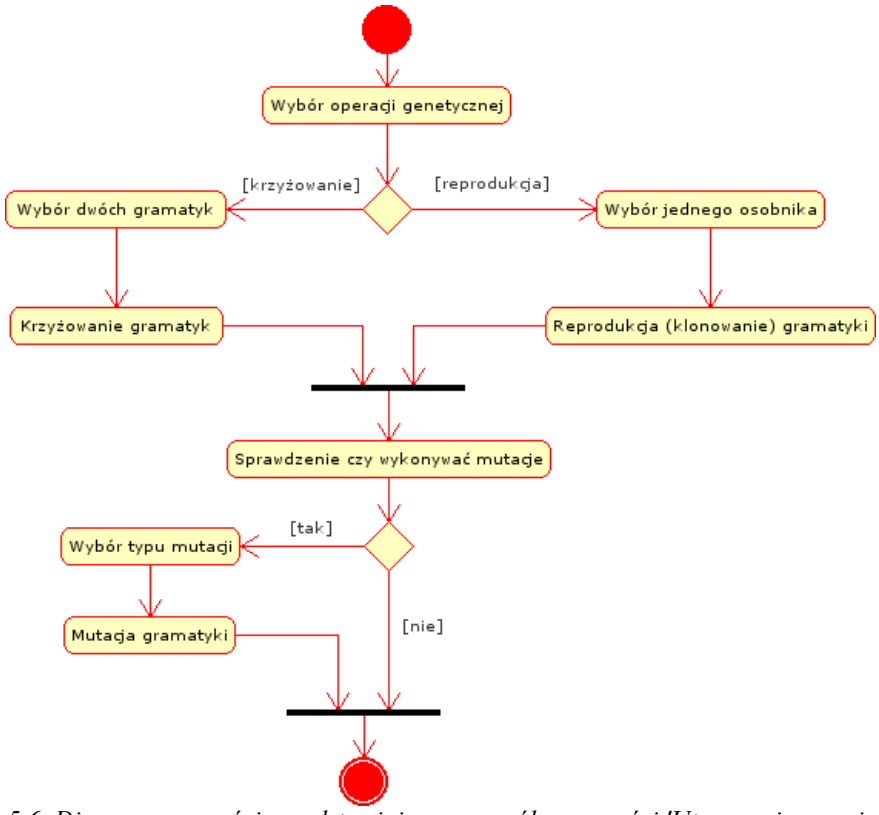


Rysunek 5.4. Diagram czynności dla ewolucji klasycznej realizowanej przez skrypt evolution_classical.lua



Rysunek 5.5. Diagram czynności dla ewolucji ciągłej realizowanej przez skrypt evolution_continuous.lua

Szczegóły czynności 'Utworzenie nowej gramatyki' w przypadku ewolucji klasycznej przedstawione zostały na rysunku 5.6.



Rysunek 5.6. Diagram czynności przedstawiający szczegóły czynności 'Utworzenie nowej gramatyki' w przypadku ewolucji klasycznej

W przypadku ewolucji klasycznej nowe osobniki (gramatyki) tworzone są w wyniku działania jednej z dwóch operacji genetycznych:

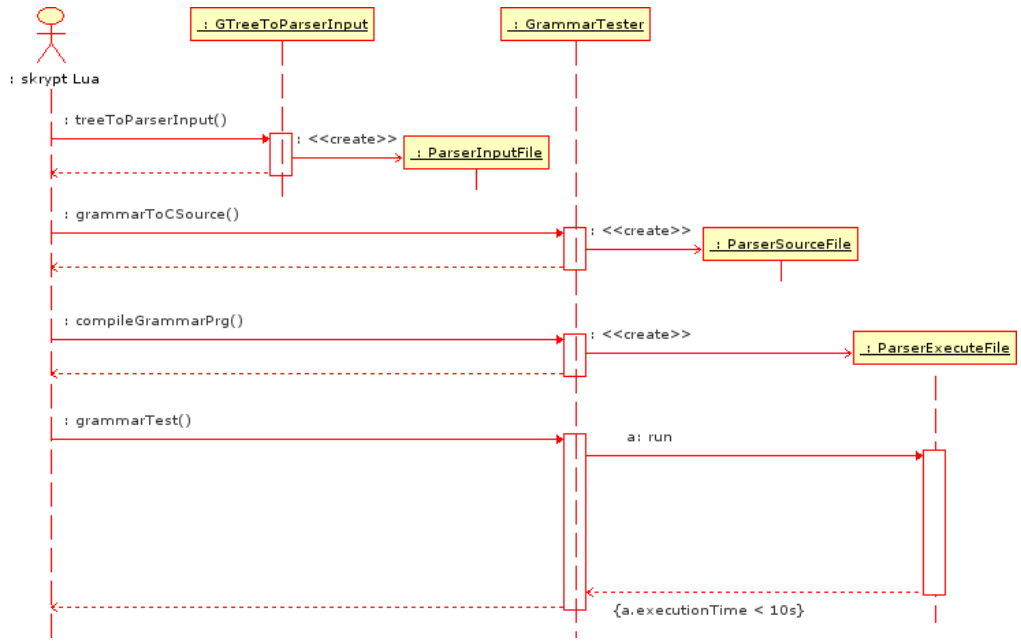
- krzyżowania dwóch osobników rodzicielskich – w wyniku czego powstaje para osobników potomnych;
- reprodukcji osobnika – w wyniku czego do populacji nowego pokolenia kopiowany jest jeden z osobników wchodzących w skład aktualnego pokolenia

Następnie tak powstałe osobniki poddawane są operacji mutacji z zadanym prawdopodobieństwem. W przypadku ewolucji 'ciągłej' nowe osobniki mogą dodatkowo powstawać przez wygenerowanie drzewa reprezentującego produkcję gramatyki, tak jak dzieje się to w przypadku tworzenia pierwszego pokolenia gramatyk.

Przebieg procesu testowania powstałych gramatyk, przedstawiony na diagramach (rysunki 5.4 i 5.5) jako czynności 'Testowanie gramatyk' i 'Testowanie poprawności rozpoznania wybranego przykładu języka przez wybraną gramatykę', uzależniony jest od rodzaju zastosowanego parsera lub generatora parserów. W przypadku użycia generatora parserów Bison lub Elkhound, w pierwszym kroku

gramatyka tłumaczona jest z postaci drzewa produkcji przechowywanej w klasie `GrammarTree` do postaci pliku wejściowego dla generatora parserów zawierającego opis gramatyki – komunikat 'treeToParserInput' na diagramie. Następnie za pomocą generatora parserów z pliku wejściowego tworzony jest kod źródłowy w języku C (dla generatora Bison) lub C++ (dla generatora Elkhound), co dzieje się pod wpływem komunikatu 'grammarToCSource'. W następnym kroku przesyłany jest komunikat 'compileGrammarPrg' do klasy `GrammarTester` powodujący skompilowanie pliku źródłowego, w wyniku czego powstaje plik wykonywalny parsera. W ostatnim kroku, po przesłaniu komunikatu 'grammarTest', uruchamiany jest plik wykonywalny, do którego przekazywane są testowane przykłady języka, a wynik klasyfikacji przykładu języka przez gramatykę przekazywany jest do obiektu wywołującego (do skryptu Lua). Proces testowania gramatyk w przypadku użycia generatorów parserów Bison i Elkhound przedstawiony został na diagramie sekwencji (ang. *sequence diagram*) (rysunek 5.7).

Przebieg czynności 'Testowanie gramatyk' w przypadku zastosowania analizatora syntaktycznego CYKP jest podobny, z tym że pomijane są kroki tworzenia kodu źródłowego w języku C lub C++ (komunikat 'grammarToCSource') oraz kompilacji tego kodu do postaci programu wykonywalnego (komunikat 'compileGrammarPrg'). Zamiast tego utworzony plik z opisem gramatyki jest bezpośrednio przekazywany, razem z przykładami języka, do programu wykonywalnego parsera (CYKP).



Rysunek 5.7. Diagram sekwencji przedstawiający proces testowania gramatyk w przypadku użycia generatorów parserów Bison lub Elkhound.

Kod źródłowy prezentowanego systemu został udostępniony na licencji GNU i dostępny jest na stronie internetowej [76].

6. Wyniki badań

6.1. Porównanie operatorów krzyżowania

Poniżej przedstawiono porównanie wyników działania operatorów krzyżowania zaproponowanych w rozdziale 4, tj.: krzyżowania standardowego, homologicznego i krzyżowania zachowującego rozmiar. W celu wybrania najbardziej odpowiedniego dla celów wnioskowania gramatycznego operatora krzyżowania porównane zostały zarówno przebiegi wartości funkcji przystosowania, zmiany rozmiaru drzew reprezentujących produkcje gramatyki, jak również zróżnicowanie osobników wchodzących w skład populacji mierzone wartością entropii dla poszczególnych typów operatorów.

W przedstawionych poniżej badaniach dotyczących porównania skuteczności różnych rodzajów operatora krzyżowania w procesie ewolucji nie zastosowano operacji mutacji (przyjęto prawdopodobieństwo mutacji równe 0). Miało to na celu wyeliminowanie wpływu operacji mutacji na przebieg procesu ewolucji tak, aby decydującą w niej rolę odgrywały operatory krzyżowania. Prezentowane poniżej badania przeprowadzono na przykładzie generowania gramatyk dla dwóch języków: L_1 oraz L_2. Pierwszy z nich ma postać $L_1 = \{ a^n b^n \text{ dla } n > 0 \}$, natomiast drugi – złożony jest z 10 losowo wygenerowanych słów o długości w zakresie od 1 do 20, składających się z symboli terminalnych **a** oraz **b**. Wybrane języki charakteryzują się znacznym zróżnicowaniem stopnia złożoności. Podczas gdy język L_1, który składa się z nieskończonej ilości słów, można opisać za pomocą gramatyki składającej się tylko z dwóch produkcji, to na ogół do opisu języka L_2, złożonego ze skończonej ilości (w tym przypadku 10) słów, wymagana jest gramatyka posiadająca co najmniej tyle produkcji, ile słów zawiera język L_2. Użyty w procesie wnioskowania zbiór przykładów dla języka L_1 przedstawiony został w tabeli 6.1, natomiast zbiór przykładów dla języka L_2 - w tabeli 6.2. Wartości parametrów kontrolujących proces ewolucji wraz z krótkim opisem ich znaczenia przedstawiono w tabeli 6.3, natomiast dokładniejszy opis poszczególnych parametrów znajduje się w dodatku A.

Tabela 6.1. Przykłady pozytywne i negatywne języka L_1 użyte do generowania gramatyk w celu porównywania operatorów krzyżowania

Przykłady pozytywne (S+)	Przykłady negatywne (S-)
ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, aaaaaabbbbbb, aaaaaaabbbbbbb, aaaaaaaabbbbbbbb, aaaaaaaaabbbbbbbbb	abb, abbb, abbbb, ba, bbaa, bbbaaa, aab, aaab, aaabb, aaabbbb, aaabbbbb, bbba, bbbba, bbbabb, abab, aabbab, aba, abba, aabbbbaaa, bbbabbab, ababa, baa, bbaaaaaa, aabbabbbb, aaaba, aabba, baaabaaabba, abbba, abaa, babba, aabaa, aaabaa, aaaabaa, aaaaabaa, babaabaaaa, bbabaabaaaa, bbbabaabaaaa, bbbbabaabaaaa, aababababa

Tabela 6.2. Przykłady pozytywne i negatywne języka L_2 użyte do generowania gramatyk w celu porównywania operatorów krzyżowania

Przykłady pozytywne (S+)	Przykłady negatywne (S-)
baaaaabbabbbbbababaaa, bbbbbbbbaababaabbb, aabbbbbabbbbaabba, baabbaababababbbbab, bbaa, abaabab, bbbaaabbab, abaaabababbabaabbabab, bbababababaaabaabaabbb, bbbabbbbaabbaaaabba	aabbabbbbbbaabab, aaaaa, a, ba, abababababbaabbaba, bbbab, baaaaabba, bbbbaabbabaabaaabb, aabaaaabaaabb, bababababbbaabaaaabba, baabbbbbabbbbaa, abbababb, aa, bbbababab, baa, babbbaabaa, baabababbbbbbabbab, babbbbabababaabbaabb, aaab, aabbb

Tabela 6.3. Parametry kontrolujące proces ewolucji gramatyk

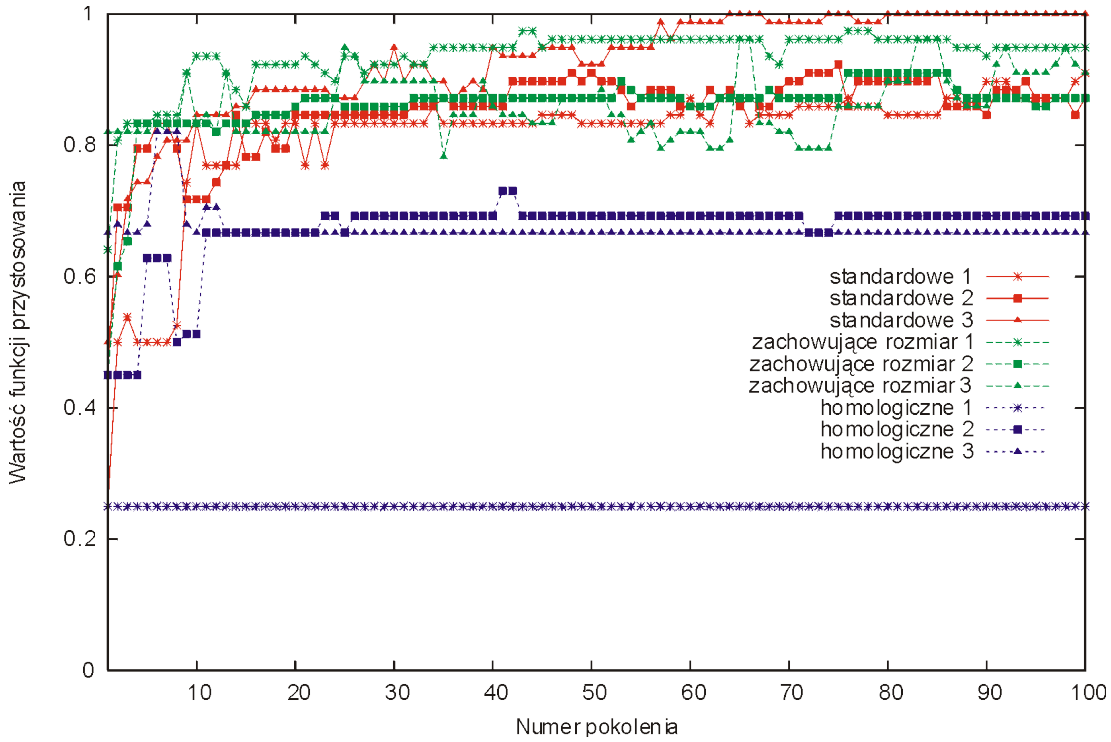
Nazwa parametru	Wartość parametru	Opis
Terminals	a b	Zbiór symboli terminalnych
NTerminals	S N O P Q R	Zbiór symboli nieterminalnych
TotalNoOfGenerations	100	Maksymalna ilość generowanych pokoleń
GenerationSize	100	Ilość osobników w jednym pokoleniu
MaxNoOfNT	4	Maksymalna ilość symboli nieterminalnych które mogą wystąpić w prawej stronie produkcji
MaxNoOfT	4	Maksymalna ilość symboli terminalnych które mogą wystąpić w prawej stronie produkcji

Nazwa parametru	*Wartość parametru*	*Opis*
NTRedefinitionProb	0.8	Prawdopodobieństwo, że dla symbolu nieterminalnego znajdującego się w prawej produkcji zostanie wygenerowane poddrzewo, którego będzie on korzeniem. Parametr ten ma znaczenie dla generowania pierwszego pokolenia metodami *grow* oraz *ramped half and half*
InitialMaxTreeDepth	5	Maksymalna wysokość drzew generowanych w pierwszym pokoleniu
MaxTreeDepth	10	Maksymalna wysokość drzew w drugim i następnych pokoleniach
CrossoverProb	0.8	Prawdopodobieństwo tego, iż osobnik wchodzący w skład nowego pokolenia zostanie utworzony na drodze krzyżowania osobników należących do bieżącego pokolenia. (1 – **CrossoverProb)** określa prawdopodobieństwo, że osobnik wchodzący w skład nowego pokolenia powstanie przez reprodukcję (kopiowanie) osobnika z bieżącego pokolenia
CrossoverOnTerminalPointProb	0.1	Prawdopodobieństwo, że jako punkt krzyżowania dla pierwszego osobnika zostanie wybrany liść drzewa opisującego produkcje gramatyki, czyli, że wysokość pierwszego poddrzewa wymienianego w operacji krzyżowania będzie równa 1 (a więc poddrzewo składa się tylko z jednego wierzchołka). (1 – **CrossoverOnTerminalPointProb**) określa prawdopodobieństwo, że wysokość pierwszego poddrzewa wymienianego w operacji krzyżowania będzie większa od 1.

Pierwsze pokolenie gramatyk we wszystkich przypadkach generowane było przy użyciu metody *ramped half and half* z ustaloną maksymalną wysokością generowanych drzew równą 5 (parametr **InitialMaxTreeDepth**). Na rysunku 6.1 przedstawione zostało porównanie maksymalnych wartości funkcji przystosowania w poszczególnych pokoleniach dla różnych rodzajów operatorów krzyżowania w przypadku generowania gramatyk dla języka L_1, natomiast na rysunku 6.2 – dla języka L_2. Dla każdego rodzaju krzyżowania przedstawiono po trzy niezależne przebiegi ewolucji – odpowiednio dla standardowego operatora krzyżowania: *standardowe 1*, *standardowe 2*, *standardowe 3*; dla operatora krzyżowania zachowującego rozmiar: *zachowujące rozmiar 1*,

zachowujące rozmiar 2, *zachowujące rozmiar 3* oraz dla operatora krzyżowania homologicznego: *homologiczne 1*, *homologiczne 2*, *homologiczne 3*.

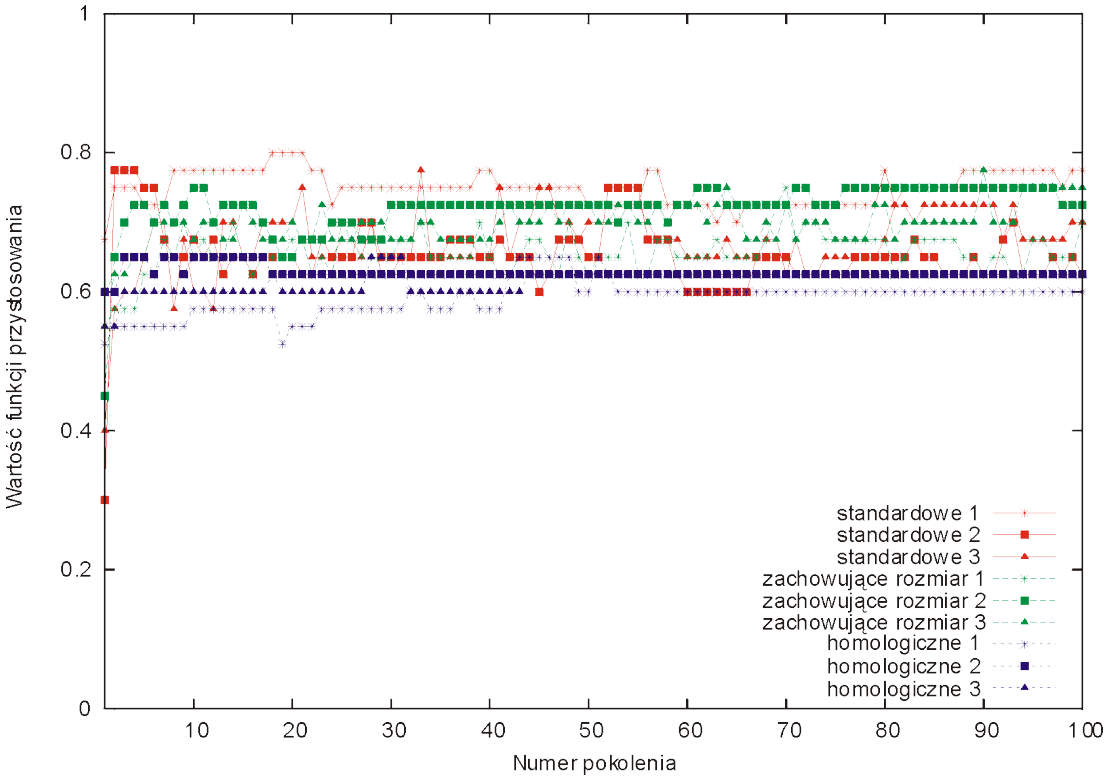
Na wykresie przedstawionym na rysunku 6.1 widać, że w przypadku języka L_1 operatory krzyżowania standardowego oraz krzyżowania zachowującego rozmiar zapewniają porównywalny wzrost wartości funkcji przystosowania najlepszego osobnika, natomiast operator krzyżowania homologicznego daje w tym przypadku znacznie gorsze rezultaty. W przypadku przebiegu pierwszej ewolucji z zastosowaniem operatora krzyżowania homologicznego (oznaczonego na wykresie jako *homologiczne 1*) wartość funkcji przystosowania dla najlepszego osobnika była stała i równa 0.25 przez cały czas trwania ewolucji. W pozostałych dwóch przypadkach zastosowania operatora krzyżowania homologicznego następował wzrost wartości funkcji przystosowania najlepszego osobnika w stosunku do wartości tej funkcji dla pierwszego pokolenia, jednak wartość ta ustalała się ostatecznie na poziomie nieprzekraczającym wartości 0.7.



Rysunek 6.1. Porównanie maksymalnych wartość funkcji przystosowania w poszczególnych pokoleniach dla ewolucji z zastosowaniem różnych rodzajów operatorów krzyżowania w przypadku języka L_1.

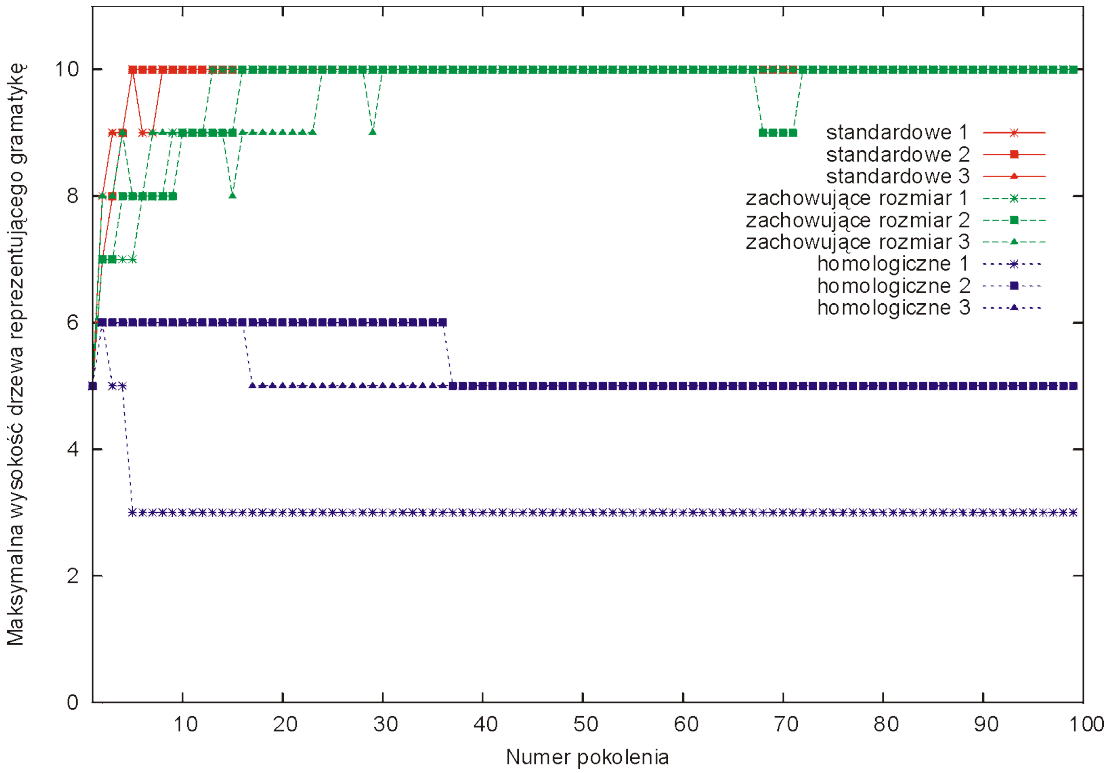
Jak widać na wykresie przedstawionym na rysunku 6.2, również w przypadku języka L_2 operatory krzyżowania standardowego i krzyżowania zachowującego rozmiar dają porównywalne rezultaty, jeśli chodzi o wartość przystosowania najlepszego osobnika w danym pokoleniu. Natomiast operator krzyżowania homologicznego daje

wyniki nieco gorsze. Jednak różnica pomiędzy krzyżowaniem homologicznym i pozostałymi rodzajami operatorów krzyżowania jest znacznie mniejsza niż w przypadku generowania gramatyk dla języka L_1.

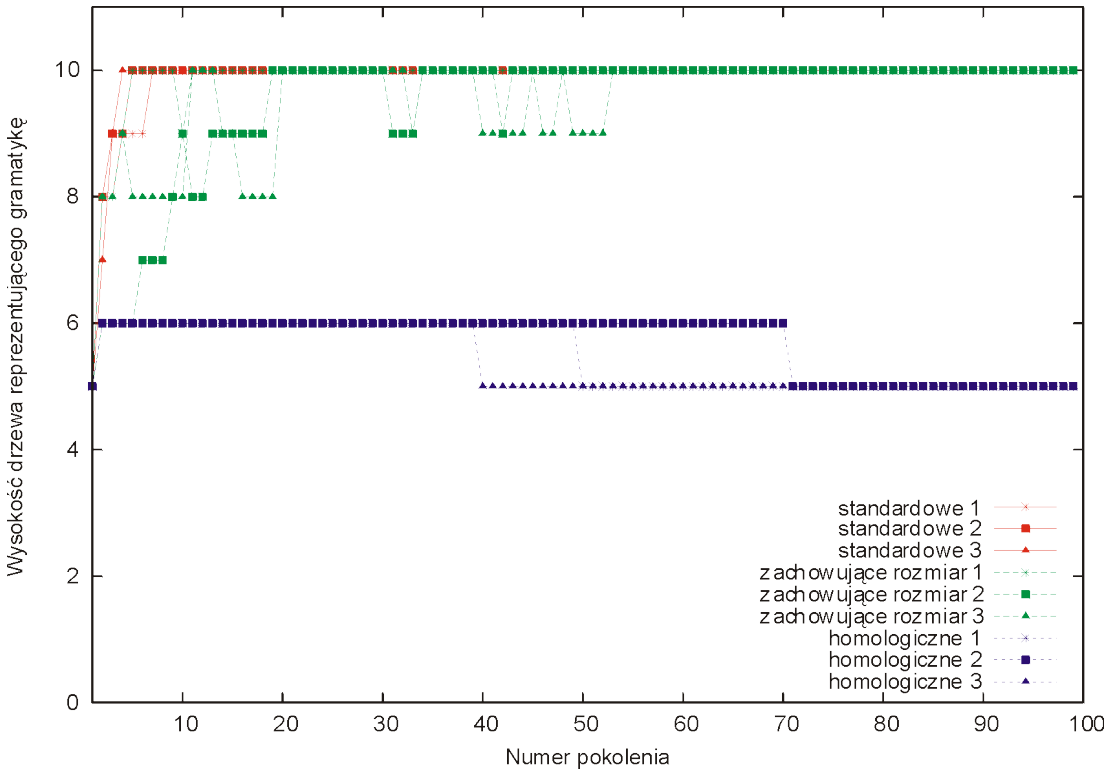


Rysunek 6.2. Porównanie maksymalnych wartość funkcji przystosowania w poszczególnych pokoleniach dla ewolucji z zastosowaniem różnych rodzajów operatorów krzyżowania, w przypadku języka L_2.

Rysunki 6.3 oraz 6.4 przedstawiają wysokość najwyższego drzewa reprezentującego produkcje gramatyki w poszczególnych pokoleniach w zależności od rodzaju użytego operatora krzyżowania w procesie generowania gramatyk dla języków L_1 i L_2. Jak widać na przedstawionych wykresach, operator krzyżowania homologicznego jako jedyny ograniczał wysokość drzew powstających w procesie ewolucji do wartości mniejszej niż maksymalna wartość dozwolona (w tym przypadku 10 – określona przez parametr **`MaxTreeDepth`**). Natomiast zarówno w przypadku ewolucji z zastosowaniem operatora krzyżowania standardowego, jak i krzyżowania zachowującego rozmiar maksymalna wysokość drzewa osiągała największą wartość dopuszczalną, z tym że w przypadku operatora zachowującego rozmiar działo się to wolniej niż w przypadku operatora krzyżowania standardowego.



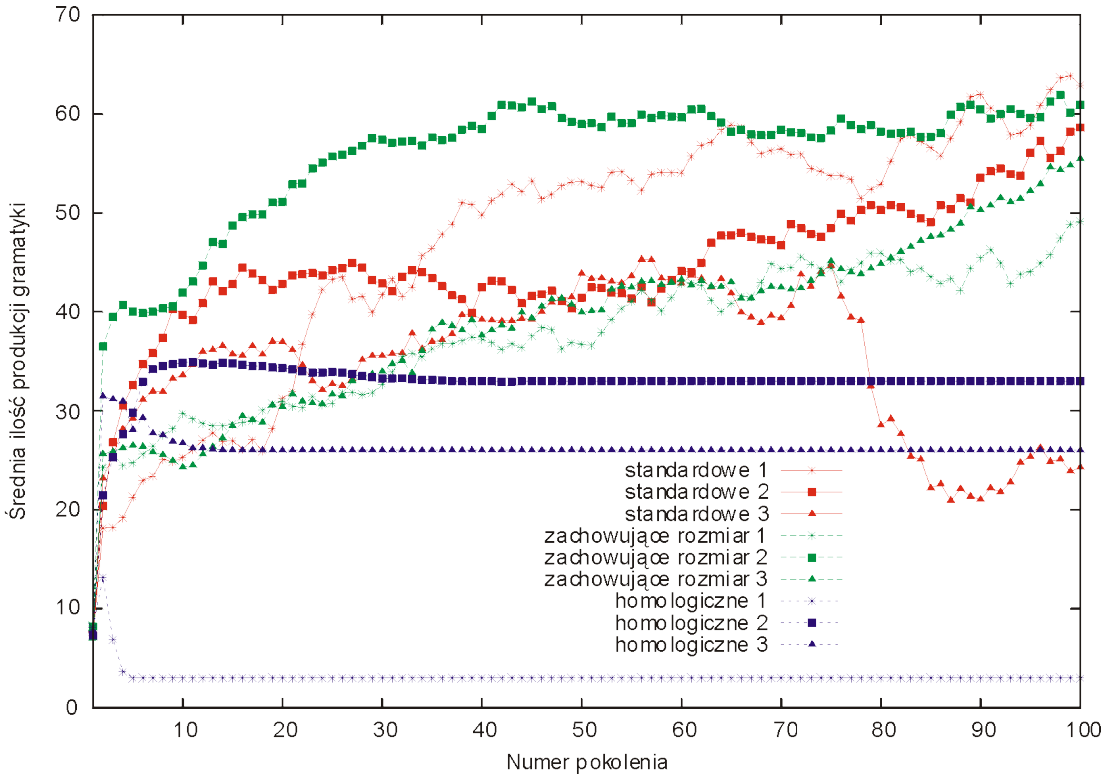
Rysunek 6.3. Maksymalna wysokość drzewa reprezentującego produkcje gramatyki w poszczególnych pokoleniach dla różnych rodzajów operatorów krzyżowania w przypadku generowania gramatyk dla języka L_1



Rysunek 6.4. Maksymalna wysokość drzewa reprezentującego produkcje gramatyki w poszczególnych pokoleniach dla różnych rodzajów operatorów krzyżowania w przypadku generowania gramatyk dla języka L_2

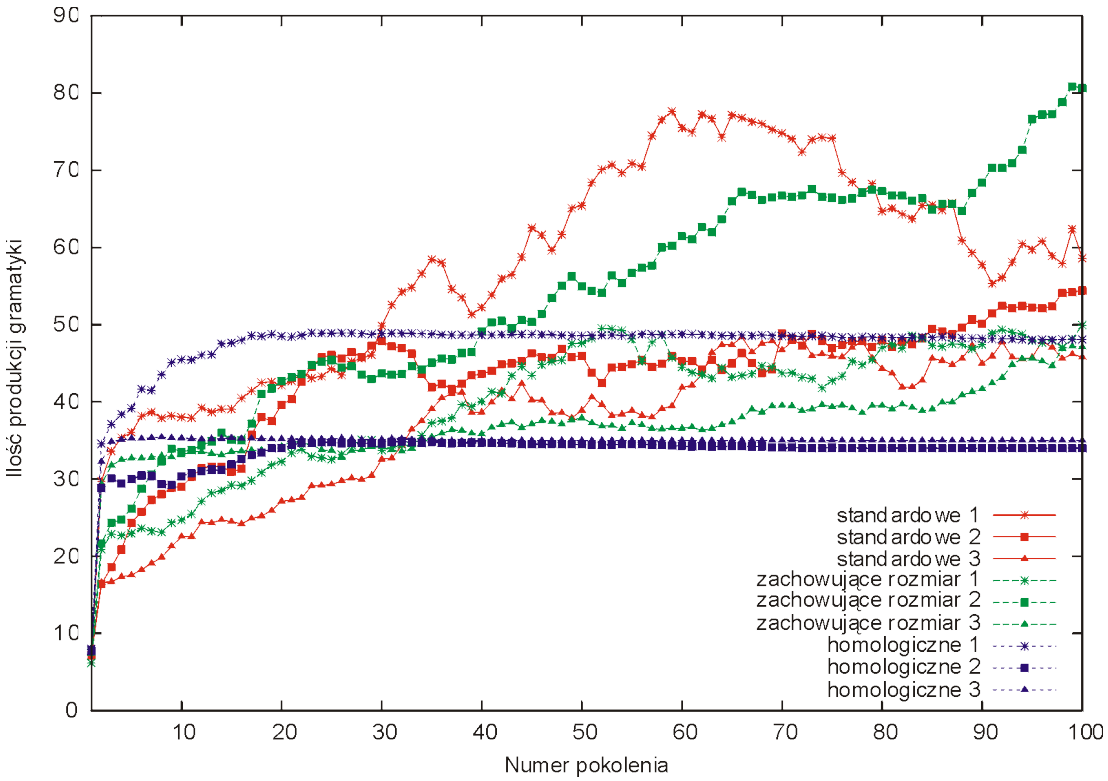
Na rysunkach 6.5 i 6.6 przedstawiona została średnia ilość produkcji przypadających na jedną gramatykę w poszczególnych pokoleniach dla różnych rodzajów operatorów krzyżowania w procesie wnioskowania gramatyk dla języków L_1 i L_2. Z porównania wykresów z rysunków 6.3 i 6.5 widać, że dla języka L_1, mimo iż od pokolenia numer 38 największa wysokość drzewa reprezentującego produkcje gramatyki jest dla operatora krzyżowania homologicznego w przypadku *homologiczne 2* i *homologiczne 3* dwukrotnie mniejsza niż dla operatorów krzyżowania standardowego i zachowującego rozmiar, to średnia ilość produkcji przypadających na jedną gramatykę, pomijając przypadek *standardowe 3,* jest tylko dwukrotnie mniejsza. Natomiast w przypadku *standardowe 3,* dla którego w pokoleniu 64 została znaleziona gramatyka z wartością funkcji przystosowania równą 1, następuje spadek średniej ilości produkcji przypadających na gramatykę tak, że w ostatnim pokoleniu (numer 100) wartość ta jest mniejsza niż w przypadkach *homologiczne 2* i *homologiczne 3.*

Jeśli chodzi o przypadek *homologiczne 1*, dla którego zarówno maksymalna wysokość, jak i średnia ilość produkcji przypadających na gramatykę jest najmniejsza i równa 3, to, jak widać na rysunku 6.1, osiągana maksymalna wartość funkcji przystosowania jest zdecydowanie mniejsza niż w pozostałych przypadkach.



Rysunek 6.5. Średnia ilość produkcji przypadająca na jednego osobnika (gramatykę) w poszczególnych pokoleniach dla różnych rodzajów operatorów krzyżowania w przypadku generowania gramatyk dla języka L_1

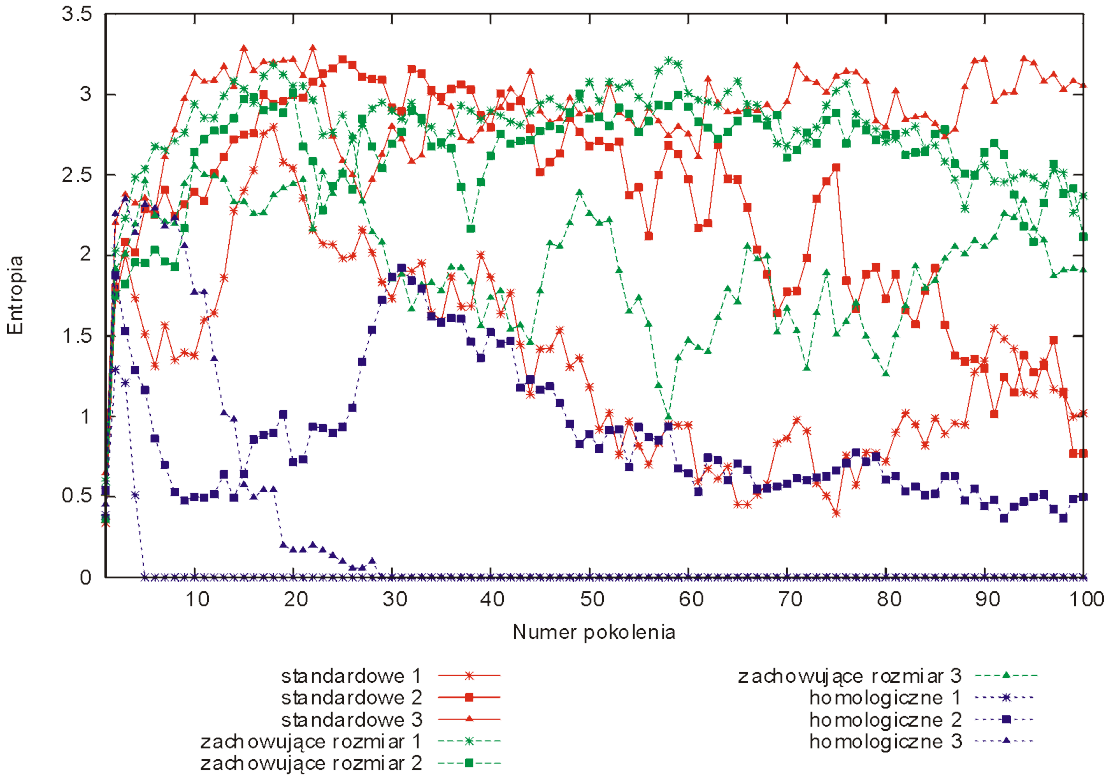
Na wykresie przedstawionym na rysunku 6.6, widać, że podobnie jak w przypadku generowania gramatyk dla języka L_1, także dla języka L_2 operator krzyżowania homologicznego tworzy gramatyki o średniej ilości produkcji tylko niespełna dwukrotnie mniejszej niż w przypadku pozostałych operatorów krzyżowania, mimo iż maksymalna wysokość tworzonych drzew (rysunek 6.4) jest dla niego dwukrotnie mniejsza.



Rysunek 6.6. Średnia ilość produkcji przypadająca na jednego osobnika (gramatykę) w poszczególnych pokoleniach dla różnych rodzajów operatorów krzyżowania w przypadku generowania gramatyk dla języka L_2

Operacja krzyżowania homologicznego powoduje także szybki spadek wartości entropii, co w rezultacie może prowadzić do zdominowania całej populacji przez jednego osobnika, a co za tym idzie, do degeneracji dostępnego w ramach populacji fenotypu i, ostatecznie, do zahamowania wzrostu wartości funkcji przystosowania w kolejnych pokoleniach. Wykres wartości entropii w poszczególnych pokoleniach dla różnych typów operatorów krzyżowania w przypadku wnioskowania gramatyk dla języka L_1 przedstawiony został na rysunku 6.7. Widać na nim, że w dwóch przypadkach na trzy (tj. w przypadkach *homologiczne 1* i *homologiczne 3*) dla krzyżowania homologicznego następuje spadek entropii do wartości 0. Sytuacja taka jest niezwykle niekorzystna z punktu widzenia dalszej ewolucji, ponieważ dostępne są tylko osobniki z jednakowym fenotypem (w prezentowanym tutaj przypadku wszystkie osobniki są wręcz identyczne), co znacząco ogranicza możliwość powstania w wyniku operacji

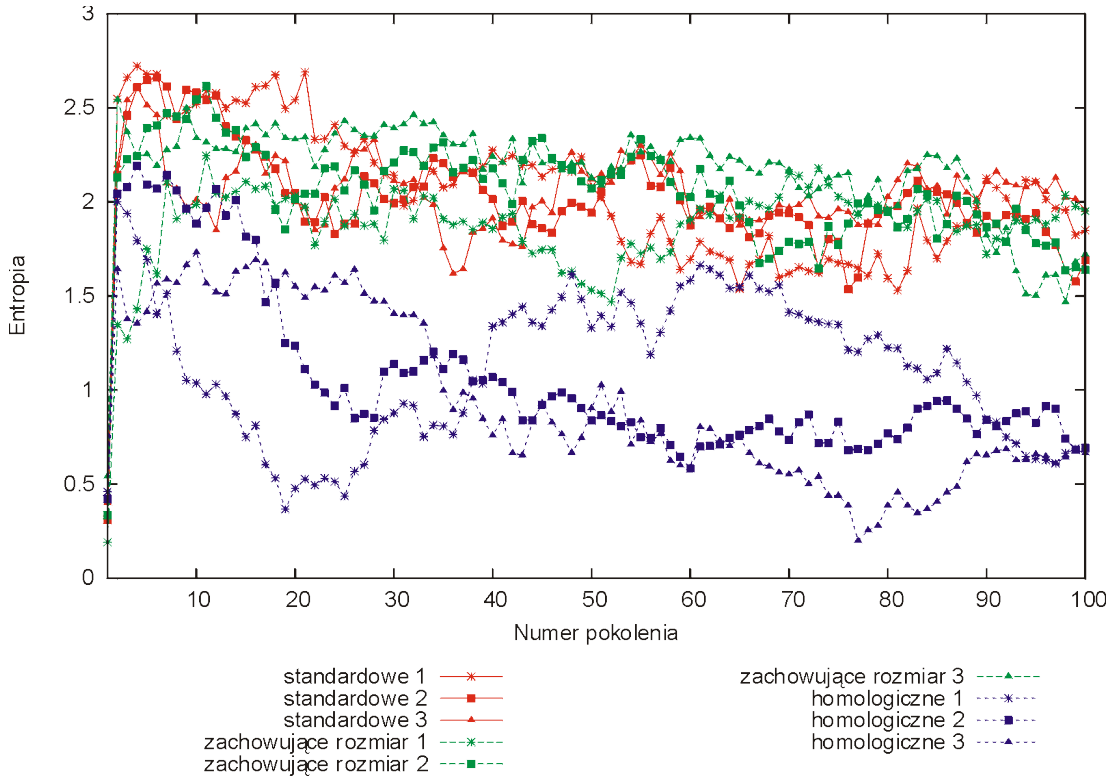
krzyżowania osobników lepiej przystosowanych. Dodatkowo, jeśli tak jak w występującym tutaj przypadku wszystkie osobniki w sytuacji, gdy entropia ma wartość 0, są identyczne, to krzyżowanie homologiczne nie może doprowadzić do powstania innych osobników. Wynika to ze sposobu działania operatora krzyżowania homologicznego, który wybiera punkt krzyżowania dla drugiego osobnika w taki sposób, aby jego pozycja względem korzenia drzewa była jak najbardziej zbliżona do pozycji punktu krzyżowania dla pierwszego osobnika. W przypadku krzyżowania jednakowych osobników powoduje to, że powstające w wyniku tej operacji osobniki są identyczne z tymi, które zostały poddane krzyżowaniu[7].



Rysunek 6.7. Porównanie wartości entropii w poszczególnych pokoleniach dla różnych typów operatorów krzyżowania w przypadku generowania gramatyk dla języka L_1

Rysunek 6.8 przedstawia wartość entropii w poszczególnych pokoleniach w przypadku generowania gramatyk dla języka L_2. Także w tym przypadku populacje tworzone za pomocą operatora krzyżowania homologicznego posiadają znacznie mniejszą entropię niż populacje utworzone z zastosowaniem operatorów krzyżowania standardowego i zachowującego rozmiar.

7 Zasada działania operatora krzyżowania homologicznego została dokładniej omówiona w rozdziale 4.5.1



Rysunek 6.8. Porównanie wartości entropii w poszczególnych pokoleniach dla różnych typów operatorów krzyżowania w przypadku generowania gramatyk dla języka L_2

Wnioski

Mimo iż badania porównujące efekty działania różnych rodzajów operatorów krzyżowania przeprowadzono dla dwóch języków różniących się w sposób zasadniczy stopniem ogólności, to uzyskane wyniki są dla nich podobne. Pozwala to sformułować następujące ogólne wnioski na temat przydatności poszczególnych rodzajów operatorów krzyżowania dla potrzeb prezentowanego w niniejszej rozprawie sposobu ewolucyjnego generowania gramatyk:

1) Pomimo tego, że operator krzyżowania homologicznego sprawia, iż powstające w wyniku jego działania osobniki (gramatyki) są znacząco mniejsze niż osobniki powstałe w wyniku działania operatorów krzyżowania standardowego i zachowującego rozmiar, co jest korzystne z punktu widzenia ewolucji, to jednak ze względu na szybką degenerację fenotypu populacji, odzwierciedlającą się w spadku wartości entropii nawet do wartości 0, nie jest on odpowiedni dla przedstawionego tutaj sposobu generowania gramatyk.
2) Z pozostałych dwóch typów operatorów krzyżowania – tj. standardowego i zachowującego rozmiar – które dają porównywalne rezultaty, zarówno jeśli chodzi o wzrost wartości funkcji przystosowania, jak i o wzrost rozmiaru drzew reprezentujących produkcje gramatyki w trakcie ewolucji, do dalszych badań wybrany został operator krzyżowania zachowującego rozmiar. Decyzja ta

podyktowana została tym, iż działanie operatora krzyżowania zachowującego rozmiar jest mniej chaotyczne niż działanie standardowego operatora krzyżowania, dla którego wymieniane fragmenty drzew dwóch osobników są wybierane w sposób zupełnie przypadkowy. W związku z tym operator krzyżowania zachowującego rozmiar ma na ogół mniej destrukcyjne działanie na fenotyp osobnika, co czyni proces poszukiwania rozwiązania lepiej zbieżnym niż w wypadku zastosowania operatora krzyżowania standardowego. Dodatkowo, jak to widać na rysunkach 6.3 oraz 6.4, operator krzyżowania zachowującego rozmiar powoduje nieco wolniejszy wzrost maksymalnej wysokości drzewa reprezentującego gramatykę w stosunku do operatora krzyżowania standardowego, co również jest korzystne z punktu widzenia procesu poszukiwania rozwiązań, gdyż ogranicza nadmierny przyrost wielkości osobników [35].

6.2. Porównanie operatorów mutacji

W rozdziale tym przedstawiono wyniki badań dotyczących przydatności poszczególnych rodzajów operatorów mutacji przedstawionych w rozdziale 4.5.2. Porównanie przeprowadzone zostało na bazie języka L_1 z rozdziału 6.1. Wykorzystano przykłady języka L_1 przedstawione w tabeli 6.1. W przedstawionych poniżej badaniach porównujących poszczególne rodzaje operatorów mutacji nie zastosowano operacji krzyżowania w celu wyeliminowania jej wpływu na uzyskiwane wyniki ewolucji, co spowodowało, że decydującą rolę w procesie ewolucji odgrywała operacja mutacji.

W tabeli 6.4 przedstawione zostały wartości parametrów kontrolujących przebieg ewolucji. Wartości te są takie same jak te, które zostały użyte dla porównania różnych operatorów krzyżowania, z tym że wartość parametru *CrossoverProb* jest równa 0 (brak operacji krzyżowania w procesie ewolucji). Uwzględnione zostały też dwa parametry mające znaczenie dla działania operacji mutacji: *NoOfMutationTry* oraz *MutationProbabilityOnOneTry.*

Tabela 6.4. Parametry kontrolujące proces ewolucji gramatyk

Nazwa parametru	***Wartość parametru***	***Opis***
Terminals	a b	Zbiór symboli terminalnych
NTerminals	S N O P Q R	Zbiór symboli nieterminalnych
TotalNoOfGenerations	100	Maksymalna ilość generowanych pokoleń
GenerationSize	100	Ilość osobników w jednym pokoleniu

Nazwa parametru	*Wartość parametru*	*Opis*
MaxNoOfNT	4	Maksymalna ilość symboli nieterminalnych, które mogą wystąpić w prawej stronie produkcji
MaxNoOfT	4	Maksymalna ilość symboli terminalnych, które mogą wystąpić w prawej stronie produkcji
NTRedefinitionProb	0.8	Prawdopodobieństwo, że dla symbolu nieterminalnego znajdującego się w prawej stronie produkcji zostanie wygenerowane poddrzewo, którego będzie on korzeniem. Parametr ten ma znaczenie dla generowania pierwszego pokolenia metodami *grow* oraz *ramped half and half*
InitialMaxTreeDepth	5	Maksymalna wysokość drzew generowanych w pierwszym pokoleniu
MaxTreeDepth	10	Maksymalna wysokość drzew w drugim i następnych pokoleniach
CrossoverProb	0	Prawdopodobieństwo tego, iż osobnik wchodzący w skład nowego pokolenia zostanie utworzony w drodze krzyżowania osobników należących do bieżącego pokolenia. (1 – **CrossoverProb**) określa prawdopodobieństwo, że osobnik wchodzący w skład nowego pokolenia powstanie przez reprodukcję (kopiowanie) osobnika z pokolenia bieżącego
CrossoverOnTerminalPointProb	Nieistotna, ponieważ w procesie ewolucji nie występuje krzyżowanie (**CrossoverProb** = 0)	Prawdopodobieństwo, że jako punkt krzyżowania dla pierwszego osobnika zostanie wybrany liść drzewa opisującego produkcje gramatyki, czyli że wysokość pierwszego poddrzewa wymienianego w operacji krzyżowania będzie równa 1 (a więc poddrzewo składa się tylko z jednego wierzchołka). (1 – **CrossoverOnTerminalPointProb**) określa prawdopodobieństwo, że wysokość pierwszego poddrzewa wymienianego w operacji krzyżowania będzie większa od 1

Nazwa parametru	*Wartość parametru*	*Opis*
NoOfMutationTry	2	Ilość prób mutacji przeprowadzona dla nowo powstałego osobnika (w wyniku krzyżowania lub reprodukcji). Jest to maksymalna ilość mutacji, którym może zostać poddany nowo powstały osobnik
MutationProbabilityOnOneTry	0.05	Prawdopodobieństwo, z jakim dla jednej próby zostanie wykonana operacja mutacji. Iloczyn **NoOfMutationTry * MutationProbabilityOnOneTry** określa prawdopodobieństwo wykonania co najmniej jednej mutacji dla danego osobnika. W tym wypadku prawdopodobieństwo to wynosi 0.1 czyli 10%

Dla każdego rodzaju operatorów mutacji przedstawionych w rozdziale 4.5.2 tj. dla: *transwersji*, *delecji*, *addycji terminala*, *addycji nieterminala* oraz *tranzycji nieterminala,* przeprowadzono po trzy niezależne przebiegi ewolucji, oznaczone na dalej zaprezentowanych rysunkach jako: *ewolucja nr 1*, *ewolucja nr 2* oraz *ewolucja nr 3*. Dodatkowo przedstawiono wyniki przebiegu trzech procesów ewolucji z zastosowaniem wszystkich pięciu rodzajów operatorów mutacji. W przypadku tym stosowany rodzaj operatora mutacji wybierany był z jednakowym prawdopodobieństwem ze wszystkich pięciu rodzajów dla każdej próby mutacji.

6.2.1. Struktura drzew generowanych w pierwszym pokoleniu

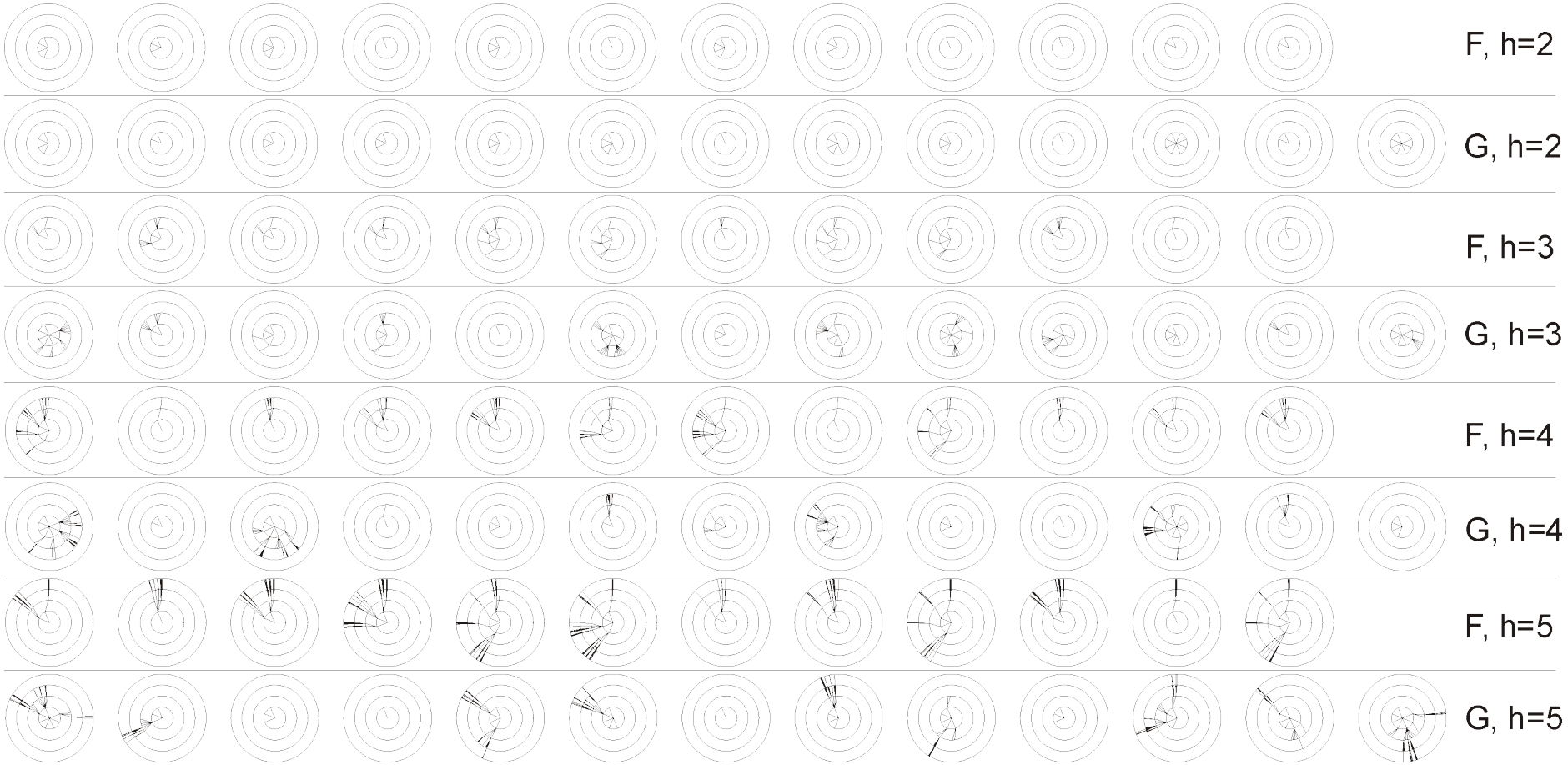
Na rysunku 6.9 przedstawiona została przykładowa struktura drzew reprezentujących produkcje gramatyki wygenerowanych w pierwszym pokoleniu.

Ponieważ generowanie osobników dla pierwszego pokolenia odbywa się za pomocą metody *ramped half and half* z maksymalną wysokością generowanych drzew równą 5 (parametr **InitialMaxTreeDepth**), więc zgodnie z zasadą działania tej metody 25% populacji (liczącej 100 osobników) będzie miało maksymalną wysokość drzew równą 2, 25% – wysokość 3, 25% – wysokość 4 oraz 25% – wysokość 5. Przy czym połowa osobników, o określonej maksymalnej wysokości, jest generowana przy pomocy metody full, a druga połowa przy pomocy metody grow[8].

Na rysunku 6.9, w poszczególnych wierszach, przedstawione są osobniki o zadanej maksymalnej wysokości drzewa określonej przez parametr h (w zakresie od 2 do 5), generowane metodami *full* (oznaczone literą F), oraz *grow* (oznaczone literą G).

8 Ponieważ 25 nie dzieli sie całkowicie przez 2, 12 osobników generowanych jest za pomocą metody full, natomiast 13 za pomocą metody grow

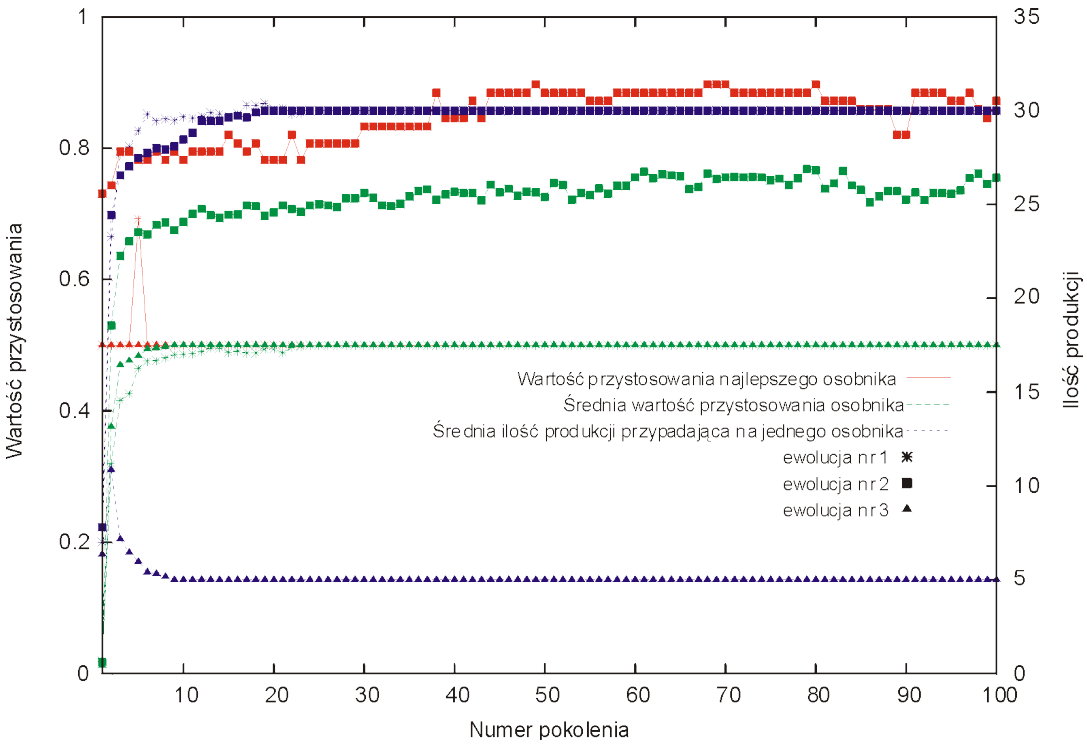
Jak widać na przedstawionym przykładzie, struktura drzew reprezentujących produkcje gramatyki wygenerowanych w pierwszym pokoleniu charakteryzuje się znaczną różnorodnością.



Rysunek 6.9. Przykład struktury drzew reprezentujących produkcje gramatyk wygenerowanych metodą ramped half and half w pierwszym pokoleniu. Parametr h oznacza maksymalną dopuszczalną wysokość drzewa, F oznacza generowanie metodą full, natomiast G generowanie metodą grow

6.2.2. Operator mutacji typu *transwersja*

Transwersja jest operacją mutacji polegającą na zamianie dwóch symboli występujących w prawej stronie pewnej produkcji. Na rysunku 6.10 przedstawiony został wykres pokazujący wartość przystosowania najlepszego osobnika, średniego przystosowania osobnika oraz średniej ilości produkcji gramatyki przypadającej na jednego osobnika w poszczególnych pokoleniach dla trzech niezależnych przebiegów ewolucji gramatyk. Jak wynika z przedstawionego wykresu, tylko w przypadku ewolucji nr 2 następuje wzrost wartości przystosowania najlepszego osobnika osiągając wartość ok. 0.82, natomiast w przypadku ewolucji nr 1 i nr 3 wartość przystosowania najlepszego osobnika jest w zasadzie stała i równa 0.5.



Rysunek 6.10. Wyniki ewolucji z zastosowaniem operatora transwersji

Podobnie zachowuje się średnia wartość przystosowania osobnika – tylko w przypadku ewolucji nr 2 wykazuje ona trend wzrostowy przez cały okres trwania ewolucji, natomiast w wypadku ewolucji nr 1 i nr 3 szybko (w pokoleniu nr 10 dla ewolucji 3 i w pokoleniu nr 24 dla ewolucji 1) osiąga ona wartość stałą równą 0.5.

Jeśli chodzi o średnią ilość produkcji przypadających na jedną gramatykę, to w przypadku ewolucji nr 1 i nr 2 rośnie ona szybko, osiągając w pokoleniu nr 25 wartość stałą równą 30, natomiast w przypadku ewolucji nr 3 średnia ilość produkcji początkowo rośnie, a następnie maleje, osiągając w pokoleniu nr 10 wartość 5.

Należy zauważyć, że zarówno wzrost, jak i spadek średniej ilości produkcji przypadających na jednego osobnika (gramatykę) związany jest z procesem selekcji osobników, a nie z działaniem operatora mutacji. Wynika to z zasady działania operatora transwersji, który jedynie zamienia miejscami dwa symbole znajdujące się w prawej stronie produkcji, co nie może zmieniać ani ilości produkcji gramatyki, ani też wysokości drzewa reprezentującego te produkcje.

Na rysunku 6.11 przedstawiona została struktura drzew reprezentujących produkcje gramatyk w ostatnim pokoleniu – tj. w pokoleniu nr 100 dla ewolucji nr 1. Widać na nim, że wszystkie drzewa wchodzące w skład ostatniego pokolenia mają wysokość równą 5, czyli taką, jaka była największa wysokość drzew generowanych w pierwszym pokoleniu. Dodatkowo można zauważyć, że struktura wszystkich drzew wchodzących w skład ostatniego pokolenia jest bardzo podobna i odpowiada strukturze drzewa przedstawionego na rysunku 6.9 znajdującego się na pierwszej pozycji od prawej w wierszu oznaczonym F, h = 5.

W przypadku ewolucji nr 1 i ewolucji nr 3 następuje stosunkowo szybko spadek wartości entropii do 0, co można wywnioskować na podstawie pokrycia się wartości przystosowania najlepszego osobnika ze średnią wartością przystosowania osobnika. Jak zostało to już stwierdzone przy porównaniu różnych typów operatorów krzyżowania, jest to zjawisko niekorzystne.

Podsumowując – działanie operatora transwersji nie prowadzi do istotnego wzrostu wartości przystosowania najlepszego osobnika, natomiast wzrost średniej wartości przystosowania osobnika występujący w początkowych pokoleniach spowodowany jest głównie przez proces selekcji osobników, a nie przez działanie operatora transwersji. Czasami jednak zamiana miejscami symboli znajdujących się w prawej stronie produkcji może prowadzić do powstania osobników lepiej przystosowanych niż osobniki przed zamianą. Dlatego też operator ten będzie stosowany w procesie ewolucji, ale ze stosunkowo niskim prawdopodobieństwem jego wyboru.

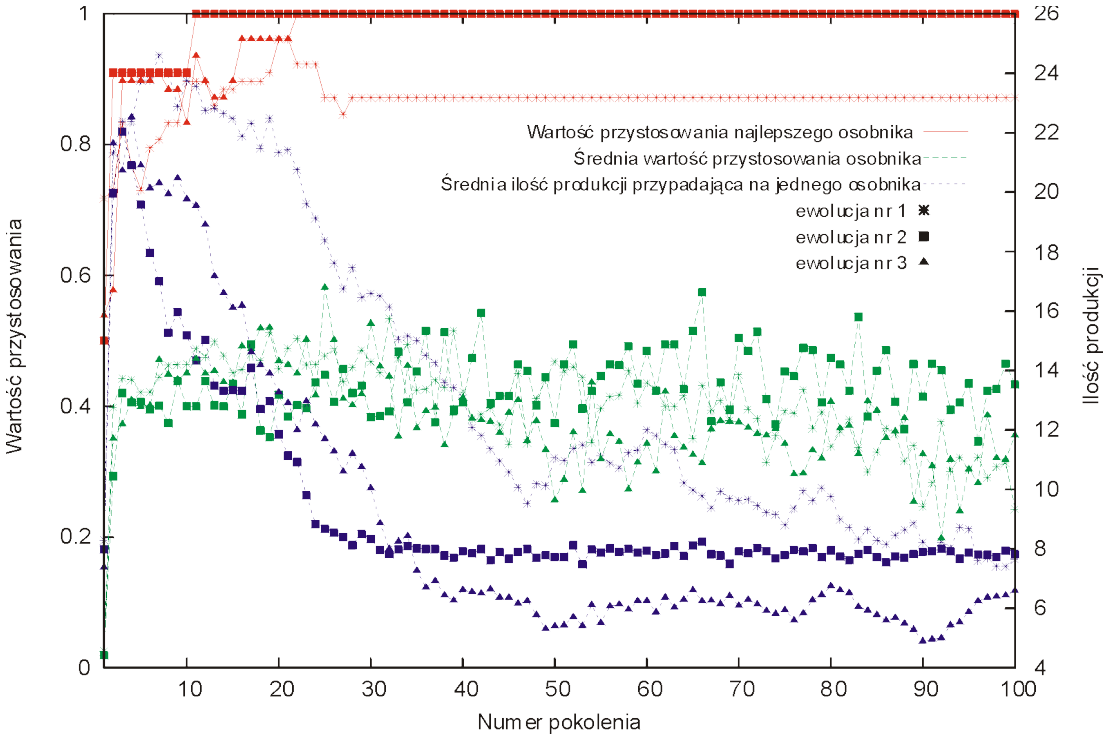
Rysunek 6.11. Struktura drzew reprezentujących produkcje gramatyk w ostatnim pokoleniu dla ewolucji z użyciem operatora transwersji

6.2.3. Operator mutacji typu d*elecja*

Operacja delecji polega na usunięciu jednego z synów dla wybranego wierzchołka drzewa. Powoduje to usunięcie jednego symbolu znajdującego się w prawej stronie produkcji oraz w przypadku, gdy usuwany wierzchołek był korzeniem poddrzewa, usunięciem jednej lub większej ilości produkcji.

Rysunek 6.12 przedstawia wartości przystosowania najlepszego osobnika, średniego przystosowania osobnika oraz średniej ilości produkcji przypadającej na jednego osobnika (gramatykę) w poszczególnych pokoleniach dla trzech przebiegów ewolucji z zastosowaniem operatora mutacji typu delecja. Przedstawiony wykres pokazuje, iż w dwóch przypadkach na trzy (tj. w przypadku ewolucji nr 2 i nr 3) operator delecji prowadzi do wzrostu wartości przystosowania najlepszego osobnika aż do wartości 1, czyli do znalezienia gramatyki poprawnie klasyfikującej zarówno wszystkie zadane przykłady pozytywne, jak i wszystkie zadane przykłady negatywne języka.

Na wykresie przedstawionym na rysunku 6.12 widać, że średnia wartość przystosowania osobnika, we wszystkich trzech przypadkach ewolucji, oscyluje w pobliżu wartości 0.4, natomiast średnia ilość produkcji przypadających na gramatykę zmniejsza się od początkowych wartości ponad 22 do wartości około 8 w ostatnim pokoleniu.



Rysunek 6.12. Wyniki ewolucji z zastosowaniem operatora delecji

Z przedstawionych tutaj wyników ewolucji z zastosowaniem operatora delecji nasuwa się wniosek o znacznej przydatności tego operatora w procesie wnioskowania gramatyk bazującym na programowaniu genetycznym. Jest to z jednej strony spowodowane znacznym wzrostem przystosowania najlepszego osobnika, a z drugiej spadkiem średniej ilości produkcji przypadających na osobnika (gramatykę), co prowadzi do uproszczenia gramatyk podlegających ewolucji. Uproszczenie to jest korzystne, zarówno jeśli chodzi o sam przebieg ewolucji, jako że osobniki z mniejszą ilością produkcji wymagają mniej czasu na obliczenie dla nich wartości funkcji przystosowania, jak i z punktu widzenia ostatecznie znalezionych rozwiązań postawionego problemu, ponieważ gramatyki z mniejszą ilością produkcji są na ogół łatwiejsze w interpretacji niż gramatyki charakteryzujące się większą ilością produkcji.

W przypadku języków, dla których istnieją gramatyki o małym stopniu skomplikowania, tak jak dla zastosowanego tutaj języka L_1, użycie samego operatora delecji w procesie ewolucji może doprowadzić do znalezienia poszukiwanej gramatyki (tak jak w przedstawionych tu przypadkach ewolucji nr 2 i ewolucji nr 3).

Na rysunku 6.13 przedstawiona została przykładowa gramatyka stanowiąca rozwiązanie zadanego problemu[9] znaleziona w ostatnim pokoleniu w przypadku ewolucji nr 2, natomiast na rysunku 6.14 – gramatyka znaleziona w ostatnim pokoleniu w przypadku ewolucji nr 3.

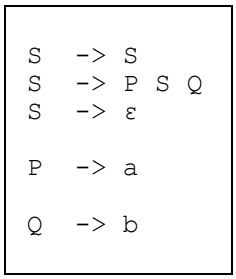
```
S  -> O Q
S  -> R

N  -> N
N  -> a

O  -> N

P  -> P
P  -> b

Q  -> S P

R  -> ε [10]
```

Rysunek 6.13. Przykładowa gramatyka[11], stanowiąca rozwiązanie zadanego problemu znaleziona w ostatnim pokoleniu w ewolucji nr 2

9 To znaczy gramatyka poprawnie rozpoznająca zarówno przykłady pozytywne jak i negatywne zawarte w tabeli 6.1.

10 ε – oznacza symbol pusty.

11 Na rysunkach zostały przedstawione same produkcje gramatyk, zbiory symboli terminalnych i nieterminalnych określone zostały w tabeli 6.4. Symbol startowy gramatyki to pierwszy symbol występujący na liście parametrów **`Nterminals`** (w prezentowanych tu przykładach jest to symbol **S**).

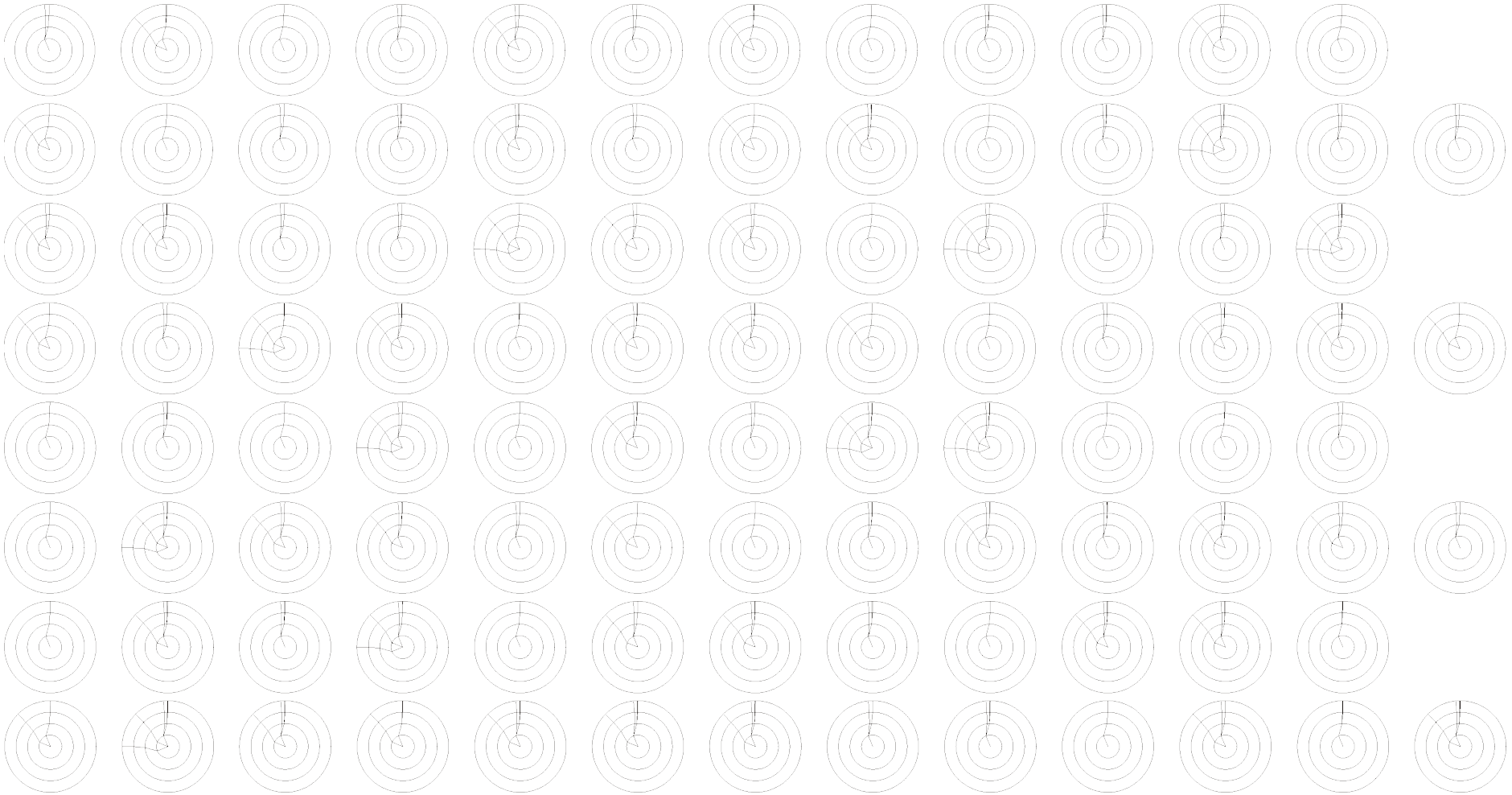


Rysunek 6.14. Przykładowa gramatyka, stanowiąca rozwiązanie zadanego problemu znaleziona w ostatnim pokoleniu w ewolucji nr 3

Jak widać, gramatyki znalezione w wyniku ewolucji z wykorzystaniem operatora delecji posiadają postać charakteryzującą się stosunkowo małą złożonością, co ułatwia ich interpretację.

Na rysunku 6.15 przedstawiona została przykładowa struktura drzew (osobników) w ostatnim pokoleniu w przypadku ewolucji nr 1. Wynika z niego że, struktura osobników powstałych w procesie ewolucji z wykorzystaniem operatora delecji jest znacznie mniej złożona niż w przypadku osobników powstałych w drodze ewolucji z zastosowaniem operatora transwersji (rysunek 6.11).

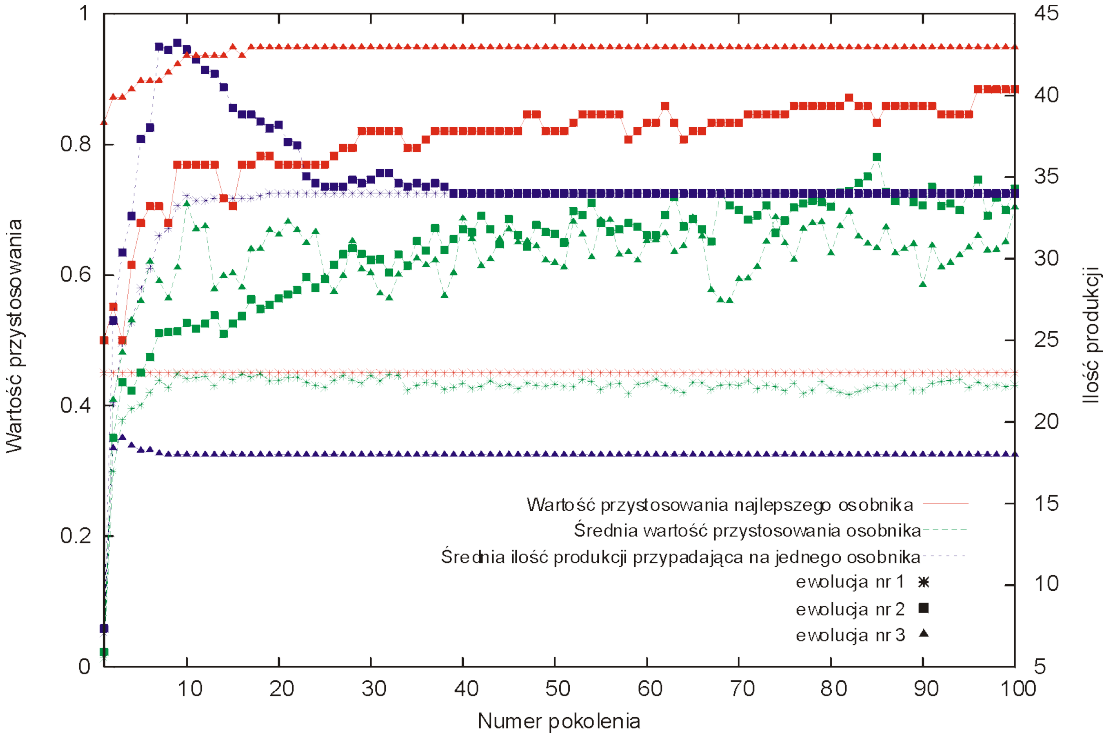
Biorąc pod uwagę korzystny wpływ operatora delecji zarówno na wartość przystosowania osobników, jak i na ich złożoność, operator delecji stosowany będzie jako podstawowy operator mutacji w dalszych badaniach.



Rysunek 6.15. Struktura drzew reprezentujących produkcje gramatyk w ostatnim pokoleniu dla ewolucji z użyciem operatora delecji.

6.2.4. Operator mutacji typu *addycja terminala*

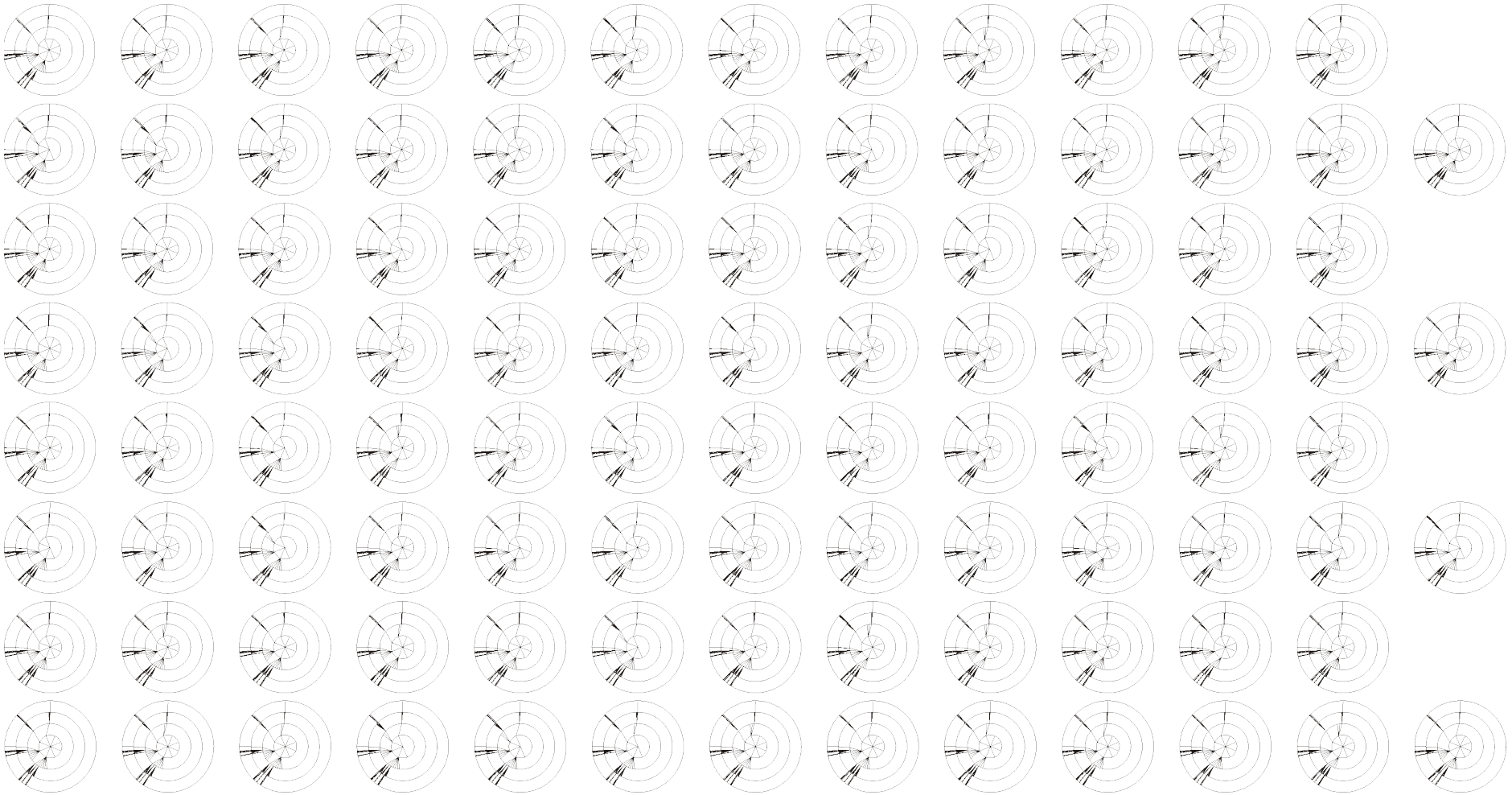
Operacja addycji terminala polega na losowym dodaniu symbolu terminalnego do prawej strony wybranej produkcji. Drzewo powstające w wyniku działania tego operatora ma taką samą wysokość oraz ilość produkcji jak drzewo, z którego powstało. Na wykresie przedstawionym na rysunku 6.16 widać, że w przypadku ewolucji nr 2 i nr 3, działanie tego operatora prowadzi zarówno do wzrostu wartości przystosowania najlepszego osobnika, jak i wzrostu średniego przystosowania osobnika.



Rysunek 6.16. Wyniki ewolucji z zastosowaniem operatora addycji terminala.

Ponieważ działanie operatora addycji terminala nie prowadzi do zwiększenia ilości produkcji dla danego osobnika (widoczny na rysunku 6.16 wzrost średniej ilości produkcji przypadającej na jednego osobnika wynika z procesu selekcji osobników, a nie z działania tego operatora mutacji), a przy tym na ogół powoduje wzrost wartości przystosowania osobników, zastosowanie go jest korzystne z punktu widzenia ewolucji gramatyk. Skutkiem działania operatora addycji terminala jest wzrost ilości symboli terminalnych w prawych stronach produkcji gramatyki, czyli zwiększanie się ilości krawędzi wychodzących z poszczególnych wierzchołków w drzewie reprezentującym produkcje gramatyki.

Rysunek 6.17 przedstawia strukturę drzew reprezentujących produkcje gramatyk dla ostatniego pokolenia w przypadku ewolucji nr 1.



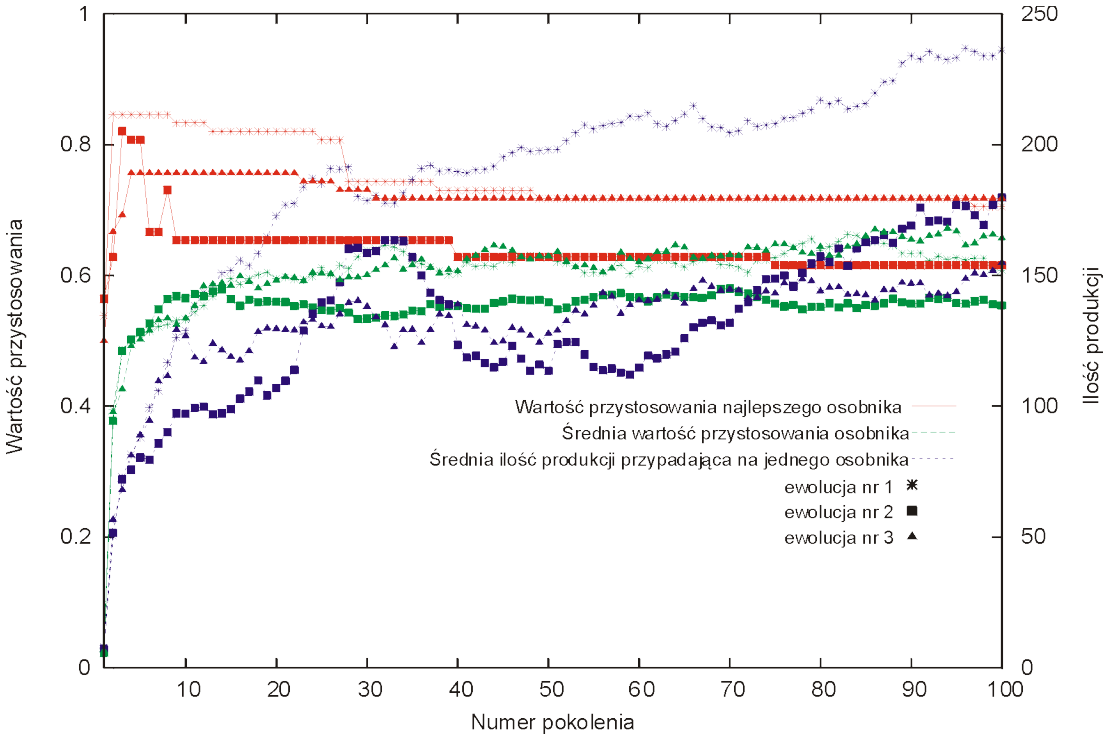
Rysunek 6.17. Struktura drzew reprezentujących produkcje gramatyk w ostatnim pokoleniu dla ewolucji z użyciem operatora addycji terminala

Jak widać na tym rysunku, drzewa powstałe w wyniku ewolucji z zastosowaniem operatora mutacji typu addycja terminala cechują się znacznie bardziej rozbudowaną strukturą niż drzewa wygenerowane w pierwszym pokoleniu (rysunek 1), co wynika właśnie z większej ilości symboli terminalnych znajdujących się w prawych stronach produkcji. Maksymalna ilość symboli terminalnych, jaka może się pojawić w prawej stronie produkcji ograniczana jest przez parametr kontrolujący proces ewolucji **`MaxNoOfT`**. Natomiast drugim sposobem ograniczenia wzrostu ilości symboli terminalnych w prawych stronach produkcji jest zastosowanie w procesie ewolucji łącznie z operatorem addycji terminala operatora delecji, opisanego wcześniej, który usuwa symbole (zarówno terminalne, jak i nieterminalne) z prawych stron produkcji.

6.2.5. Operator mutacji typu *addycja nieterminala*

Działanie operatora addycji nieterminala podobne jest do działania operatora addycji terminala, z tym że do drzewa produkcji wstawiany jest wierzchołek reprezentujący symbol nieterminalny wraz z losowo wygenerowanym poddrzewem, którego jest on korzeniem. Operator addycji nieterminala, jako jedyny z prezentowanych tutaj typów operatorów mutacji, powoduje zwiększenie wysokości drzew reprezentujących produkcje gramatyk. Prowadzi on także do szybkiego wzrostu ilości produkcji osobników, na które działa.

Pokazany na rysunku 6.18 wykres przedstawia wartość przystosowania najlepszego osobnika, średnią wartość przystosowania osobnika oraz średnią ilość produkcji przypadającą na jednego osobnika w poszczególnych pokoleniach w przypadku ewolucji z zastosowaniem operatora addycji nieterminala. Jak widać, działanie tego operatora prowadzi do wzrostu wartości przystosowania najlepszego osobnika tylko w początkowej fazie ewolucji, natomiast w dalszym jej przebiegu powoduje zmniejszania się tej wartości. Wartości te nie spadają jednak poniżej 0.5, co wynika z faktu, że gramatyki z większą ilością produkcji poprawnie rozpoznają przykłady pozytywne, jednak także większość przykładów negatywnych klasyfikują jako należące do języka. Podobnie średnia wartość przystosowania przypadająca na jednego osobnika zwiększa się tylko na początku ewolucji, natomiast w dalszym jej przebiegu pozostaje właściwie stała (z niewielkimi oscylacjami wokół wartości stałej). Natomiast średnia ilość produkcji przypadająca na jednego osobnika rośnie przez cały czas ewolucji (szczególnie wyraźnie jest to widoczne w przypadku ewolucji nr 1), osiągając w ostatnim pokoleniu wartości od około 150 (dla ewolucji nr 3) do ok. 240 (dla ewolucji nr 1).



Rysunek 6.18. Wyniki ewolucji z zastosowaniem operatora addycji nieterminala.

Wzrost wielkości drzew reprezentujących produkcje gramatyk spowodowany działaniem operatora addycji nieterminala ograniczany jest przez dwa parametry: **`MaxTreeDepth`**, ograniczający maksymalną wysokość drzewa, oraz **`MaxNoOfNT`**, który określa maksymalną ilość symboli nieterminalnych, jaka może wystąpić w prawej stronie produkcji.

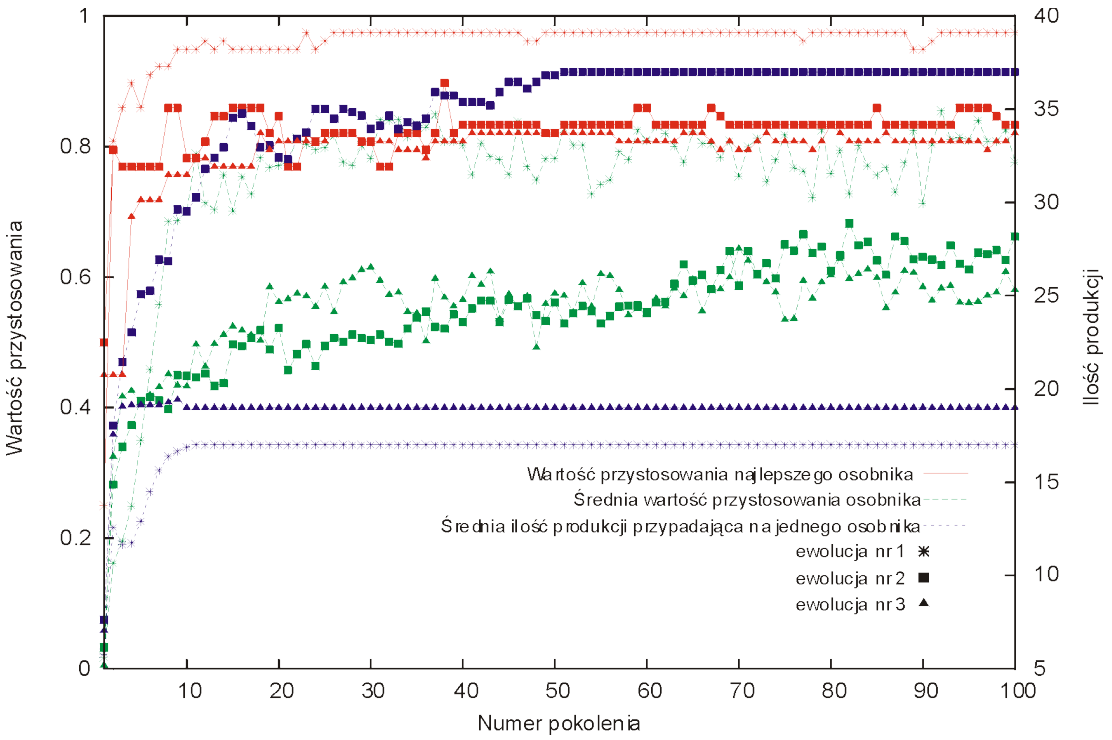
Rysunek 6.19 przedstawia strukturę drzew w ostatnim pokoleniu ewolucji z użyciem operatora addycji nieterminala. Wysokość wszystkich drzew w ostatnim pokoleniu jest równa 10, czyli maksymalna, dopuszczalna przez parametr **`MaxTreeDepth`**. Podobnie jak w przypadku operatora addycji terminala, także w tym przypadku można skompensować efekt wzrostu rozmiaru drzew przez zastosowanie operatora delecji.

Rysunek 6.19. Struktura drzew reprezentujących produkcje gramatyk w ostatnim pokoleniu dla ewolucji z użyciem operatora addycji nieterminala

6.2.6. Operator mutacji typu tranzycja nieterminala

Efektem działania operatora tranzycji nieterminala jest zamiana symbolu nieterminalnego, znajdującego się w lewej stronie pewnej produkcji, na inny, losowo wybrany, symbol nieterminalny. Operator ten nie zmienia wysokości drzewa reprezentującego produkcje gramatyki, ilości produkcji danej gramatyki, ani też ilości symboli w poszczególnych produkcjach.

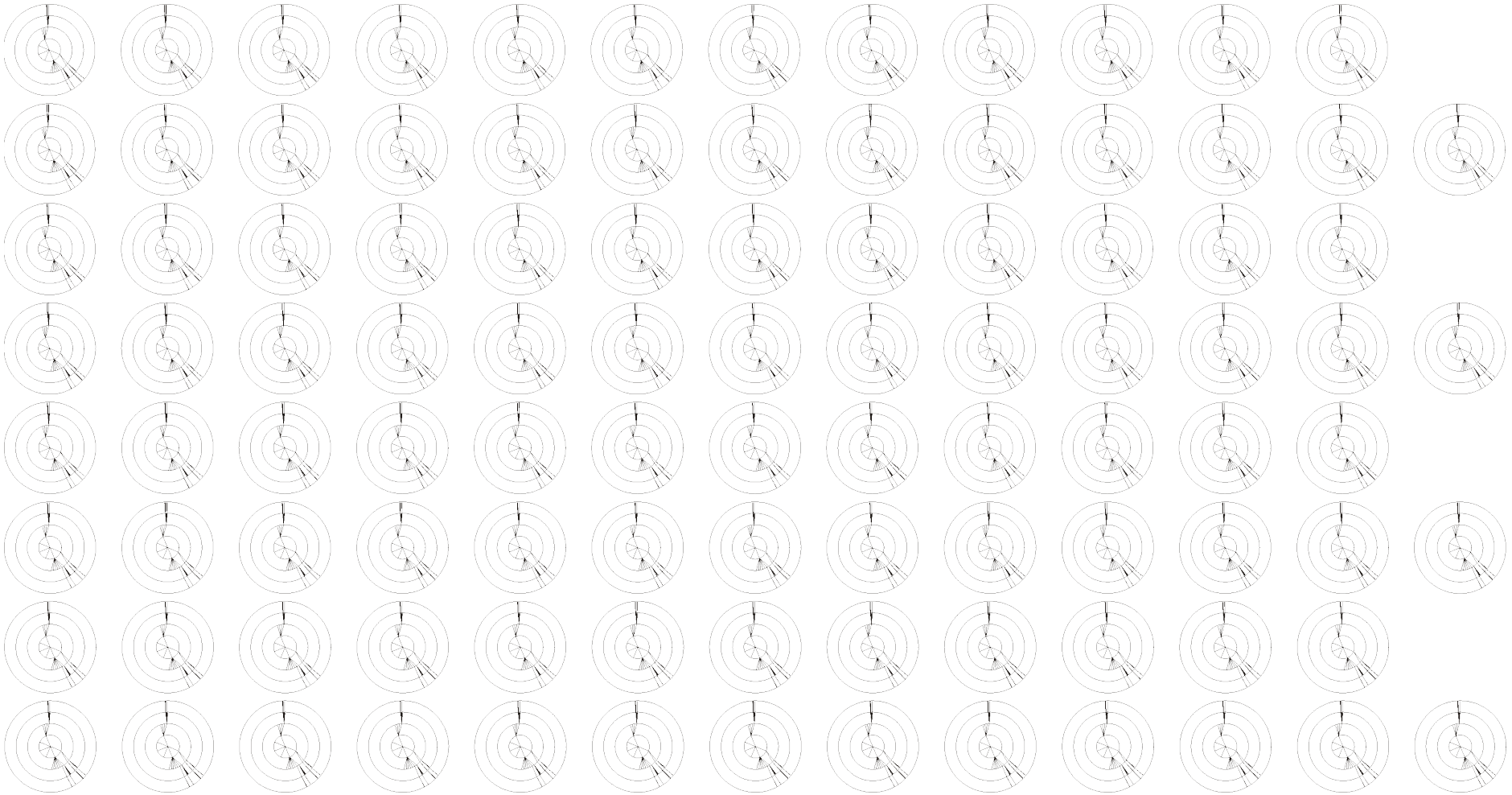
Efekt działania tego operatora przedstawiony został na rysunku 6.20. Widać na nim, że operator tranzycji nieterminala powoduje wzrost zarówno wartości przystosowania najlepszego osobnika, jak i średniej wartości przystosowania osobnika, szczególnie w początkowej fazie ewolucji (do około 30 pokolenia). Natomiast widoczny na wykresie wzrost średniej ilości produkcji przypadających na jedną gramatykę (szczególnie wyraźny w przypadku ewolucji nr 2) jest efektem selekcji osobników, gdyż operator tranzycji nie może zwiększyć ilości produkcji danego osobnika.



Rysunek 6.20. Wyniki ewolucji z zastosowaniem operatora tranzycji nieterminala.

Struktura drzew dla ostatniego pokolenia ewolucji z użyciem operatora tranzycji nieterminala jest uwidoczniona na rysunku 6.21. Ponieważ omawiany operator nie zmienia struktury drzewa, zatem struktura drzew w ostatnim pokoleniu musi być taka sama jak struktura pewnych drzew wchodzących w skład pierwszego pokolenia.

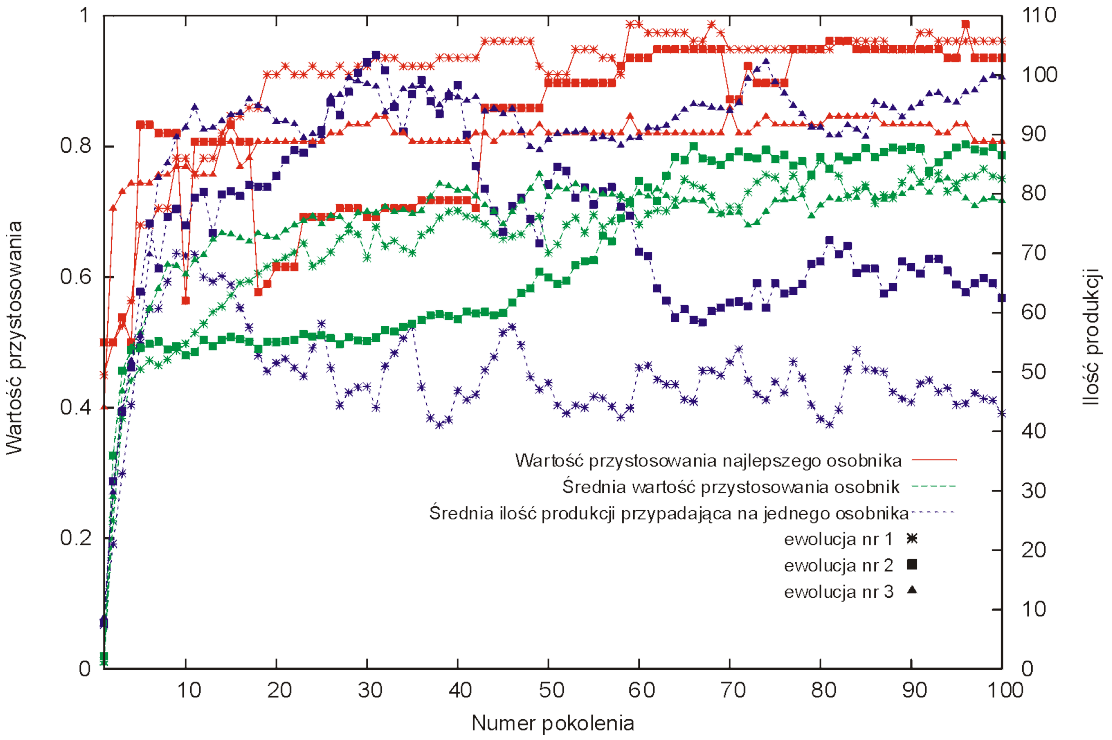
Ze względu na to, że działanie operatora tranzycji nieterminala z jednej strony powoduje wzrost zarówno wartości przystosowania najlepszego osobnika, jak i średniej wartości przystosowania osobnika, a z drugiej nie powoduje zwiększenia ilości produkcji gramatyki ani ilości symboli w produkcjach tej gramatyki, jego zastosowanie jest korzystne z punktu widzenia ewolucji gramatyk.



Rysunek 6.21. Struktura drzew reprezentujących produkcje gramatyk w ostatnim pokoleniu dla ewolucji z użyciem operatora tranzycji nieterminala

6.2.7. Ewolucja z zastosowaniem wszystkich rodzajów operatorów mutacji

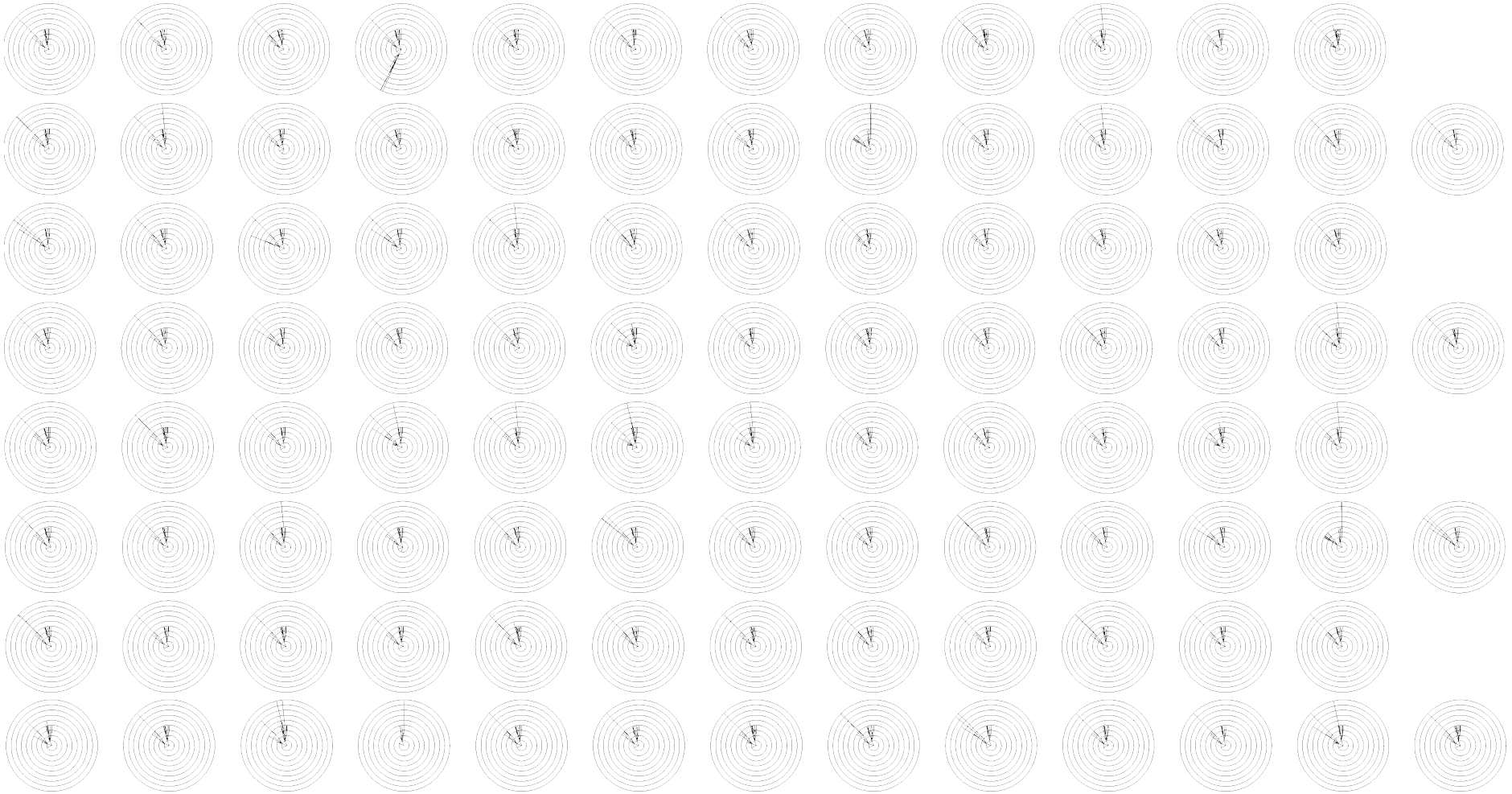
Na rysunku 6.22 przedstawione zostały wyniki ewolucji z zastosowaniem wszystkich omówionych wcześniej rodzajów operatorów mutacji – tj. transwersji, delecji, addycji terminala, addycji nieterminala oraz tranzycji nieterminala. Operatory mutacji były stosowane równoprawnie, tzn. prawdopodobieństwo ich wyboru było jednakowe. Przedstawiony wykres pokazuje wzrost zarówno wartości przystosowania najlepszego osobnika, jak i średniej wartości przystosowania osobnika w trakcie całej ewolucji.



Rysunek 6.22. Wyniki ewolucji z zastosowaniem wszystkich rodzajów operatorów mutacji.

Na wykresie tym widać także znaczny wzrost średniej ilości produkcji przypadających na jedną gramatykę w początkowej fazie ewolucji. Wzrost ten powodowany jest przez dwa czynniki: pierwszym z nich jest proces selekcji, a drugim działanie operatora addycji nieterminala. Jednak porównując wzrost średniej ilości produkcji przypadających na gramatykę dla ewolucji z zastosowaniem tylko operatora addycji nieterminala i ewolucji z zastosowaniem wszystkich operatorów mutacji, widać, że w tym drugim przypadku wzrost ilości produkcji jest znacznie mniejszy. W przypadkach ewolucji nr 1 i nr 2 z zastosowaniem wszystkich rodzajów operatorów mutacji od pewnego momentu ewolucji następuje nawet spadek średniej ilości produkcji przypadających na jedną gramatykę. Jest to wynikiem kompensacji działania operatora addycji nieterminala przez operator delecji.

Na rysunku 6.23 przedstawiona została struktura drzew w ostatnim pokoleniu ewolucji z zastosowaniem wszystkich rodzajów mutacji. Widać, że podobnie jak w przypadku ewolucji z zastosowaniem operatora addycji nieterminala, także w tym przypadku wysokość większości drzew osiągnęła maksymalną wartość dopuszczalną przez parametr **`MaxTreeDepth`** (równy 10). Jednak w populacji pojawiają się również drzewa o mniejszej wysokości niż maksymalna, co nie miało miejsca w przypadku ewolucji z zastosowaniem wyłącznie operatora addycji niterminala. Także struktura drzew (ilość krawędzi dla poszczególnych wierzchołków) jest mniej złożona niż w przypadku ewolucji z zastosowaniem wyłącznie operatora addycji nieterminala, co jest spowodowane kompensującym działaniem operatora delecji.



Rysunek 6.23. Struktura drzew reprezentujących produkcje gramatyk w ostatnim pokoleniu dla ewolucji z użyciem wszystkich rodzajów operatorów mutacji

Wnioski

Mimo że przedstawione tutaj porównanie operatorów mutacji zostało dokonane na przykładzie języka L_1 mającego stosunkowo prostą gramatykę, pozwala ono na sformułowanie pewnych ogólnych wniosków. Wynika to między innymi z faktu, iż poza przypadkami ewolucji nr 2 i nr 3 z zastosowaniem operatora delecji nie zostało znalezione poprawne rozwiązanie zadanego problemu (tzn. gramatyka, dla której wartość funkcji przystosowania byłaby równa 1.0), czyli wpływ selekcji na proces ewolucji był podobny jak w przypadku języków z bardziej złożonymi gramatykami. Porównując procesy ewolucji z zastosowaniem różnych rodzajów operatorów mutacji, można zauważyć, że wykorzystanie wszystkich operatorów mutacji daje właściwie najlepsze (pomijając ewolucje z zastosowaniem wyłącznie operatora delecji) rezultaty, jeśli chodzi o wartość przystosowania najlepszego osobnika oraz pod względem średniej wartości przystosowania przypadającej na osobnika. Ponieważ wyniki uzyskane dla operatora delecji nie są w pełni miarodajne dla języków posiadających bardziej złożone gramatyki[12], można wnioskować, iż łączne zastosowanie wszystkich rodzajów operatorów mutacji daje lepsze rezultaty niż zastosowanie tylko niektórych z nich. Pojawia się jednak pytanie, jakie wagi powinny zostać przypisane poszczególnym rodzajom operatorów mutacji, tzn. jakie powinno być prawdopodobieństwo wyboru danego rodzaju operatora w przypadku wystąpienia mutacji. Przyjęcie jednakowych wag (prawdopodobieństw) nie wydaje się być odpowiednie. Wynika to, po pierwsze, z tego, że skuteczność (tzn. wpływ na wzrost wartości przystosowania osobników) różnych operatorów mutacji jest różna – więc operatory dające potencjalnie możliwość większego wzrostu funkcji przystosowania powinny być stosowane częściej (mieć większe wagi) niż pozostałe. Po drugie zaś z tego, iż w przypadku jednakowego prawdopodobieństwa stosowania operatorów addycji terminala, addycji nieterminala oraz delecji następuje, jak widać na rysunku 6.22, znaczny wzrost średniej ilości produkcji przypadających na jedną gramatykę. Gdyby prawdopodobieństwo zastosowania operatora delecji było większe, wzrost ten mógłby zostać przez niego lepiej skompensowany.

Biorąc pod uwagę otrzymane rezultaty omówione powyżej, ustalono następujące wagi dla poszczególnych rodzajów operatorów mutacji, które będą stosowane w dalszych badaniach:

- waga dla operatora transwersji – $w_1 = 1.0$
- waga dla operatora delecji – $w_2 = wx$[13]
- waga dla operatora addycji terminala – $w_3 = 2.0$

12 Dla bardziej złożonych gramatyk, na ogół nie jest możliwe znalezienie poprawnego rozwiązania na drodze wyłącznie upraszczania gramatyk wygenerowanych w pierwszym pokoleniu.

13 Sposób ustalenia wagi wx zostanie omówiony poniżej.

- waga dla operatora addycji nieterminala – $w_4 = 1.0$
- waga dla operatora tranzycji nieterminala – $w_5 = 3.0$

Przy czym prawdopodobieństwo wyboru i-tego operatora w przypadku mutacji wynosi:

$$p_i = \frac{w_i}{\sum_{i=1}^{5} w_i} \tag{6.1}$$

6.2.8. Sposób ustalenia wagi dla operatora delecji

Waga dla operatora delecji powinna być ustalona w taki sposób, aby operator ten kompensował wzrost rozmiaru drzew spowodowany działaniem operatorów addycji terminala i addycji nieterminala[14]. Podobnie jak w przypadku operacji krzyżowania zachowującego rozmiar, także dla operacji mutacji postulujemy, aby wartość oczekiwana różnicy wielkości drzewa przed mutacją i po niej była równa 0. Uzyskanie tego nie jest możliwe przez ustalenie stałej wartości wagi wx dla operatora delecji, ponieważ wzrost wielkości drzew na skutek działania operatora addycji nieterminala zmienia się w trakcie ewolucji. Dlatego też w dalszych badaniach zastosowany zostanie dynamiczny dobór wagi wx w trakcie procesu ewolucji. Algorytm ustalający wartość wx przedstawiony został na rysunku 6.10.

Na początku procesu ewolucji do zmiennej wx określającej wagę operatora delecji oraz do zmiennej mutationDelta zliczającej przyrost ilości wierzchołków w drzewach produkcji spowodowanych operatorami mutacji przypisywana jest wartość zero. Przed i po każdej operacji mutacji zapamiętywana jest ilość wierzchołków w drzewie produkcji poddawanym tej operacji (odpowiednio w zmiennych np i nk). Następnie po każdej operacji mutacji do zmiennej mutationDelta dodawana jest różnica ilości wierzchołków w drzewie produkcji po mutacji i przed nią. Na koniec ewolucji każdego pokolenia gramatyk sprawdzane jest, czy wartość mutationDelta jest większa od zera - jeśli tak, to waga wx operatora delecji jest podwajana, jeśli natomiast wartość mutationDelta jest mniejsza od zera, to waga wx jest zmniejszana dwukrotnie. Dodatkowo wprowadzone zostały ograniczenia górne i dolne wartości wx, tak że waga wx zwiększana jest tylko do momentu osiągnięcia wartości 1000 razy wartość sumy wag pozostałych operatorów mutacji (zmienna wo) i zmniejszana tylko do czasu osiągnięcia wartość 1/1000 sumy wag pozostałych operatorów mutacji.

14 Pozostałe rodzaje operatorów mutacji nie powodują zwiększenia rozmiaru drzew reprezentujących produkcje gramatyk.

```
wx = 1.0                -- na początku przypisz do wx
                        -- wartość 1.0
wo = 1 + 2 + 1 + 3      -- suma wag pozostałych typów mutacji
mutationDelta = 0       -- przyrost ilości wierzchołków
                        -- spowodowany mutacją

-- dla każdej mutacji
np = ilosc_wierzcholkow_w_drzewie_przed_mutacja
wykonaj_mutacje()
nk = ilosc_wierzcholkow_w_drzewie_po_mutacji
mutationDelta = mutationDelta + (nk - np)

-- na koniec ewolucji każdego pokolenia aktualizuj wx:
if(mutationDelta > 0) and (wx < 1000*wo) then
   wx = 2.0 * wx        -- zwiększ udział operatora delecji
elseif (mutationDelta < 0) and (wx > 1/(1000*wo)) then
   wx = wx / 2.0        -- zmniejsz udział operatora delecji
end

-- przejdź do ewolucji następnego pokolenia gramatyk
```

Rysunek 6.24. Algorytm obliczania wagi wx dla operatora delecji

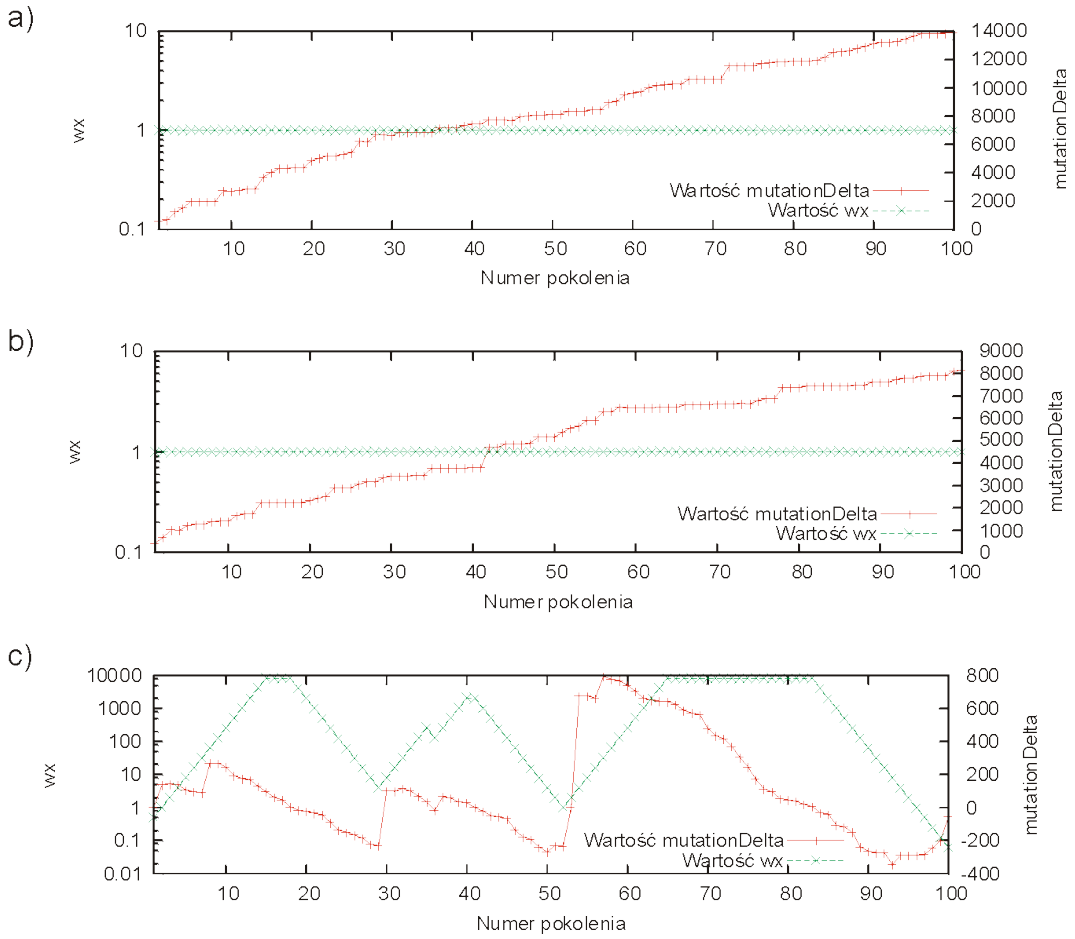
Na wykresach znajdujących się na rysunkach 6.25 i 6.26 przedstawione zostało porównanie ewolucji dla trzech wariantów wag operatora mutacji:

a) $w_1 = w_3 = w_4 = w_5 = 1$, $w_2 = wx = 1$ – wszystkie wagi operatorów mutacji stałe i równe 1;

b) $w_1 = 1$, $w_2 = wx = 1$, $w_3 = 2$, $w_4 = 1$, $w_5 = 3$ – wartości wag operatorów mutacji takie jak postulowane powyżej, przy czym waga wx operatora delecji stała i równa 1;

c) $w_1 = 1$, $w_3 = 2$, $w_4 = 1$, $w_5 = 3$, $w_2 = wx$ – dynamiczna – wagi operatorów mutacji takie jak postulowane powyżej, przy czym waga wx operatora delecji ustalana zgodnie z algorytmem przedstawionym na rysunku 6.24

Pozostałe parametry ewolucji były takie jak w tabeli 6.4, z tym że wartość parametru **`CrossoverProb`** wynosiła 0.8, natomiast wartość parametru **`CrossoverOnTerminalPointProb`** wynosiła 0.1. Gramatyki generowane były dla języka L_1 na podstawie przykładów przedstawionych w tabeli 6.1.

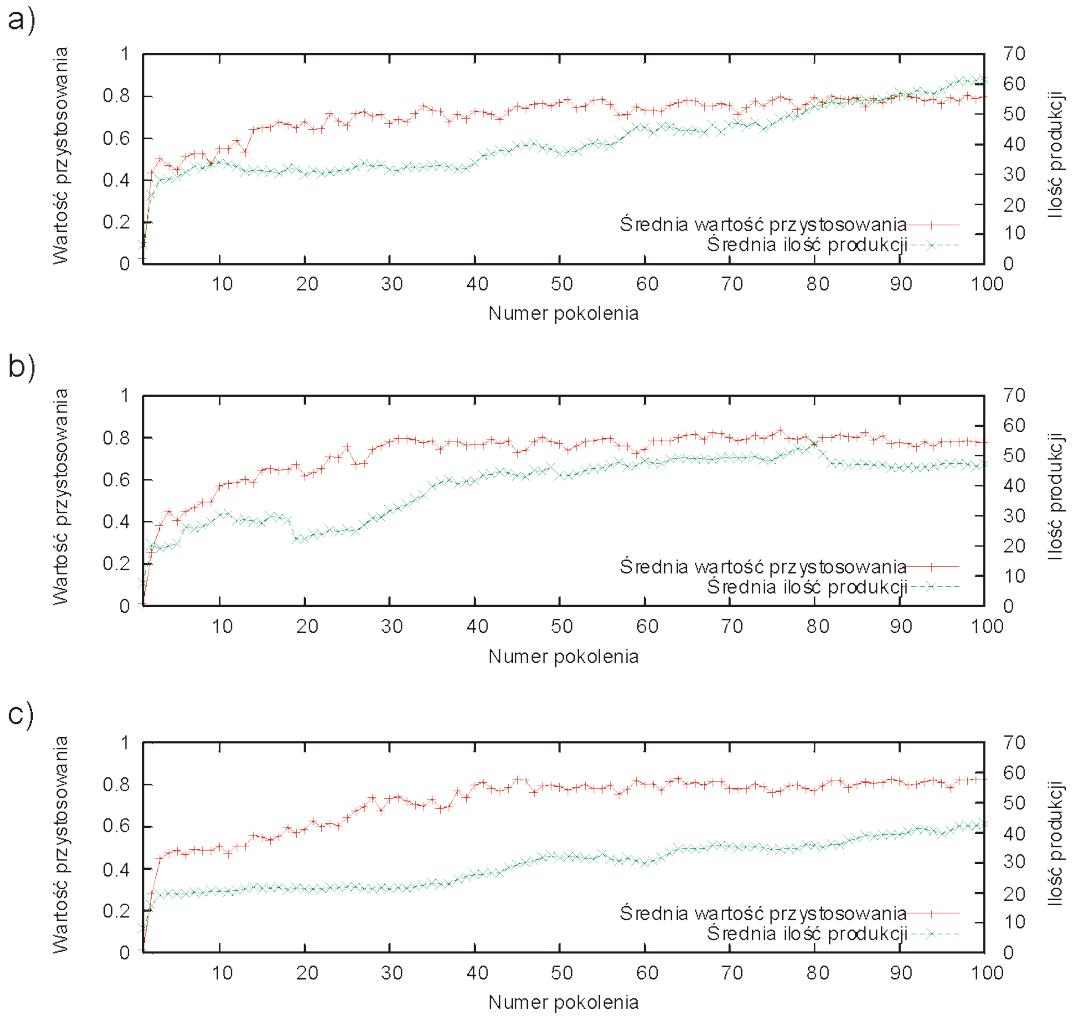


Rysunek 6.25. Porównanie przyrostów ilości wierzchołków w drzewach produkcji spowodowanych operatorami mutacji (zmienna mutationDelta) oraz wag wx operatora delecji dla trzech wariantów ustawienia wag operatorów mutacji a) wariant dla wag stałych i równych 1: $w_1 = wx = w_3 = w_4 = w_5 = 1$, b) wariant dla stałych wag wartości: $w_1 = 1$, $wx = 1$, $w_3 = 2$, $w_4 = 1$, $w_5 = 3$, c) wariant dla dynamicznej wagi wx wartości $w_1 = 1$, $w_3 = 2$, $w_4 = 1$, $w_5 = 3$

Jak wynika to z wykresów na rysunku 6.25, przypadek stałych i jednakowych wag dla wszystkich rodzajów operatorów mutacji powoduje najszybszy wzrost wielkości drzew reprezentujących produkcje gramatyki – w pokoleniu numer 100 przyrost ilości wierzchołków wszystkich drzew spowodowany działaniem operatorów mutacji w tym wariancie wynosi około 14000. Ustalenie stałych wag dla operatorów mutacji o niejednakowych wartościach postulowanych powyżej powoduje także stały wzrost wielkości drzew reprezentujących produkcje gramatyk, ale w tym przypadku jest on znacznie mniejszy – w pokoleniu numer 100 łączny przyrost ilości wierzchołków w drzewach produkcji spowodowany działaniem operatorów mutacji wynosi około 8000. Natomiast zastosowanie dynamicznie obliczanej wagi wx dla operatora delecji zgodnie z algorytmem przedstawionym na rysunku 6.24 powoduje oscylacyjne zmiany przyrostu ilości wierzchołków spowodowane działaniem operatorów mutacji. W przedstawionym przypadku ewolucji zmiany te były w zakresie od –400 do +800, czyli największy przyrost ilości wierzchołków był dziesięciokrotnie mniejszy niż przyrost ilości

wierzchołków w setnym pokoleniu w wypadku stałej wagi wx operatora delecji (przypadek b).

Na wykresach znajdujących się na rysunku 6.26 przedstawione zostało porównanie średniej wartości przystosowania oraz średniej ilości produkcji gramatyk w danym pokoleniu.



Rysunek 6.26. Porównanie średniej wartości przystosowania i średniej ilości produkcji osobników wchodzących w skład danego pokolenia dla trzech wariantów ustawienia wag operatorów mutacji a) wariant dla wag stałych i równych 1: $w_1 = wx = w_3 = w_4 = w_5 = 1$, b) wariant dla stałych wag wartości: $w_1 = 1$, $wx = 1$, $w_3 = 2$, $w_4 = 1$, $w_5 = 3$, c) wariant dla dynamicznej wagi wx, wartości $w_1 = 1$, $w_3 = 2$, $w_4 = 1$, $w_5 = 3$

Jak widać na wykresach, średnia wartość przystosowania jest porównywalna we wszystkich trzech wariantach ustawień wag operatorów mutacji, natomiast średnia ilość produkcji jest zdecydowanie mniejsza w przypadku dynamicznie ustalanej wagi wx niż w pozostałych przypadkach.

Podsumowując – zastosowanie dynamicznie ustalanej wagi wx dla operatora delecji w taki sposób, aby wartość oczekiwana różnicy wielkości drzewa przed mutacją

i po niej była równa 0, jest korzystne z punktu widzenia ewolucji, ponieważ tworzy gramatyki o mniejszej ilości produkcji posiadające jednak taką samą wartość przystosowania[15] jak gramatyki o większej ilości produkcji tworzone w procesie ewolucji ze stałą wartością wagi wx.

6.3. Wpływ rodzaju parsera na przebieg ewolucji gramatyk

Jednym z najważniejszych elementów systemu ewolucyjnego wnioskowania gramatyk na podstawie przykładów języka jest parser. Jego zadaniem jest weryfikacja, w jakim stopniu wygenerowana gramatyka rozpoznaje zadane przykłady języka. W systemie wnioskowania gramatyk przedstawionym w niniejszej rozprawie (patrz rozdział 5) stosowane były alternatywnie trzy parsery[16]:

- Bison – odpowiednik generatora parserów Yacc powstały w ramach projektu GNU; wymaga jednoznacznej gramatyki bezkontekstowej klasy LALR[17];
- Elkhound – generator parserów klasy GLR; pozwala na generowanie parserów dla dowolnych gramatyk bezkontekstowych;
- CYKP – parser akceptujący dowolne gramatyki bezkontekstowe będący implementacją algorytmu Cocke-Younger-Kasami.

6.3.1. Wnioskowanie gramatyki pozwalającej na wykrywanie przewężeń tętnic wieńcowych

Przedstawione tutaj wyniki pokazują wpływ rodzaju zastosowanego parsera na proces wnioskowania gramatyk opisujących przewężenia tętnic wieńcowych [46], [50].

Parametry kontrolujące przebieg procesu ewolucji przedstawione zostały w tabeli 6.5.

Dla każdego z parserów (Bison, Elkhound oraz CYKP) przedstawiono wyniki trzech przebiegów ewolucji dla różnej ilości przykładów uczących języka:

- jeden przykład pozytywny i jeden przykład negatywny języka (zbiór S1 Dodatek C),
- 20 przykładów pozytywnych i 20 przykładów negatywnych języka (zbiór S2 Dodatek C),

15 Mowa jest tutaj o wartościach średnich.

16 Programy Bison i Elkhound są generatorami parserów. Określenia 'parser Bison' i 'parser Elkhound' oznaczają tutaj odpowiednio: parser wygenerowany przy użyciu programu Bison oraz parser wygenerowany przy użyciu programu Elkhound.

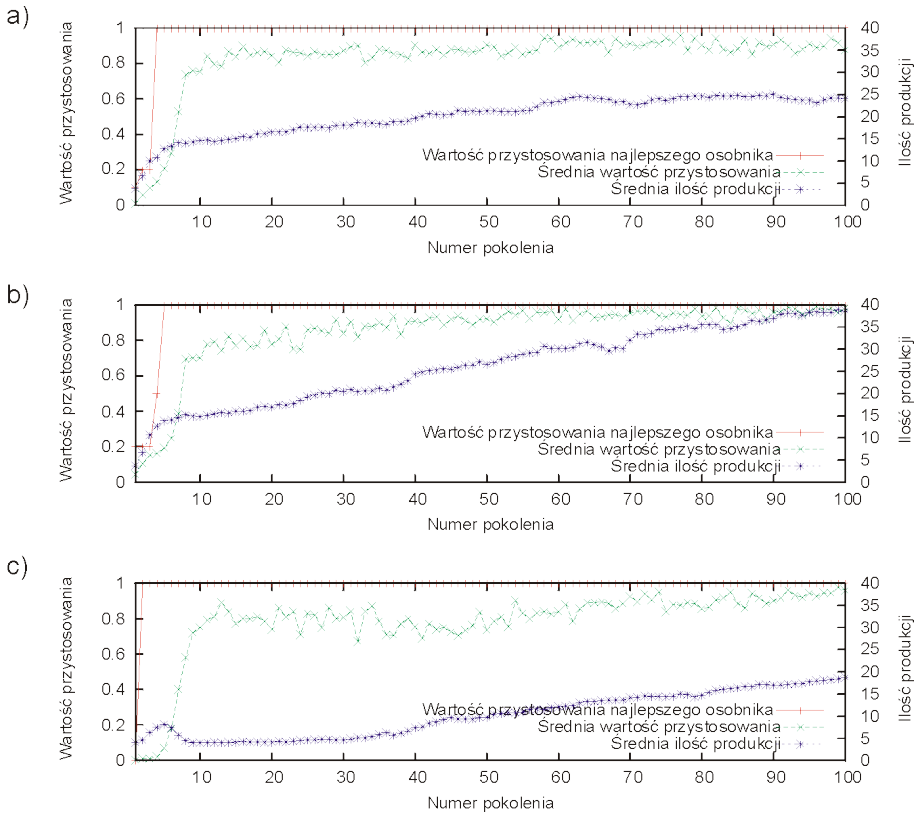
17 Generator parserów Bison pozwala także na generowanie parserów klasy GLR, ale ta funkcjonalność nie była tu wykorzystywana.

- 50 przykładów pozytywnych i 50 przykładów negatywnych języka (zbiór S3 Dodatek C).

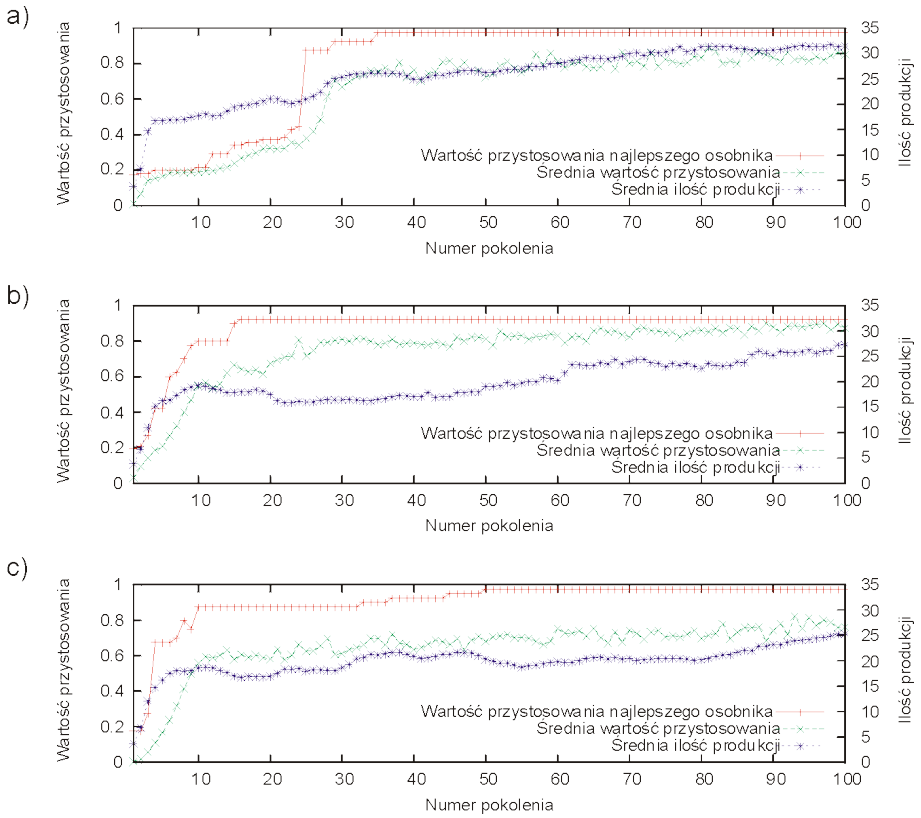
Tabela 6.5. Parametry kontrolujące proces ewolucji gramatyk

Nazwa parametru	*Wartość parametru*
Terminals	n v h
NTerminals	S N O P Q R
TotalNoOfGenerations	100
GenerationSize	100
MaxNoOfNT	4
MaxNoOfT	4
NTRedefinitionProb	0.8
InitialMaxTreeDepth	4
MaxTreeDepth	6
CrossoverProb	0.8
CrossoverOnTerminalPointProb	0.1
NoOfMutationTry	2
MutationProbabilityOnOneTry	0.05

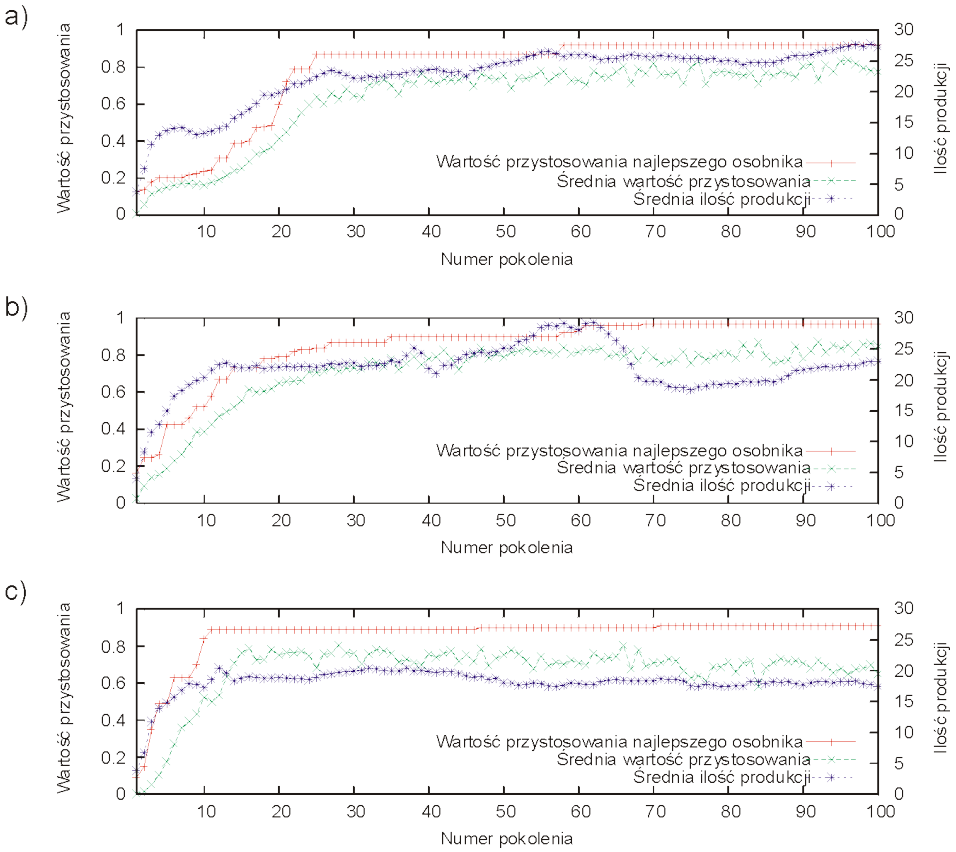
Wykresy znajdujące się na rysunkach 6.27, 6.28 oraz 6.29 przedstawiają wartość przystosowania najlepszego osobnika, średnią wartość przystosowania osobników oraz średnią ilość produkcji w poszczególnych pokoleniach dla ewolucyjnego wnioskowania gramatyk opisujących przewężenia tętnic wieńcowych. Pokazują one, że średnia ilość produkcji dla deterministycznych gramatyk klasy LALR (parser Bison) jest na ogół większa niż w przypadku gramatyk niedeterministycznych (parsery Elkhound i CYKP) przy porównywalnych wartościach przystosowania gramatyk. Widać także, iż przy zastosowaniu parserów Elkhound i CYKP znacznie szybciej, niż w przypadku użycia parsera Bison, znajdowana jest gramatyka akceptująca wszystkie przykłady pozytywne (o wartości przystosowania większej lub równej 0.5).



Rysunek 6.27. Wyniki ewolucyjnego wnioskowania gramatyk opisujących przewężenia tętnic wieńcowych na podstawie jednego przykładu pozytywnego i jednego przykładu negatywnego języka a) parser Bison, b) parser Elkhound, c) parser CYKP



Rysunek 6.28. Wyniki ewolucyjnego wnioskowania gramatyk opisujących przewężenia tętnic wieńcowych na podstawie 20 przykładów pozytywnych i 20 przykładów negatywnych języka a) parser Bison, b) parser Elkhound, c) parser CYKP



Rysunek 6.29. Wyniki ewolucyjnego wnioskowania gramatyk opisujących przewężenia tętnic wieńcowych na podstawie 50 przykładów pozytywnych i 50 przykładów negatywnych języka a) parser Bison, b) parser Elkhound, c) parser CYKP

6.3.2. Wnioskowanie gramatyk opisujących ruchy robota

Poniżej przedstawione zostanie porównanie rezultatów, zamieszczonych w pracy [43][18], dotyczących wnioskowania gramatyki bezkontekstowej opisującej ruchy robota, z rezultatami uzyskanymi przy użyciu systemu wnioskowania gramatyk, który został zaprezentowany w niniejszej rozprawie. Porównane zostaną także wyniki ewolucji gramatyk z wykorzystaniem parsera klasy LALR (Bison) oraz parsera CYKP.

Gramatyka generowana była na podstawie zbioru przykładów języka pochodzących z pracy [43] przedstawionych w tabeli 6.6.

18 System wnioskowania gramatyk prezentowany w cytowanej pracy, podobnie jak system zaprezentowany w niniejszej rozprawie, oparty jest na bazie programowania genetycznego.

Tabela 6.6. Przykłady języka opisującego ruchy robota

Przykłady pozytywne	*Przykłady negatywne*

`begin left right end,`
`begin left right left left end,`
`begin left left right right right left left left end`
	`left right end,`
	`begin begin left right end,`
	`begin left left right`

Zbiór symboli terminalnych składał się z elementów: {c, b, e} zdefiniowanych za pomocą wyrażeń regularnych w następujący sposób:

```
c: left | right
b: begin
e: end
```

Parametry kontrolujące proces ewolucji, odpowiadające tym, które zostały zastosowane w pracy [43], przedstawione zostały w tabeli 6.7.

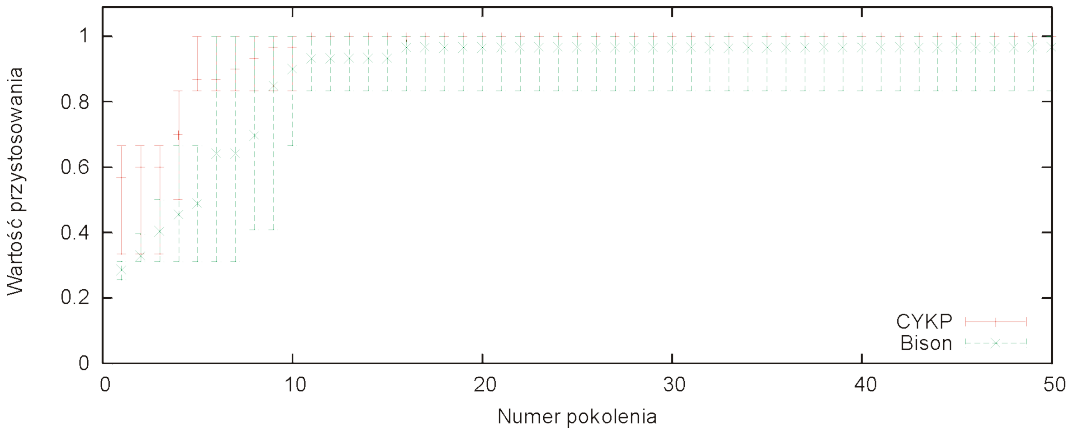
Tabela 6.7. Parametry kontrolujące proces ewolucji gramatyk opisujących ruchy robota

Nazwa parametru	*Wartość parametru*
Terminals	c b e
NTerminals	S E T F R
TotalNoOfGenerations	50
GenerationSize	300
MaxNoOfNT	4
MaxNoOfT	4
MaxNoOfSymbols	4
NTRedefinitionProb	0.8
InitialMaxTreeDepth	5
MaxTreeDepth	6
CrossoverProb	0.6
CrossoverOnTerminalPointProb	0.1
NoOfMutationTry	2
MutationProbabilityOnOneTry	0.05

Wykonano po pięć przebiegów ewolucji dla parsera Bison i CYKP. Wykres 6.30 przedstawia średnią wartość przystosowania najlepszego osobnika z pięciu przebiegów ewolucji w poszczególnych pokoleniach oraz zakres od wartości najmniejszej do największej dla tych przebiegów.

W przypadku zastosowania w procesie ewolucji parsera CYKP już w piątym pokoleniu, w jednym z przebiegów ewolucji, została znaleziona gramatyka z funkcją przystosowania równą 1.0, natomiast od 11 pokolenia we wszystkich pięciu przebiegach wartość najlepszej gramatyki wynosiła 1.0.

Natomiast w przypadku zastosowania parsera Bison w szóstym pokoleniu jednego z przebiegów ewolucji została znaleziona poprawna gramatyka (z funkcją przystosowania równą 1.0), jednak w jednym z przebiegów ewolucji, aż do pięćdziesiątego pokolenia nie została znaleziona gramatyka z funkcją przystosowania równą 1.0.



Rysunek 6.30. Wartość przystosowania najlepszej gramatyki w poszczególnych pokoleniach dla ewolucyjnego wnioskowania gramatyk opisujących ruchy robota z zastosowaniem parserów CYKP i Bison. (Zaznaczone punkty prezentują wartość średnią przystosowania najlepszego osobnika z pięciu przebiegów ewolucji, natomiast zaznaczone przedziały przedstawiają zakres od wartości najmniejszej do największej w poszczególnych przebiegach ewolucji)

Dla porównania, w podejściu prezentowanym w pracy [43], poprawna gramatyka znaleziona została w pokoleniu dziesiątym.

Tabela 6.8 przedstawia numery pokoleń, w których po raz pierwszy została znaleziona poprawna gramatyka – na zielono zostały zaznaczone przypadki, w których znalezienie gramatyki nastąpiło wcześniej niż w przypadku pracy [43].

Tabela 6.8. Pokolenia, w których została po raz pierwszy znaleziona poprawna gramatyka w poszczególnych przebiegach ewolucji

Przebieg ewolucji	***Nr pokolenia, w którym została znaleziona poprawna gramatyka***	
	Parser Bison	***Parser CYKP***
1	**9**	**7**
2	**-**	**5**
3	**16**	**11**
4	**9**	**9**
5	**6**	**8**

Jak widać, przy użyciu parsera CYKP otrzymano znacznie lepsze rezultaty niż w przypadku parsera Bison, jednak w obydwu wariantach otrzymane wyniki są lepsze niż przedstawione w pracy [43].

6.3.3. Wnioskowanie gramatyk dla wyrażeń przypisania z operacjami arytmetycznymi z prawej strony

Poniżej przedstawione zostało porównanie efektywności dla celów wnioskowania gramatyk opisujących wyrażenia przypisania z prostymi operacjami arytmetycznymi (z prawej strony wyrażenia) prezentowanego w niniejszej pracy systemu ewolucyjnego wnioskowania gramatyk z systemem przedstawionym w pracy [42]. Gramatyka generowana była na podstawie przykładów języka pochodzących z pracy [42], które przedstawiono w tabeli 6.9.

Tabela 6.9. Przykłady języka opisującego wyrażenia przypisania z operacjami arytmetycznymi z prawej strony przypisania

Przykłady pozytywne	***Przykłady negatywne***
`a := 9 + 2,`	`22 := d,`
`a := 10 + 2 + 12,`	`d := := 32,`
`abc := 22,`	`:= 2,`
`j := 0`	`a :=,`
	`+ 6`

Zbiór symboli terminalnych składał się z elementów: {i, a, p, d} zdefiniowanych za pomocą wyrażeń regularnych następująco:

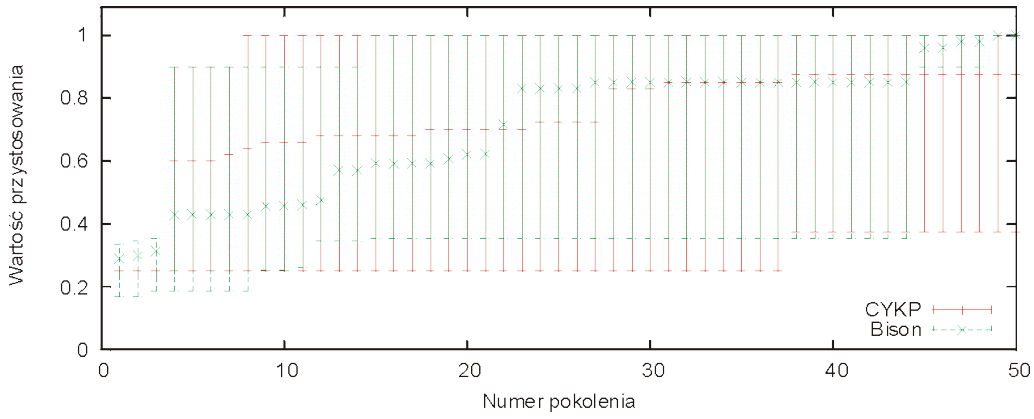
```
i: [0-9]+
a: \:=
p: \+
d: [a-z]+
```

Parametry kontrolujące proces ewolucji, odpowiadające zastosowanym w pracy [42], przedstawia tabela 6.10. Podobnie jak w przypadku wnioskowania gramatyk opisujących ruchy robota (patrz rozdział 6.3.2) wykonano po pięć przebiegów ewolucji dla parserów Bison i CYKP. Wykres na rysunku 6.31 przedstawia średnią wartość przystosowania najlepszego osobnika z pięciu przebiegów ewolucji w poszczególnych pokoleniach oraz zakres wartości od najmniejszej do największej dla tych przebiegów. Tabela 6.11 zawiera numery pokoleń dla poszczególnych przebiegów ewolucji, w których została znaleziona poprawna gramatyka (o wartości przystosowania równej 1.0). Na zielono zaznaczono w niej przypadki, w których poprawna gramatyka została

znaleziona we wcześniejszym pokoleniu niż miało to miejsce w wynikach zaprezentowanych w pracy [42]. Dla porównania, w pracy [42] bez zastosowania w procesie generowania pierwszego pokolenia drzew wyprowadzenia dla poszczególnych przykładów pozytywnych nie udało się znaleźć poprawnej gramatyki w 50 pokoleniach, natomiast zastosowanie drzew wyprowadzenia pozwoliło na znalezienie poprawnej gramatyki w dwudziestym pierwszym pokoleniu.

Tabela 6.10. Parametry kontrolujące proces ewolucji gramatyk dla wyrażeń przypisania z operacjami arytmetycznymi z prawej strony

Nazwa parametru	***Wartość parametru***
Terminals	i a p d
NTerminals	S M N O P Q R
TotalNoOfGenerations	50
GenerationSize	300
MaxNoOfNT	5
MaxNoOfT	5
MaxNoOfSymbols	5
NTRedefinitionProb	0.8
InitialMaxTreeDepth	4
MaxTreeDepth	5
CrossoverProb	0.5
CrossoverOnTerminalPointProb	0.1
NoOfMutationTry	2
MutationProbabilityOnOneTry	0.05



Rysunek 6.31. Wartość przystosowania najlepszej gramatyki w poszczególnych pokoleniach dla ewolucyjnego wnioskowania gramatyk opisujących wyrażenia przypisania z zastosowaniem parserów CYKP i Bison. (Zaznaczone punkty prezentują wartość średnią przystosowania najlepszego osobnika z pięciu przebiegów ewolucji, natomiast zaznaczone przedziały przedstawiają zakres od wartości najmniejszej do największej w poszczególnych przebiegach ewolucji)

Tabela 6.11. Pokolenia, w których po raz pierwszy została znaleziona poprawna gramatyka w poszczególnych przebiegach ewolucji

Przebieg ewolucji	Nr pokolenia w którym została znaleziona poprawna gramatyka	
	Parser Bison	Parser CYKP
1	15	12
2	23	31
3	27	8
4	47	18
5	49	-

Także w tym przypadku rezultaty otrzymane dla parsera CYKP są lepsze niż dla parsera Bison. Warto zauważyć także, że wyniki uzyskane przy użyciu w procesie ewolucyjnego wnioskowania gramatyk parsera CYKP są lepsze niż prezentowane w pracy [42], mimo iż nie zastosowano tutaj specjalnych operatorów heurystycznych oraz tworzenia populacji pierwszego pokolenia na bazie drzew wyprowadzenia dla poszczególnych przykładów pozytywnych.

6.3.4. Wnioski

Przedstawione powyżej wyniki sugerują, iż użycie parsera akceptującego dowolne gramatyki bezkontekstowe, a nie tylko gramatyki deterministyczne klasy LALR, prowadzi do szybszego znalezienia poprawnej gramatyki. Także średnia ilość produkcji dla ogólnych gramatyk bezkontekstowych jest mniejsza niż w przypadku gramatyk klasy LALR. Rezultaty te są zgodne z oczekiwaniami, ponieważ gramatyki deterministyczne klasy LALR stanowią podzbiór właściwy zbioru wszystkich gramatyk bezkontekstowych. Wyniki te pozwalają wysunąć wniosek, że korzystniejsze z punktu widzenia ewolucji gramatyk jest użycie parsera akceptującego szerszą klasę gramatyk bezkontekstowych (Elkhound lub CYKP). Ponieważ jednak złożoność czasowa analizy syntaktycznej dla parserów Elkhound oraz CYKP wynosi $O(n^3)$, natomiast – dla parsera klasy LALR (Bison) – O(n), w pewnych przypadkach korzystniejsze jest użycie parsera Bison w procesie ewolucji gramatyk, ponieważ prowadzi do znalezienia rozwiązań w postaci deterministycznych gramatyk klasy LALR. Jednym z takich przypadków, dla których złożoność czasowa może mieć decydujące znaczenie, może być na przykład zastosowanie procesu ewolucyjnego generowania gramatyk jako części systemu semantycznego wyszukiwania danych w multimedialnych medycznych bazach danych, którego koncepcja została zaprezentowana w pracy [48].

6.4. Porównanie metod generowania pierwszego pokolenia

Jak zostało to zaznaczone w opisie modelu ewolucji, testowane były dwa sposoby tworzenia pierwszego pokolenia gramatyk: tworzenie w sposób losowy (metodą *ramped half and half)* oraz na bazie przykładów pozytywnych języka. Przedstawione w niniejszym rozdziale wyniki prezentują wpływ sposobu generowania pierwszego pokolenia gramatyk na dalszy przebieg ewolucji w przypadku generowania gramatyk opisujących przewężenia tętnic wieńcowych.

Algorytm generowania pierwszego pokolenia na podstawie przykładów jest następujący: wybierany jest pierwszy przykład pozytywny, na podstawie którego tworzone jest drzewo opisujące produkcje gramatyki (osobnik), a następnie drugi przykład pozytywny i na jego podstawie tworzone jest kolejny osobnik – drzewo opisujące produkcje gramatyki. W sytuacji, gdy nie ma już więcej przykładów pozytywnych języka, a nie zostały wygenerowane jeszcze wszystkie osobniki z pierwszego pokolenia, proces generowania kontynuowany jest ponownie od pierwszego pozytywnego przykładu. Algorytm tworzenia drzewa opisującego produkcje gramatyki na podstawie pozytywnego przykładu języka został przedstawiony w dodatku D.

Zaprezentowany algorytm gwarantuje, iż każda wygenerowana przy jego użyciu gramatyka będzie poprawnie rozpoznawała przynajmniej jeden przykład pozytywny języka, jednak nie zapewnia tego, że gramatyka ta będzie klasy LALR(1), co w przypadku użycia w procesie ewolucji parsera Bison może powodować, iż wartość przystosowania tak wygenerowanej gramatyki będzie równa 0.

Zaprezentowane poniżej wyniki dotyczą generowania gramatyk bezkontekstowych opisujących przewężenia tętnic wieńcowych; parametry sterujące procesem ewolucji przedstawia tabela 6.12, natomiast zbiór przykładów języka S4 składający się z 50 przykładów pozytywnych i 41 przykładów negatywnych znajduje się w dodatku C.

Przeprowadzono po pięć przebiegów ewolucji z zastosowaniem parsera Bison i CYKP. Uzyskane wyniki zostały zaprezentowane na rysunku 6.32. Wykresy a) oraz c) przedstawiają wartość przystosowania najlepszego osobnika odpowiednio dla parsera Bison i CYKP (na wykresie pokazano wartości średnie z pięciu przebiegów ewolucji oraz odchylenie standardowe dla tych wartości). Wykresy b) i d) prezentują wartości średnie z całej populacji w pięciu przebiegach ewolucji wraz z zaznaczonym odchyleniem standardowym tych wartości. Jak widać na wykresach, w przypadku użycia generatora parserów Bison zarówno wartość przystosowania najlepszego osobnika, jak i średnia wartość przystosowania w populacji jest podczas ewolucji mniejsza w przypadku generowania pierwszego pokolenia gramatyk z przykładów języka niż w przypadku losowego generowania tych osobników. Jest to z jednej strony spowodowane tym, iż w wypadku generowania gramatyk na podstawie przykładów języka

zróżnicowanie populacji pierwszego pokolenia jest mniejsze niż w przypadku generowania ich metodą *ramped half and half*, zaś z drugiej strony faktem, iż część wygenerowanych gramatyk nie jest klasy LALR(1), co powoduje, że nie jest możliwe utworzenie na ich podstawie poprawnych parserów za pomocą generatora parserów Bison, co prowadzi do uzyskania przez nie wartości przystosowania równej 0.

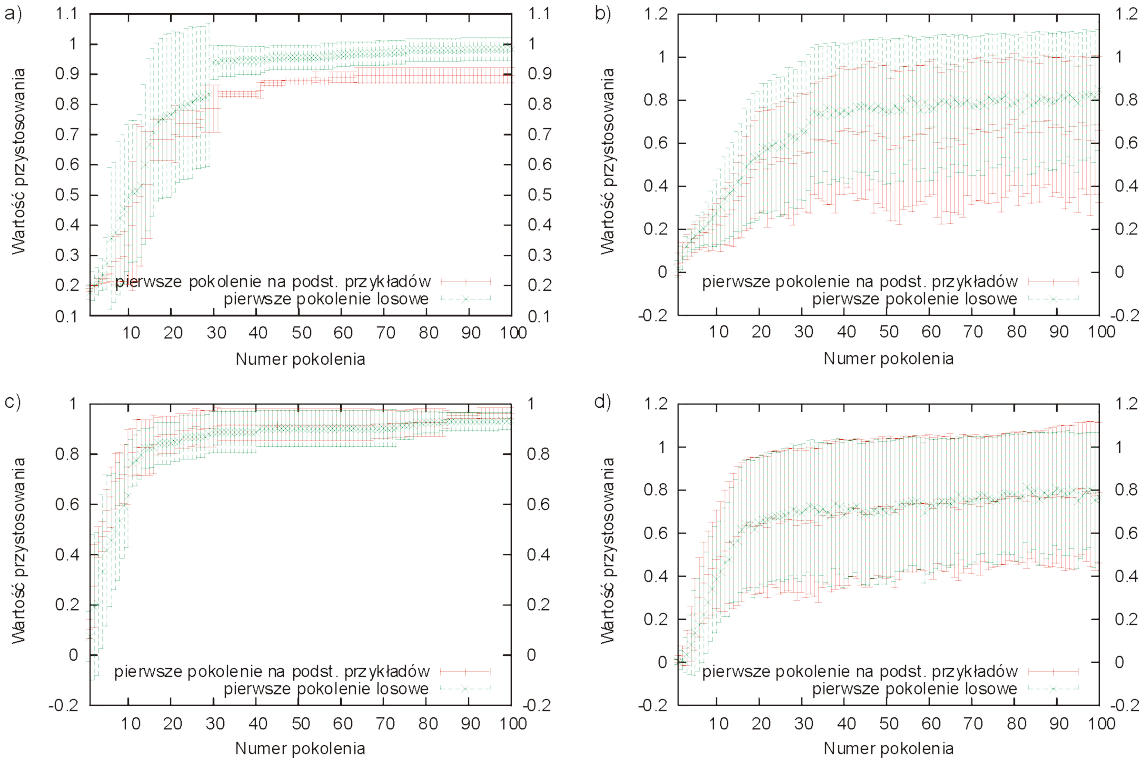
Tabela 6.12. Parametry kontrolujące proces ewolucji gramatyk

Nazwa parametru	*Wartość parametru*
Terminals	n v h
NTerminals	S N O P Q R
TotalNoOfGenerations	100
GenerationSize	100
MaxNoOfNT	4
MaxNoOfT	4
NTRedefinitionProb	0.8
InitialMaxTreeDepth	4
MaxTreeDepth	6
CrossoverProb	0.8
CrossoverOnTerminalPointProb	0.1
NoOfMutationTry	2
MutationProbabilityOnOneTry	0.05

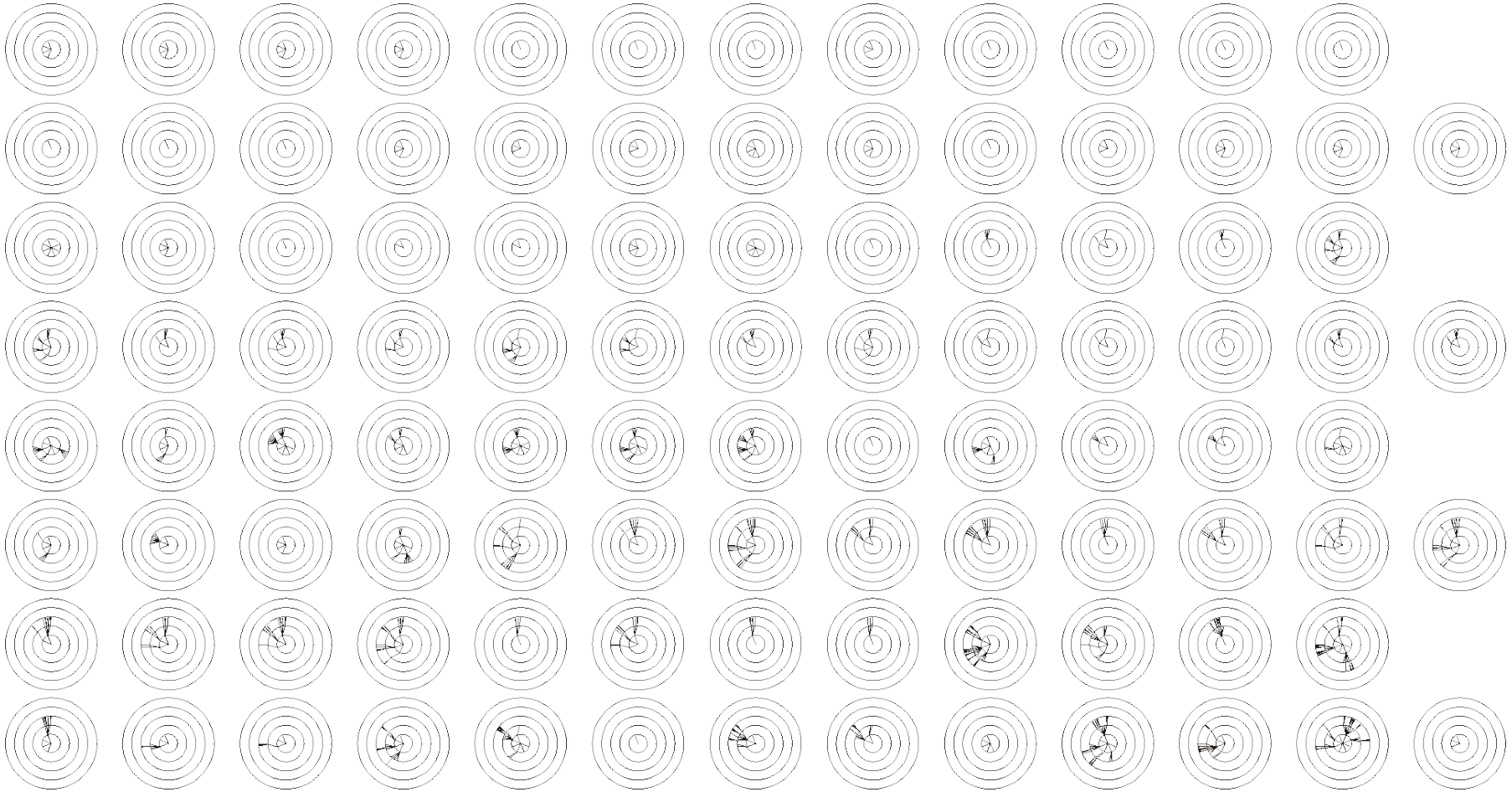
Natomiast w przypadku użycia parsera CYKP w początkowych pokoleniach zarówno wartość przystosowania najlepszego osobnika, jak i średnia wartość przystosowania wszystkich osobników w populacji jest większa dla wariantu generowania pierwszego pokolenia na podstawie przykładów języka. Wynika to głównie z tego, że w przypadku parsera CYKP wszystkie wygenerowane na podstawie przykładów gramatyki poprawnie rozpoznają przynajmniej jeden pozytywny przykład języka, co nie jest prawdą dla gramatyk wygenerowanych metodą *ramped half and half*. Jednak stosunkowo szybko (w prezentowanych tutaj wynikach około pokolenia nr 20) następuje zrównanie zarówno wartości przystosowania najlepszego osobnika, jak i średniej wartości przystosowania wszystkich osobników w danym pokoleniu.

Na rysunkach 6.33 i 6.34 przedstawiona została struktura drzew wygenerowanych w pierwszym pokoleniu odpowiednio dla metod generowania pierwszego pokolenia w sposób losowy (rys. 6.33) oraz na podstawie przykładów pozytywnych języka (rys. 6.34). Widać na nich, że pomimo tego, iż maksymalna wysokość drzew generowanych w sposób losowy wynosiła 4 (parametr **InitialMaxTreeDepth**), a maksymalna wysokość drzew generowanych na podstawie przykładów języka 6, to drzewa wygenerowane w sposób losowy mają bardziej złożoną

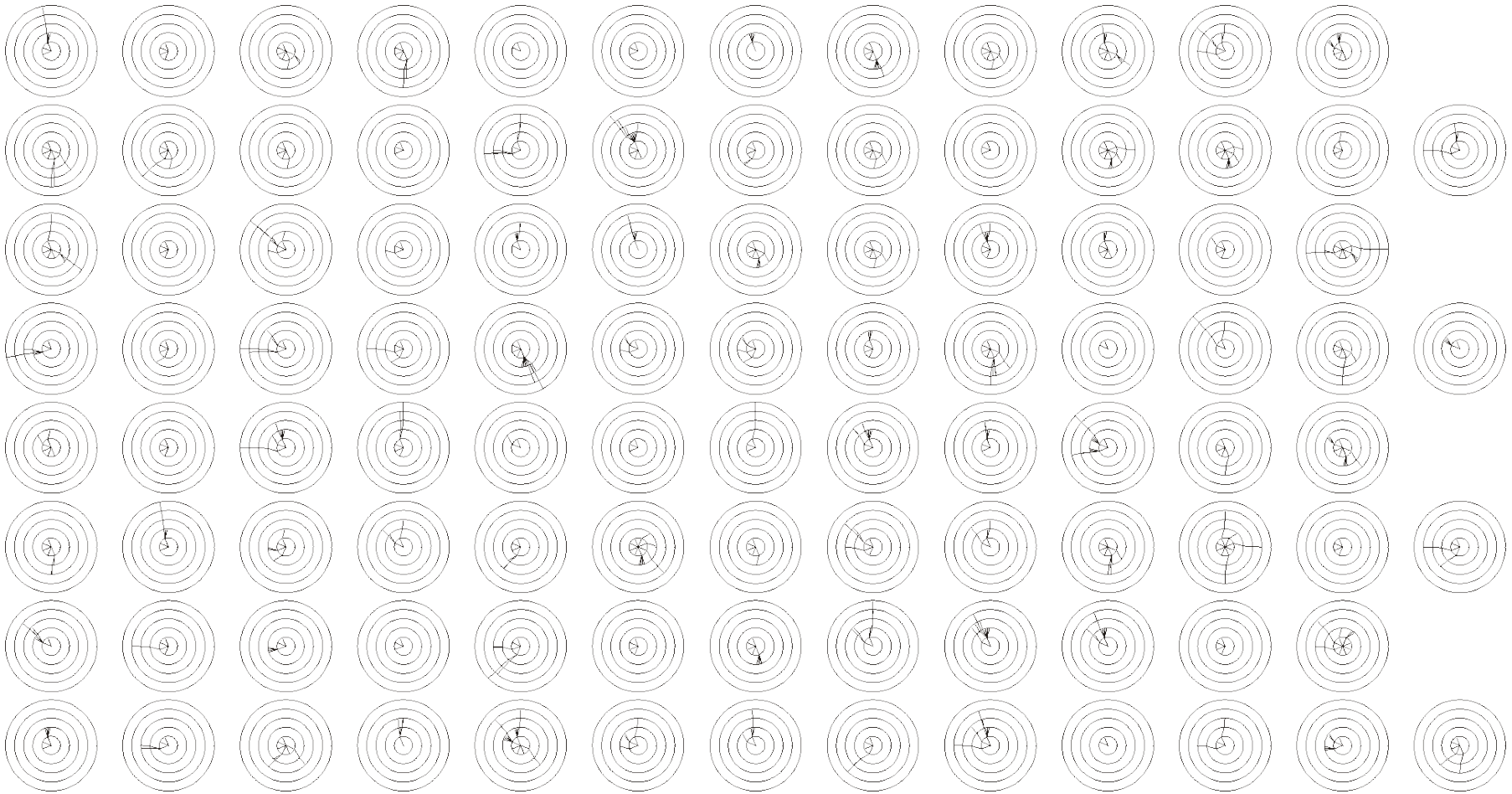
strukturę. Jest to jedna z przyczyn, ze względu na które w przypadku wykorzystania w procesie ewolucji parsera Bison, rezultaty osiągane dla pierwszego pokolenia generowanego w sposób losowy są lepsze niż na podstawie przykładów języka.



Rysunek 6.32 Porównanie wartości przystosowania osobników w procesie wnioskowania gramatyk opisujących przewężenia tętnic wieńcowych w zależności od sposobu generowania pierwszego pokolenia a) przystosowanie najlepszego osobnika parser Bison, b) średnie przystosowanie wszystkich osobników w populacji parser Bison, c) przystosowanie najlepszego osobnika parser CYKP, d) średnie przystosowanie wszystkich osobników w populacji parser CYKP



Rysunek 6.33. Struktura drzew wygenerowanych w sposób losowy dla pierwszego pokolenia w procesie wnioskowania gramatyk opisujących przewężenia tętnic wieńcowych



Rysunek 6.34. Struktura drzew wygenerowanych na podstawie przykładów języka dla pierwszego pokolenia w procesie wnioskowania gramatyk opisujących przewężenia tętnic wieńcowych

6.5. Model ewolucji 'ciągłej'

W rozdziale tym przedstawione zostaną wyniki uzyskane dla generowania gramatyk zgodnie z modelem ewolucji 'ciągłej' zaprezentowanym w rozdziale 4.6.2.

6.5.1. Generowanie gramatyk dla języka a^nb^n

Pierwszy przykład dotyczy zastosowania modelu ewolucji 'ciągłej' do generowania gramatyk dla języka $L_1 = \{a^nb^n \text{ dla } n > 0\}$. Podstawowe parametry sterujące przebiegiem ewolucji zostały przedstawione w tabeli 6.13; są one takie same jak w przypadku referencyjnym ewolucji klasycznej. Natomiast dodatkowe parametry sterujące przebiegiem ewolucji 'ciągłej' zawiera tabela 6.14.

Tabela 6.13. Podstawowe parametry kontrolujące proces ewolucji gramatyk

Nazwa parametru	*Wartość parametru*	*Opis*
Terminals	a b	Zbiór symboli terminalnych
NTerminals	S N O P Q R	Zbiór symboli nieterminalnych
TotalNoOfGenerations	100	Maksymalna ilość generowanych pokoleń
MaxNoOfNT	4	Maksymalna ilość symboli nieterminalnych, które mogą wystąpić w prawej stronie produkcji
MaxNoOfT	4	Maksymalna ilość symboli terminalnych, które mogą wystąpić w prawej stronie produkcji
NTRedefinitionProb	0.8	Prawdopodobieństwo, że dla symbolu nieterminalnego znajdującego się w prawej produkcji zostanie wygenerowane poddrzewo, którego będzie on korzeniem. Parametr ten ma znaczenie dla generowania pierwszego pokolenia metodami *grow* oraz *ramped half and half*
parserType	cykp	Rodzaj użytego w procesie ewolucji parsera lub generatora parserów

Tabela 6.14. Dodatkowe parametry kontrolujące proces ewolucji 'ciągłej'

Nazwa parametru	*Wartość parametru*	*Opis*
ChildStartFit	2	Wartość przystosowania dla nowo utworzonego osobnika (gramatyki).
NewCrossoverProb	0.65	Określa prawdopodobieństwo, iż nowo tworzony osobnik powstanie jako wynik

Nazwa parametru	*Wartość parametru*	*Opis*
		operacji krzyżowania dwóch osobników należących do populacji
`NewGenerateProb`	0.15	Określa prawdopodobieństwo, iż nowo tworzony osobnik powstanie jako w wyniku wygenerowania go metodą *ramped half and half*
`NewMutateProb`	0.2	Określa prawdopodobieństwo, iż nowo tworzony osobnik powstanie jako wynik operacji mutacji dla pewnego osobnika należącego do populacji
`NoOfConcurrentVersion`	20	Ilość osobników (gramatyk) wchodzących w skład jednego podzbioru posiadającego ustalone parametry sterujące przebiegiem ewolucji. Iloczyn `NoOfVariants` * `NoOfConcurrentVersion` określa ilość osobników wchodzących w skład całej populacji. W tym przypadku populacja liczy **100** osobników
`NoOfVariants`	5	Ilość równolicznych podzbiorów osobników różniących się parametrami sterującymi przebiegiem ewolucji (w badaniach prezentowanych w niniejszej rozprawie była to parametry `InitialMaxTreeDepth` oraz `MaxTreeDepth`)

Tabela 6.15 przedstawia parametry określające maksymalne rozmiary tworzonych drzew reprezentujących produkcje gramatyk dla poszczególnych wariantów (podzbiorów osobników wchodzących w skład populacji).

Tabela 6.15. Ograniczenia na rozmiar tworzonych drzew dla poszczególnych wariantów

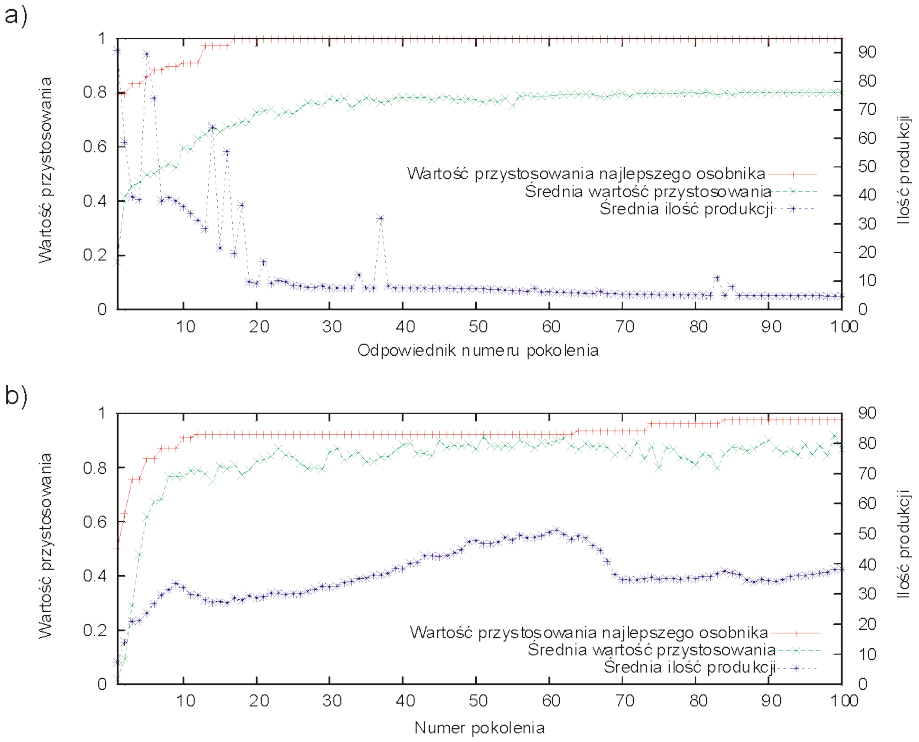
Nr wariantu	*Maksymalna wysokość drzew tworzonych na początku ewolucji (InitialMaxTreeDepth)*	*Maksymalna wysokość drzew tworzonych w trakcie ewolucji (MaxTreeDepth)*
1	1	2
2	2	4
3	3	6
4	4	8
5	5	10

Zbiór przykładów języka, na podstawie którego odbywał się proces generowania gramatyk, był taki jak przedstawiony w tabeli 6.1 w rozdziale 6.1.

Wyniki generowania gramatyk w procesie ewolucji 'ciągłej' przedstawione zostały na rysunku 6.35 a)[19]. Dla porównania, na rysunku 6.35 b) zaprezentowano

19 Jako że w modelu ewolucji 'ciągłej' nie występuje podział na poszczególne pokolenia gramatyk, aby możliwe było porównywanie uzyskanych wyników dla modelu ewolucji 'ciągłej' i modelu ewolucji

wyniki procesu generowania gramatyk z zastosowaniem klasycznego modelu ewolucji dla analogicznych parametrów sterujących przebiegiem ewolucji (pełny zbiór parametrów był taki, jak przedstawiono w tabeli 6.3).



Rysunek 6.35. Wyniki generowania gramatyk dla języka L_1 a) w modelu ewolucji 'ciągłej', b) w model ewolucji klasycznej

Jak widać na wykresach przedstawionych na rysunku 6.35, w przypadku ewolucji 'ciągłej' poprawna gramatyka (posiadająca wartość przystosowania równą 1.0) znajdowana jest już w odpowiedniku pokolenia nr 17. Natomiast w prezentowanym przypadku ewolucji klasycznej poprawnej gramatyki nie uzyskano aż do pokolenia nr 100 – najlepsza znaleziona gramatyka w tym pokoleniu miała wartość przystosowania 0.975. Także średni rozmiar gramatyk (poza początkiem ewolucji) jest znacznie mniejszy w przypadku ewolucji 'ciągłej' niż w przypadku ewolucji klasycznej.

Na rysunkach 6.36 i 6.37 przedstawione zostały produkcje najlepszych znalezionych gramatyk w pokoleniu nr 100 odpowiednio dla ewolucji 'ciągłej' i ewolucji klasycznej.

klasycznej wprowadzono dla modelu ewolucji 'ciągłej' „Odpowiednik numeru pokolenia". Jako kryterium równoważności przyjęto ilość testów przykładów języka przeprowadzanych na gramatykach wchodzących w skład populacji. Ponieważ w przypadku ewolucji klasycznej ilość testów przeprowadzanych dla jednego pokolenia wynosi: ([ilość przykładów pozytywnych] + [ilość przykładów negatywnych]) * [ilość osobników w populacji], taką samą ilość testów dla ewolucji ciągłej przyjęto traktować jako odpowiednik pokolenia.

```
S -> a N
N -> S O R N
N -> b
O -> ε
R -> ε
```

Rysunek 6.36. Najlepsza gramatyka znaleziona w pokoleniu nr 100, w przypadku ewolucji ciągłej. Wartość przystosowania tej gramatyki wynosi 1.0[20]

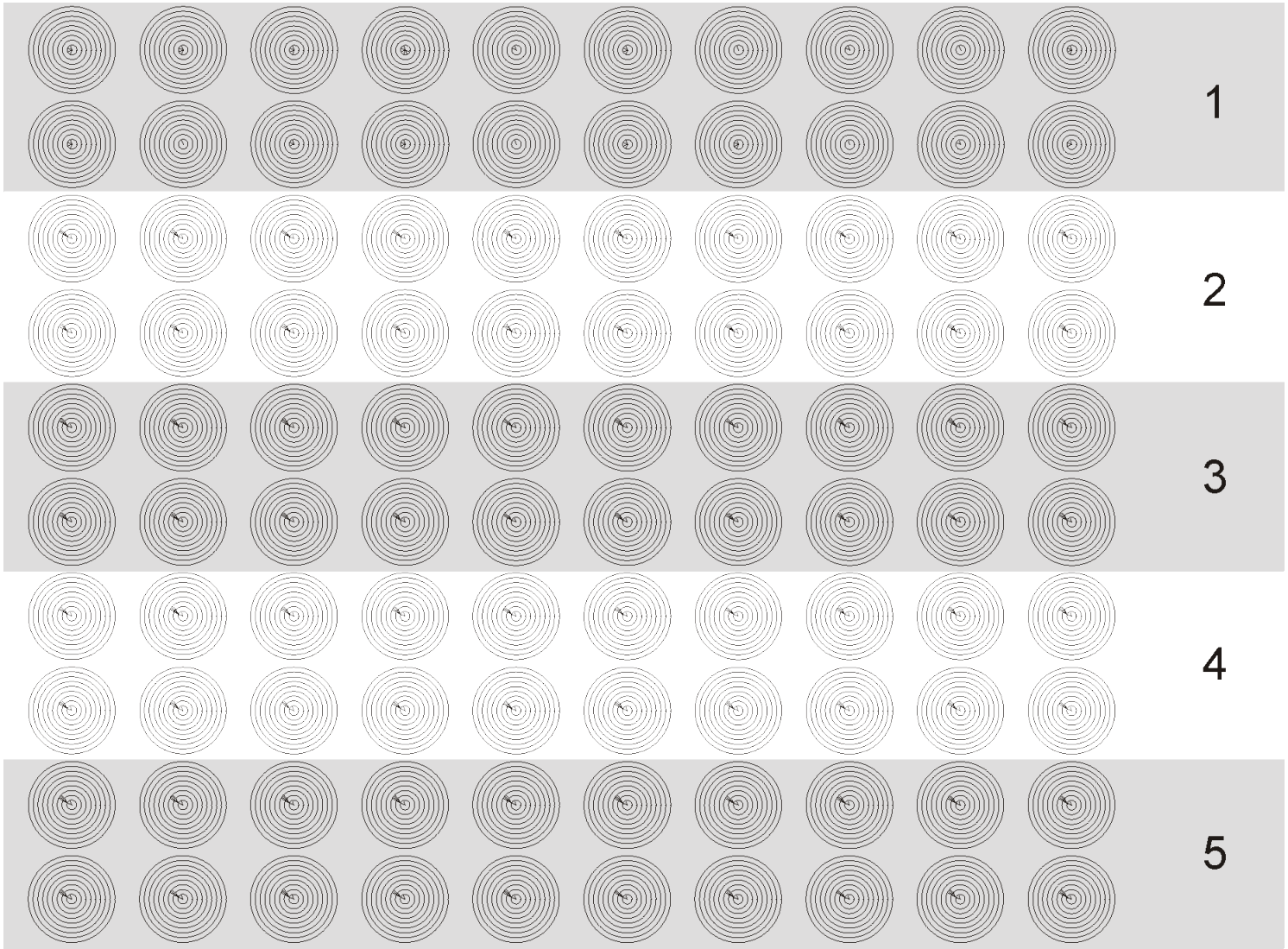
```
S -> a b
S -> a N N N
S -> b
S -> ε
S -> P b
N -> a b a a O O S
N -> a b a a Q O Q
N -> a b a N Q O Q
N -> N Q
N -> Q
N -> Q Q
N -> R b b P b
N -> R b N b
O -> b b a S
P -> ε
Q -> b b a S a R
Q -> b b a S Q R
Q -> b b b P O
Q -> Q b a S S R
Q -> S
Q -> S a a a a
Q -> S P a a a
Q -> S P a b
R -> a N Q
R -> a P O Q R
```

Rysunek 6.37. Najlepsza gramatyka znaleziona w pokoleniu nr 100, w przypadku ewolucji klasycznej. Wartość przystosowania tej gramatyki wynosi 0.975

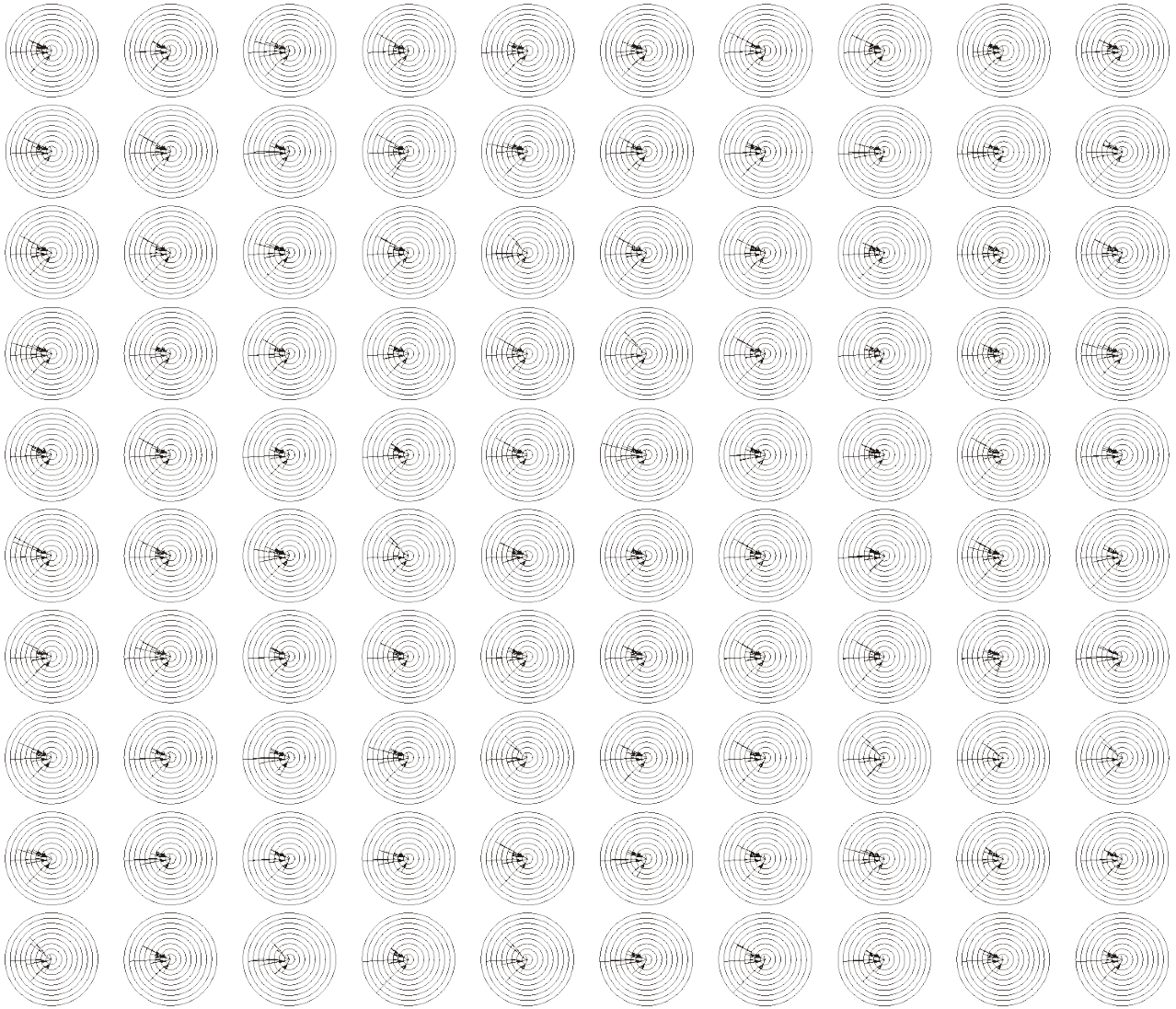
Gramatyka znaleziona w procesie ewolucji 'ciągłej', która ma wyższą wartość przystosowania, posiada także znacznie mniej produkcji (5) niż gramatyka znaleziona w procesie ewolucji klasycznej (25). Jednym z powodów, dla którego gramatyki znajdowane w procesie ewolucji 'ciągłej' są prostsze (w sensie ilości produkcji) niż gramatyki znajdowane w procesie ewolucji klasycznej, jest to, iż w przypadku ewolucji 'ciągłej' występuje podział populacji na poszczególne warianty, charakteryzujące się różną maksymalną wysokością tworzonych drzew opisujących produkcje gramatyk.

Na rysunkach 6.38 i 6.39 przedstawione zostało porównanie struktury powstałych drzew wchodzących w skład ostatniego pokolenia gramatyk dla przypadków ewolucji 'ciągłej' i ewolucji klasycznej.

20 Język generowany przez tą gramatykę zawiera w sobie język $a^n b^n$.



Rysunek 6.38. Struktura drzew w pokoleniu nr 100 dla modelu ewolucji 'ciągłej'. Po prawej stronie oznaczone zostały numery wariantów



Rysunek 6.39. Struktura drzew w pokoleniu nr 100 dla modelu ewolucji klasycznej

Rysunki te pokazują, że proces ewolucji 'ciągłej' prowadzi do utworzenia gramatyk posiadających znacznie prostszą strukturę niż ma to miejsce w przypadku ewolucji klasycznej. Jest to korzystne zarówno z punktu widzenia interpretacji otrzymanych wyników jak, i ich późniejszego wykorzystania w procesie syntaktycznego rozpoznawania wzorców (mniejsza ilość produkcji gramatyki prowadzi na ogół do krótszego czasu działania analizatora syntaktycznego rozpoznającego tę gramatykę).

Wnioski

Opracowany w ramach niniejszej rozprawy model ewolucji 'ciągłej' gramatyk posiada wiele zalet w stosunku do modelu ewolucji klasycznej. Do najważniejszych z nich można zaliczyć:

- Równoczesne poszukiwanie rozwiązań (gramatyk) o różnych rozmiarach. Ponieważ populacja osobników podzielona jest na podzbiory (warianty) różniące się parametrami określającymi rozmiary tworzonych drzew, więc w procesie ewolucji następuje poszukiwanie rozwiązań o różnym stopniu złożoności. Dzięki temu nie występuje, znany z ewolucji klasycznej, problem związany z określeniem maksymalnego rozmiaru poszukiwanego rozwiązania, tak aby z jednej strony rozmiar ten był wystarczający do znalezienia rozwiązania, a z drugiej, aby rozwiązania te nie były zbyt złożone. W przypadku ewolucji 'ciągłej' mamy do czynienia z sytuacją kilku (tylu, ile jest wariantów) równoczesnych procesów ewolucji, między którymi występuje przepływ informacji (ponieważ operatory krzyżowania i mutacji działają w ramach całej populacji, a nie jednego wariantu).
- Wczesna eliminacja osobników o niskiej wartości przystosowania. W przypadku ewolucji klasycznej ocena osobnika dokonywana jest na podstawie co najmniej pełnego zbioru przykładów pozytywnych języka (patrz rozdział 4.3), w związku z czym dla każdego nowo powstałego osobnika musi zostać sprawdzona poprawność klasyfikacji wszystkich przykładów pozytywnych języka. Proces ten może być bardzo czasochłonny, szczególnie w przypadku dużych[21] zbiorów przykładów języka. Ponieważ w przypadku ewolucji 'ciągłej' za każdym razem osobnik (gramatyka) testowany jest za pomocą tylko jednego przykładu języka i na podstawie poprawności klasyfikacji tego przykładu aktualizowana jest wartość jego przystosowania, może on zostać wyeliminowany z populacji bez konieczności sprawdzania poprawności klasyfikacji wszystkich przykładów pozytywnych, co stanowi bardzo istotny zysk czasowy.

21 W sensie ilości przykładów należących do tego zbioru oraz długości słów stanowiących poszczególne przykłady.

Przedstawione powyżej zalety modelu ewolucji 'ciągłej' oraz otrzymane przy jego zastosowaniu wyniki pozwalają wysnuć wniosek co do dużej użyteczności tego modelu dla celów ewolucyjnego generowania gramatyk bezkontekstowych na podstawie przykładów języka.

7. Zakończenie

7.1. Podsumowanie

Celem pracy było wykazanie następującej tezy:

Możliwe jest opracowanie nowych ewolucyjnych algorytmów programowania genetycznego pozwalających na dokonanie automatycznej generacji gramatyk bezkontekstowych wykorzystywanych w zadaniach inteligentnej analizy i interpretacji wybranych wzorców.

Tezę wykazano przez rozwiązanie następujących zadań szczegółowych:

1) Przyjęcie odpowiedniej reprezentacji gramatyk za pomocą struktur drzewiastych traktowanych jako osobniki podlegające procesowi ewolucji.
2) Określenie funkcji przystosowania (dopasowania) będącej miarą stopnia rozwiązania postawionego zadania poprawnej klasyfikacji zadanych przykładów języka przez poszczególne osobniki (gramatyki).
3) Określenie odpowiednich operatorów genetycznych – krzyżowania i mutacji, które będą efektywne w procesie ewolucji i równocześnie nie będą powodowały nadmiernego wzrostu osobników.
4) Opracowanie takiego modelu ewolucji, który zwiększy efektywność procesu generowania gramatyk i pozwoli na znalezienie lepszych rozwiązań.

Ponieważ w podejściu programowania genetycznego osobniki podlegające ewolucji są reprezentowane za pomocą struktur drzewiastych, pierwsze zadanie polegało na wyborze odpowiedniej reprezentacji gramatyki za pomocą takiej właśnie struktury. Przedstawiona w rozdziale 4.1 reprezentacja produkcji gramatyki za pomocą drzewa nawiązuje do podziału elementów, z których budowany jest program podlegający ewolucji, na zbiór funkcji $\mathcal{F}$ oraz zbiór terminali $\mathcal{T}$. W przedstawionej tu reprezentacji gramatyki w postaci drzewa produkcji do zbioru $\mathcal{F}$ należą symbole nieterminalne gramatyki, natomiast do zbioru $\mathcal{T}$ należą zarówno symbole terminalne, jak i nieterminalne gramatyki[22]. Należy zaznaczyć, że taki wybór reprezentacji produkcji gramatyki, mimo iż jest stosowany powszechnie przez innych autorów zajmujących się problematyką wnioskowania gramatyk w oparciu o podejście programowania genetycznego [9], [42], nie umożliwia przedstawienia produkcji dowolnej gramatyki

22 Jest to konsekwencją tego, że jako liście w drzewie produkcji gramatyki mogą występować zarówno symbole terminalne jak, i nieterminalne.

bezkontekstowej.

Drugie zadanie, którego przedmiotem było określenie funkcji przystosowania informującej o stopniu rozwiązania postawionego zadania polegającego na poprawnej klasyfikacji zadanych przykładów języka przez wygenerowane gramatyki, zostało rozwiązane na dwa sposoby. Pierwszy polegał na określeniu w rozdziale 4.3 podstawowej i rozszerzonej funkcji przystosowania, drugi zaś na zastosowaniu, w procesie ewolucji 'ciągłej', przystosowania modyfikowanego ilością poprawnie sklasyfikowanych przykładów języka. Główną zaletą zastosowanego w podejściu ewolucji 'ciągłej' obliczania przystosowania na podstawie sukcesywnie testowanych pojedynczych przykładów zamiast łącznego testowania wszystkich przykładów pozytywnych i ewentualnie negatywnych, jest krótszy czas potrzebny na odrzucenie niepoprawnej gramatyki (ponieważ nie muszą być sprawdzane wszystkie przykłady pozytywne), co istotnie skraca czas całego procesu ewolucji.

W ramach zadania trzeciego przebadana została przydatność w procesie generowania gramatyk bezkontekstowych trzech operatorów krzyżowania, tj. standardowego, zachowującego rozmiar oraz homologicznego. Zbadany został także wpływ poszczególnych operatorów na wzrost rozmiaru osobników (produkcji gramatyk) poddanych ewolucji. Zdefiniowanych zostało także pięć operatorów mutacji oraz przebadany został ich wpływ na proces ewolucji gramatyk. Dodatkowo opracowano algorytm równoważenia przyrostu rozmiaru drzew poddawanych mutacji polegający na dynamicznej zmianie udziału operatora delecji w stosunku do pozostałych operatorów mutacji. Jak to zostało przedstawione w rozdziale 6.2.8, algorytm ten pozwala na utrzymanie wartości oczekiwanej przyrostu rozmiaru drzew, spowodowanych działaniem operatorów mutacji w pobliżu wartości zero.

Ostatnim zadaniem, bazującym na wynikach uzyskanych we wszystkich poprzednich zadaniach, było opracowanie modelu ewolucji gramatyk, który zwiększyłby efektywność procesu generowania gramatyk bezkontekstowych na podstawie przykładów języka. W ramach tego zadania opracowany został model ewolucji 'ciągłej', w którym nie występuje podział na poszczególne pokolenia (rozdział 4.6.2), a także przeprowadzono porównanie jego skuteczności z podejściem klasycznym (rozdział 6.5). Opracowanie modelu ewolucji 'ciągłej', który pozwala na bardziej efektywne generowanie gramatyk charakteryzujących się lepszym przystosowaniem i mniejszą ilością produkcji niż gramatyki generowane w sposób klasyczny, należy uznać za jedno z najważniejszych osiągnięć pracy.

Dodatkowo w ramach przeprowadzonych badań porównano przydatność różnych analizatorów syntaktycznych w procesie generowania gramatyk (rozdział 6.3), sprawdzono wpływ sposobu generowania pierwszego pokolenia gramatyk na dalszy

przebieg ewolucji (rozdział 6.4), oraz zestawiono otrzymane wyniki z wynikami uzyskanymi przez innych autorów (rozdziały 6.3.2 i 6.3.3).

Wobec rozwiązania wszystkich, przedstawionych powyżej, zadań szczegółowych można wysunąć wniosek o wykazaniu prawdziwości postawionej tezy badawczej.

7.2. Wnioski

Uzyskane w ramach niniejszej pracy wyniki pozwalają na sformułowanie następujących wniosków:

1) Odpowiedni dobór operatorów genetycznych pozwala zwiększyć efektywność procesu ewolucji i jednocześnie ograniczyć nadmierny wzrost rozmiarów osobników. W przypadku operatorów krzyżowania sam sposób zdefiniowania operatora krzyżowania zachowującego rozmiar zapewnia, iż wartość oczekiwana przyrostu rozmiaru osobników spowodowana działaniem tego operatora będzie równa zero. Natomiast w przypadku operatorów mutacji, aby zminimalizować ich wpływ na zmianę rozmiaru osobników, konieczne okazało się opracowanie algorytmu ustalającego udział operatora delecji w stosunku do pozostałych operatorów mutacji.
2) Zastosowanie w procesie generowania gramatyk modelu ewolucji 'ciągłej' pozwala przeważnie na znajdowanie gramatyk o większej wartości przystosowania i mniejszej ilości produkcji niż w przypadku zastosowania modelu ewolucji klasycznej. Mniejsza ilość produkcji w znalezionych gramatykach pozwala na ich łatwiejszą interpretację przez człowieka oraz zmniejsza czas analizy syntaktycznej.
3) Opracowane metody ewolucyjnego generowania gramatyk bezkontekstowych mogą zostać wykorzystane dla celów automatycznego wnioskowania gramatyk opisujących zmiany chorobowe na różnego typu zobrazowaniach medycznych. Mimo że w pewnych przypadkach analizy syntaktycznej gramatyki bezkontekstowe okazują się niewystarczającym formalizmem i konieczne jest użycie gramatyk ciągowych o większej mocy, jak na przykład gramatyk klasy DPLL(k) [21], GDPLL(k) [32] lub też gramatyk grafowych [49], [72], to jednak dla wielu zastosowań związanych między innymi z analizą obrazów medycznych [50], [46], [71] gramatyki bezkontekstowe są w zupełności wystarczające. Dla takich zastosowań zaprezentowany system ewolucyjnego generowania gramatyk na podstawie przykładów języka może stanowić istotne udogodnienie, ponieważ

pozwala wyeliminować konieczność konstruowania przez człowieka odpowiedniej gramatyki klasyfikującej analizowane wzorce.

7.3. Kierunki dalszych badań

Planowane dalsze badania będą miały na celu poprawienie efektywności przedstawionego tu systemu wnioskowania gramatyk opartego na podejściu programowania genetycznego. Do najważniejszych zadań na przyszłość można zaliczyć:

- Opracowanie lepszej reprezentacji gramatyk bezkontekstowych za pomocą struktur drzewiastych. Ponieważ obecna reprezentacja narzuca istotne ograniczenia na postać produkcji gramatyki, ma to niekorzystny wpływ na przebieg procesu ewolucji i powoduje, że dla pewnych przypadków nie jest możliwe znalezienie właściwej gramatyki. Opracowanie bardziej ogólnej reprezentacji może w sposób istotny poprawić skuteczność systemu wnioskowania.
- Rozszerzenie definicji funkcji przystosowania w taki sposób, aby preferowane były w procesie ewolucji gramatyki o mniejszej ilości produkcji i zarazem bardziej ogólne. Tak rozszerzona funkcja przystosowania musi zapewnić także uniknięcie efektu nadmiernej generalizacji (ang. o*ver–generalization*) [26] generowanych gramatyk.
- Próbę zastosowania przedstawionego tu sposobu ewolucyjnego wnioskowania gramatyk na podstawie przykładów języka do generowania gramatyk grafowych, co powinno być możliwe po opracowaniu odpowiedniej reprezentacji gramatyk grafowych za pomocą struktur drzewiastych i użyciu w procesie ewolucji parsera dla gramatyk grafowych w miejsce stosowanego teraz parsera gramatyk bezkontekstowych.

Dodatek A. Parametry konfiguracyjne sterujące przebiegiem procesu ewolucji

Nazwa parametru	*Dopuszczalne wartości*	*Znaczenie*
CrossoverOnTerminalPointProb	Rzeczywiste w przedziale [0,1]	Określa prawdopodobieństwo, z jakim dopuszcza się, że w operacji krzyżowania losowo wybrany punkt w pierwszym drzewie będzie terminalem (wysokość wybranego poddrzewa będzie wynosić 0). W przeciwnym razie wybrane poddrzewo będzie musiało mieć wysokość > 0
CrossoverProb	Rzeczywiste w przedziale [0,1]	Określa prawdopodobieństwo, z jakim osobniki wchodzące w skład nowego pokolenia będą utworzone w wyniku operacji krzyżowania, w pozostałych przypadkach zostaną one utworzone w drodze reprodukcji (klonowania)
GenerationSize	Całkowite większe od 0	Ilość osobników (gramatyk) wchodzących w skład jednego pokolenia
InitialMaxTreeDepth	Całkowite większe od 1	Maksymalna, początkowa wysokość drzew reprezentujących produkcje gramatyk, jakie zostaną wygenerowane w pierwszym pokoleniu
MaxNoOfNT	Całkowite nieujemne	Maksymalna ilość symboli nieterminalnych, które mogą pojawić się w prawej stronie produkcji.
MaxNoOfSymbols	Całkowite nieujemne lub niezdefiniowane	Maksymalna ilość symboli terminalnych i nieterminalnych, które mogą pojawić się w prawej stronie produkcji. W przypadku, gdy parametr ten jest niezdefiniowany przyjmowana jest dla niego wartość **MaxNoOfNT + MaxNoOfT**
MaxNoOfT	Całkowite nieujemne	Maksymalna ilość symboli terminalnych, które mogą pojawić się w prawej stronie produkcji
MaxTreeDepth	Całkowite większe od 1	Maksymalna dopuszczalna wysokość drzew reprezentujących produkcje gramatyk w drugim i następnych pokoleniach
MutationProbabilityOnAttempt	Rzeczywiste w przedziale [0,1]	Prawdopodobieństwo zajścia operacji mutacji dla jednej próby mutacji. Iloczyn **NoOfMutationAttempts** i **MutationProbabilityOnAttempt** określa całkowite prawdopodobieństwo mutacji dla tworzonego osobnika
NoOfMutationAttempts	Całkowite większe od 0	Ilość prób mutacji przypadających na każdego osobnika tworzonego w ramach nowego pokolenia

Nazwa parametru	*Dopuszczalne wartości*	*Znaczenie*
NTRedefinitionProbability	Rzeczywiste w przedziale [0, 100]	Określa procentowo prawdopodobieństwo z jakim, w przypadku generowaniu drzewa metodą grow, dla nieterminala zostanie wygenerowane poddrzewo, którego będzie on korzeniem; czyli parametr ten określa prawdopodobieństwo, że nieterminal, który znajduje się w prawej stronie produkcji, stanie się lewą stroną jakiejś nowej produkcji. Im wartość tego parametru jest większa, tym generowane drzewo zawiera więcej produkcji. W przypadku, gdy wartość tego parametru wynosi 100 dla każdego nieterminala znajdującego się w drzewie zostanie wygenerowane poddrzewo, którego będzie on korzeniem (generowanie poddrzew będzie trwało, aż do osiągnięcia maksymalnej dopuszczalnej wysokości drzewa określonej przez parametr **InitialMaxTreeDepth**).
parserType	'yacc', 'elkhound', 'cykp'	Rodzaj użytego w procesie ewolucji parsera lub generatora parserów. 'yacc' oznacza użycie parsera Bison w trybie zgodnym z parserem Yacc (generowanie parserów klasy LALR, a nie GLR)
TotalNoOfGenerations	Całkowite większe od 0	Ilość pokoleń, po której zostanie zakończony proces ewolucji
tournamentSize	Całkowite większe od 0	Rozmiar turnieju (ilość osobników wchodzących w skład turnieju) dla przypadku selekcji turniejowej
w1	Rzeczywiste nieujemne	Waga operatora mutacji typu transwersja
w3	Rzeczywiste nieujemne	Waga operatora mutacji typu addycja terminala
w4	Rzeczywiste nieujemne	Waga operatora mutacji typu addycja nieterminala
w5	Rzeczywiste nieujemne	Waga operatora mutacji typu tranzycja nieterminala

Dodatkowe parametry sterujące przebiegiem ewolucji ciągłej

Nazwa parametru	***Dopuszczalne wartości***	***Znaczenie***
`ChildStartFit`	Rzeczywiste większe od 0	Wartość przystosowania dla nowo utworzonego osobnika (gramatyki).
`NewCrossoverProb`	Rzeczywiste w przedziale od 0.0 do 1.0	Określa prawdopodobieństwo, iż nowo tworzony osobnik powstanie jako wynik operacji krzyżowania dwóch osobników należących do populacji
`NewGenerateProb`	Rzeczywiste w przedziale od 0.0 do 1.0	Określa prawdopodobieństwo, iż nowo tworzony osobnik powstanie jako w wyniku wygenerowania go metodą *ramped half and half*
`NewMutateProb`	Rzeczywiste w przedziale od 0.0 do 1.0 Suma: **`NewCrossoverProb`** + **`NewMutateProb`** + **`NewGenerateProb`** powinna być równa 1.0	Określa prawdopodobieństwo, iż nowo tworzony osobnik powstanie jako wynik operacji mutacji dla pewnego osobnika należącego do populacji
`NoOfConcurrentVersion`	Całkowite większe od 0	Ilość osobników (gramatyk) wchodzących w skład jednego podzbioru posiadającego ustalone parametry sterujące przebiegiem ewolucji. Iloczyn **`NoOfVariants`** * **`NoOfConcurrentVersion`** określa ilość osobników wchodzących w skład całej populacji
`NoOfVariants`	Całkowite większe od 0	Ilość równolicznych podzbiorów osobników różniących się parametrami sterującymi przebiegiem ewolucji (w badaniach prezentowanych w niniejszej rozprawie była to parametry **`InitialMaxTreeDepth`** oraz **`MaxTreeDepth`**).

Dodatek B. Programy narzędziowe

brackets2bnf.pl

Zamienia reprezentację nawiasową drzewa produkcji gramatyk na postać BNF.
Program odczytuje ze standardowego wejścia linię zawierającą reprezentację nawiasową drzewa i wypisuje na standardowe wyjście produkcje gramatyki w notacji Backusa – Naura oraz symbol startowy gramatyki (korzeń drzewa produkcji).
W reprezentacji nawiasowej akceptowane są zarówno wierzchołki z określeniem typu (T – symbol terminalny, N – symbol nieterminalny, na przykład „(a T)" oznacza symbol terminalny „a", natomiast „(b N)" oznacza symbol nieterminalny „b") jak i bez tego określenia.

Przykłady:
1) Reprezentacja nawiasowa z określeniem typu wierzchołków

```
echo "(S N(a T)(b T))" | brackets2bnf.pl
```

Wynik działania:

```
Start symbol: S

S -> a b
```

2) Reprezentacja nawiasowa bez określenia typu wierzchołków

```
echo "(S N(a T)(b T))" | brackets2bnf.pl
```

Wynik działania:

```
Start symbol: S

S -> a b
```

get_entropy.pl

Oblicza entropię dla pokolenia gramatyk na podstawie danych zawartych w pliku result.txt.
Entropia obliczana jest zgodnie z definicją przedstawioną w rozdziale 4.4.
Poza entropią program podaje też wartość przystosowania najlepszego osobnika w danym pokoleniu oraz sumę przystosowania wszystkich osobników w pokoleniu.

Wywołanie:

```
get_entropy.pl pliku_z_danymi
```

gdzie:
pliku_z_danymi to ścieżka do pliku zawierającego wyniki ewolucji danego pokolenia gramatyk w takim formacie, jak zapisywane są przez program `grammar_generator` w pliku `result.txt`.

Format pliku wejściowego (`result.txt`):
[numer osobnika] fitness: [wartość przystosowania osobnika] tree height: [wysokość drzewa reprezentującego osobnika]
...

Przykład:

Zawartość pliku dane.txt:

```
1 fitness: 1 tree height: 6
```

```
2 fitness: 1 tree height: 6
3 fitness: 1 tree height: 6
4 fitness: 0.333333 tree height: 6
5 fitness: 1 tree height: 6
6 fitness: 1 tree height: 6
7 fitness: 1 tree height: 6
8 fitness: 1 tree height: 6
9 fitness: 1 tree height: 6
10 fitness: 1 tree height: 6
```

Wywołanie:

```
get_entropy.pl dane.txt
```

Wynik działania:

```
Entropy: 0.468995593589281
Best fit: 1
Sum fit: 9.333333
```

grammar_reductor.pl

Eliminuje powtarzające się produkcje w pliku wejściowym dla generatora analizatorów syntaktycznych Elkhound.
Działanie tego programu polega na wyeliminowaniu powtarzających się produkcji w pliku wejściowym dla generatora analizatorów syntaktycznych Elkhound, co przyspiesza proces generowania analizatora syntaktycznego.

Wywołanie:

```
grammar_reductor.pl pliku_zrodlowy_dla_generatora_parserow_Elkhound
```

gdzie:
`pliku_zrodlowy_dla_generatora_parserow_Elkhound` to ścieżka do pliku źródłowego zawierającego opis gramatyki w formacie akceptowanym przez generator parserów Elkhound.

Przykład:

Zawartość pliku `f1.y` zawierającego opis gramatyki w formacie źródłowym generatora Elkhound:

```
context_class Grammar : public UserActions {
public:
};

terminals {
  0 : TOK_EOF;
  1 : n;
  2 : v;
  3 : h;
}

nonterm(int) S {
  -> P S N R   []
  -> S Q   []
  -> P N   []
}

nonterm(int) N {
```

```
  -> h N  []
  -> h  []
  -> n N  []
  -> h  []
  -> R N N  []
  -> h  []
  -> h n  []
  -> h  []
  -> v h v  []
  -> n  []
}

nonterm(int) P {
  -> h  []
  -> n  []
  -> N  []
}

nonterm(int) Q {
  -> h h Q v  []
  -> v  []
  -> N h v v  []
}

nonterm(int) R {
  -> P Q N N  []
  -> n v h  []
}
```

Wywołanie:

```
grammar_reductor.pl f1.y
```

Wynik działania:

```
context_class Grammar : public UserActions {
public:
};

terminals {
  0 : TOK_EOF;
  1 : n;
  2 : v;
  3 : h;
}


/* After grammar_reduce: 18 productions */

nonterm(int) S {
  -> P S N R  []
  -> S Q  []
  -> P N  []
}
nonterm(int) N {
  -> h N  []
  -> h  []
  -> n N  []
  -> R N N  []
  -> h n  []
  -> v h v  []
  -> n  []
}
```

```
nonterm(int) P {
  -> h  []
  -> n  []
  -> N  []
}
nonterm(int) Q {
  -> h h Q v  []
  -> v  []
  -> N h v v  []
}
nonterm(int) R {
  -> P Q N N  []
  -> n v h  []
}
```

Dodatek C. Zbiory przykładów języków

Zbiory przykładów języka dla wnioskowania gramatyk opisujących przewężenia tętnic wieńcowych:

S1 – zbiór zawierający jeden przykład pozytywny i jeden przykład negatywny

Przykłady pozytywne	***Przykłady negatywne***
nnhhv	hn

S2 – zbiór zawierający 20 przykładów pozytywnych i 20 przykładów negatywnych

Przykłady pozytywne	***Przykłady negatywne***
nnhhv, nnnv, nnnnhhh, nnhhhv, nh, nnh, nnhv, nhhhhvvvv, nhvvvv, nnhhvvvvvv, nhhhh, nnnnnnnvvv, nnnnnnv, nnnhh, nnnhv, nhv, nnnnnhhh, nnnnnnnnhhhh, nnnnh, nnnhvv	hn, hnv, nhn, nnhnv, vh, nhhn, hv, hvhv, nhhvvh, nnhvvvn, nhhvvh, nhvh, nvhvhn, hhhnnnv, vnvn, hnvn, vvn, vvnh, vvvnh, vvh

S3 – zbiór zawierający 50 przykładów pozytywnych i 50 przykładów negatywnych

Przykłady pozytywne	***Przykłady negatywne***
nnhhv, nnnv, nnnnhhh, nnhhhv, nh, nnh, nnhv, nhhhhvvvv, nhvvvv, nnhhvvvvvv, nhhhh, nnnnnnnvvv, nnnnnnv, nnnhh, nnnhv, nhv, nnnnnhhh, nnnnnnnnhhhh, nnnnh, nnnhvv, nnv, nhhhhhhhv, nnhhhvvvvv, nnhh, nhhvv, nnvvvvvv, nnnh, nvvvvv, nhh, nhhhhhhh, nnvv, nnnhhvvv, nnnnhh, nnnnnnvv, nnnnhhv, nhhh, nnnhhhhvvv, nhhhhh, nvvv, nnnnnvv, nhvv, nnnhvvvvvvvvv, nhhv, nhhhv, nhhhhv, nnnnnnnhv, nv, nvv, nhhvvv, nvvvv	hn, hnv, nhn, nnhnv, vh, nhhn, hv, hvhv, nhhvvh, nnhvvvn, nhhvvh, nhvh, nvhvhn, hhhnnnv, vnvn, hnvn, vvn, vvnh, vvvnh, vvh, vvvn, vvvh, vvvnhv, hhnv, hhhhvhn, nnnvvvhhhn, hnvnnvnnn, hhhnvnnvvnnn, hhhnvnnvvv, hhnnvv, hnnvn, hhhhnnvn, n, nn, nnn, nnnn, nnnnn, nnnnnn, v, vv, vvv, vvvv, vvvvv, vvvvvv, h, hh, hhh, hhhh, hhhhh, hhhhhh

S4 – zbiór zawierający 50 przykładów pozytywnych i 41 przykładów negatywnych

Przykłady pozytywne	***Przykłady negatywne***
nnhhv, nnnv, nnnnhhh, nnhhhv, nh, nnh, nnhv, nhhhhvvvv, nhvvvv, nnhhvvvvvv, nhhhh, nnnnnnnvvv, nnnnnnv, nnnhh, nnnhv, nhv, nnnnnhhh, nnnnnnnnhhhh, nnnnh, nnnhvv, nnv, nhhhhhhhv, nnhhhvvvvv, nnhh, nhhvv, nnvvvvvv, nnnh, nvvvvv, nhh, nhhhhhhh, nnvv, nnnhhvvv, nnnnhh, nnnnnnvv, nnnnhhv, nhhh, nnnhhhhvvv, nhhhhh, nvvv, nnnnnvv, nhvv, nnnhvvvvvvvvv, nhhv, nhhhv, nhhhhv, nnnnnnnhv, nv, nvv, nhhvvv, nvvvv	hn, hnv, nhn, nnhnv, vh, nhhn, hv, hvhv, nhhvvh, nnhvvvn, nhhvvh, nhvh, nvhvhn, hhhnnnv, vnvn, hnvn, vvn, vvnh, vvvnh, vvh, vvvn, vvvh, vvvnhv, hhnv, hhhhvhn, nnnvvvhhhn, n, nn, nnn, nnnn, nnnnn, v, vv, vvv, vvvv, vvvvv, h, hh, hhh, hhhh, hhhhh

Dodatek D. Algorytm tworzenia drzewa produkcji na podstawie przykładu języka

Poniżej przedstawiony został algorytm tworzenia drzewa reprezentującego produkcje gramatyki na podstawie jednego przykładu pozytywnego języka. Algorytm został zapisany w języku Lua.

```
-- creates tree base on positive sample
function genTreeFromSample(sample, treeParams)
   -- debug.sethook(print, "l")
   local startSymbol = getStartSymbol()
   -- create start symbol (root node)
   local startNode = newGrammarTreeNode(startSymbol, false)
   -- create grammar tree
   local tree = GrammarTree()
   tree:setRoot(startNode)
   genSubTreeFromSample(sample, treeParams, startNode)
   return tree
end


-- creates tree node (terminal or nonterminal)
function newGrammarTreeNode(name, isTerminal)
   local nTNode = GrammarTreeNode()
   nTNode:getValue():setName(name)
   nTNode:getValue():setTerminal(isTerminal)
   nTNode:getValue():setVisible(true)
   return nTNode;
end;


-- add terminals from sample to NTSymbol
function genSubTreeTerminalsOnly(sample, NTSymbol)
   for i = 1, string.len(sample) do
      local c = string.sub(sample, i, i)
      local tNode = newGrammarTreeNode(c, true)
      NTSymbol:addChild(tNode)
   end
end


-- returns start symbol (STS)
function getStartSymbol()
   local s = XConfig["NTerminals"]
   local nt = {}
   local w
```

```
   for w in string.gfind(s, "(%a)") do
      table.insert(nt, w)
   end
   return nt[1] -- return first nonterminal symbol (STS)
end


-- returns random nonterminal symbol from NTerminals set
function getRandomNT()
   local s = XConfig["NTerminals"]
   local nt = {}
   local w
   for w in string.gfind(s, "(%a)") do
      table.insert(nt, w)
   end
   local n = math.random(1, table.getn(nt))
   return nt[n]
end

-- inserts sub tree to NTSymbol
function genSubTreeFromSample(posSample, treeParams, NTSymbol)
   local sampleStr = posSample
   local tno = 0
   local ntno = 0

   while string.len(sampleStr) > 0 do
      local pt = 0  -- type of symbol -> 0: terminals, 1: NT

      if tno == treeParams.maxNoOfProductionT then
         pt = 1
      elseif ntno == treeParams.maxNoOfProductionNT  then
         pt = 0
      else
         pt = math.random(0,1)
      end
      if (pt == 1) and
         (ntno == (treeParams.maxNoOfProductionNT-1) or
             (ntno + tno) == (treeParams.maxNoOfProductionSymbols-1))
then
         -- last NT
         local te =
            math.random(0,math.min(string.len(sampleStr)-1,
                      treeParams.maxNoOfProductionT-tno,
                                 treeParams.maxNoOfProductionSymbols-
(tno+ntno+1)))

          local str = string.sub(sampleStr, 1, string.len(sampleStr)-
te)
         local newNTNode = newGrammarTreeNode(getRandomNT(), false)
```

```
            genSubTreeFromSample(str, treeParams, newNTNode)
            NTSymbol:addChild(newNTNode)
            ntno = ntno + 1
               sampleStr = string.sub(sampleStr,1+string.len(sampleStr)-
te,-1)
         elseif (pt == 0) and (ntno == treeParams.maxNoOfProductionNT)
and
            (tno == (treeParams.maxNoOfProductionT-1) or
                (ntno + tno) == (treeParams.maxNoOfProductionSymbols-1))
then
            -- last T (no more NT)
            genSubTreeTerminalsOnly(sampleStr, NTSymbol)
            sampleStr = ""
            tno = tno + string.len(sampleStr)
         elseif (pt == 1) then
            local endIdx = math.random(1, string.len(sampleStr))
            local str = string.sub(sampleStr, 1, endIdx)
            local nts = getRandomNT()
            local newNTNode = newGrammarTreeNode(nts, false)
            genSubTreeFromSample(str, treeParams, newNTNode)
            NTSymbol:addChild(newNTNode)
            ntno = ntno + 1
            sampleStr = string.sub(sampleStr, endIdx+1, -1)
         elseif (pt == 0) then
            local maxT =
               math.min(string.len(sampleStr),
                        treeParams.maxNoOfProductionT-tno,
                        treeParams.maxNoOfProductionSymbols-(tno+ntno))
            if maxT >= 1 then
               local endIdx = math.random(1, maxT)
               local str = string.sub(sampleStr, 1, endIdx)
               genSubTreeTerminalsOnly(str, NTSymbol)
               sampleStr = string.sub(sampleStr,endIdx+1,-1)
               tno = tno + string.len(str)
            end
         else
            print("Unexpected code while genTreeFromSample")
            exit(1)
         end
      end
   end
```

Bibliografia

[1] Aho A.V., Sethi R., Ullman J.D., *Kompilatory. Reguły, metody i narzędzia*, WNT, Warszawa 2002

[2] Angeline P.J., *A Historical Perspective on the Evolution of Executable Structures*, Fundamenta Informaticae, 35 (1998), 179-195

[3] Angluin D., *Learning regular sets from queries and counterexamples*, Information and Computation, 75 (1987), 87-106

[4] Arabas J., *Wykłady z algorytmów ewolucyjnych*, WNT, Warszawa 2001

[5] Ball G., Kurlander D., Pugh D. i inni, *Lifelike Computer Characters: The Persona Project at Microsoft Research*, Software Agents, (1997), 191-222

[6] Booch G., Rumbaugh J., Jacobson I., *UML przewodnik użytkownika*, WNT, 2001

[7] Chomsky N., *Three models for the description of language*, IRE Transactions on Information Theory, vol. 2, no. 1 (1956), 113-124

[8] Clouse D.S., Giles C.L., Horne B.G., Cottrell G.W., *Representation and Induction of Finite State Machines using Time-Delay Neural Networks*, NIPS, (1996), 403-409

[9] Crepinsek M., Mernik M., Javed F., Bryant B.R., Sprague A.P., *Extracting grammar from programs: evolutionary approach*, SIGPLAN Notices, 40 (2005), 39-46

[10] D'haeseleer P., *Context Preserving Crossover in Genetic Programming*, Proceedings of the 1994 IEEE World Congress on Computational Intelligence 27-29 June 1994, 1 (1994), 256-261

[11] Daida J.M., Hilss A.M., Ward D.J., Long S.L., *Visualizing Tree Structures in Genetic Programming*, , 6 (2005), 79-110

[12] Das S., Giles C.L., Sun G.Z., *Learning Context-Free Grammars: Capabilities and Limitations of a Recurrent Neural Network with an External Stack Memory*, Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society, (1992), 791-795

[13] Das S., Giles C.L., Sun G.Z., *Using Prior Knowledge in a NNPDA to Learn Context-Free Languages*, Advances in Neural Information Processing Systems, 5 (1993), 65-72

[14] Dupont P., Miclet L., Vidal E., *What is the search space of the regular inference*, Proceedings of the Second International Colloquium on Grammar Inference (ICGI'94), LNAI, 862 (1994), 25-37

[15] Elman J.L., *Distributed Representations, Simple Recurrent Networks, and Grammatical Structure*, Machine Learning, 7 (1991), 195-225

[16] Elman J.L., *Finding Structure in Time*, Cognitive Science, 14 (1990), 179-211

[17] Flasiński M., *On the Parsing of Deterministic Graph Languages for Syntactic Pattern Recognition*, Pattern Recognition, 26 (1993), 1-16

[18] Flasiński M., *Power Properties of NLC Graph Grammars with a Polynomial Membership Problem*, Theoretical Computer Science, 201 (1998), 189-231

[19] Flasiński M., *Strukturalna analiza obrazów za pomocą gramatyk grafowych klasy ETPL(k)*, Rozprawy Habilitacyjne UJ, Kraków 1992

[20] Flasiński M., *Syntaktyczne metody rozpoznawania obrazów*, Wydawnictwo Uniwersytetu Jagiellońskiego, Kraków 1990

[21] Flasiński M., Reroń E., Jurek J., Wójtowicz P., Atłasiewicz K., *Mathematical Linguistics Model for Medical Diagnostics of Organ of Hearing in Neonates*, Lecture Notes in Computer Science, 3019 (2004), 746-753

[22] Friedberg R.M., *A Learning Machine: Part I*, IBM Journal of Research and Development, vol. 2 no. 1 (1958), 2-13

[23] Friedberg R.M., Dunham B., North J.H., *A Learning Machine: Part II*, IBM Journal of Research and Development, vol. 3 no. 3 (1959), 282-287

[24] Fu K.S., *Syntactic Pattern Recognition and Applications*, Prentice-Hall, Englewood Cliffs, NJ 1982

[25] Giles C.L., Miller C.B., Chen D. i inni, *Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks*, Neural Computation, 4 (1992), 393-405

[26] Gold E.M., *Language identification in the limit*, Information and Control, 10 (1967), 447-474

[27] Goldberg D.E., *Algorytmy genetyczne i ich zastosowania*, WNT, Warszawa 1995

[28] Higuera C., *Current Trends in Grammatical Inference*, Advances in Pattern Recognition, Joint IAPR International Workshops SSPR+SPR'2000, 1876 (2001), 28-31

[29] Holland J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975

[30] Hopcroft J.E., Ullman J.D., *Wprowadzenie do teorii automatów, języków i obliczeń*, PWN, Warszawa 2003

[31] Ierusalimschy R., *Programming in Lua*, Lua.org, 2003

[32] Jurek J., *Syntaktyczne rozpoznawanie obrazów za pomocą gramatyk ciągowych klasy GDPLL(k)*, Wydawnictwo Uniwersytetu Jagiellońskiego, 2005

[33] Koza J.R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992

[34] Koza J.R., *Hierarchical Genetic Algorithms Operating on Populations of Computer Programs*, Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 1 (1989), 768-774

[35] Langdon W.B., *Size Fair and Homologous Tree Genetic Programming Crossovers*, Proceedings of the Genetic and Evolutionary Computation Conference, 2 (1999), 1092-1097

[36] Langley P., *Simplicity and Representation Change in Grammar Induction*, Technical report, Stanford University, Palo Alto, CA 1995

[37] Lankhorst M., *A genetic algorithm for induction of nondeterministic push-down automata*, Technical report, University of Groningen, Computer Science Report CS-R 9502, 1995

[38] Lawrence S., Fong S., Giles C.L., *Natural Language Grammatical Inference: A Comparison of Recurrent Neural Networks and Machine Learning Methods*, Symbolic, Connectionist, and Statistical Approaches to Learning for Natural Language Processing, (1996), 33-47

[39] Lawrence S., Giles C.L., Fong S., *Natural Language Grammatical Inference with Recurrent Neural Networks*, IEEE Transactions on Knowledge and Data Engineering, vol. 12 no. 1 (2000), 126-140

[40] McPeak S., *Elkhound: A Fast, Practical GLR Parser Generator*, Technical report, University of California, Berkeley UCB/CSD-02-1214, 2002

[41] McPeak S., Necula G.C., *Elkhound: A Fast, Practical GLR Parser Generator*, Compiler Construction, 13th International Conferencer, LNCS, 2985 (2004), 73-88

[42] Mernik M., Crepinsek M., Gerlic G. et al., *Learning Context-Free Grammars using an Evolutionary Approach*, Technical report, University Maribor, 2003

[43] Mernik M., Gerlic G., Zumer V., Bryant B., *Can a Parser be Generated from Examples ?*, Proceedings of the ACM symposium on Applied Computing, (2003), 1063-1067

[44] Michalewicz Z., *Algorytmy genetyczne + struktury danych = programy ewolucyjne*, WNT, Warszawa 2003

[45] Ogiela M.R., *Strukturalne metody rozpoznawania obrazów w kognitywnej analizie zobrazowań medycznych*, UWND, Kraków 2004

[46] Ogiela M.R., *Syntaktyczne metody rozpoznawania obrazów i ich wykorzystanie w analizie wybranych obrazów medycznych*, Rozprawy Monografie nr 100, Kraków 2001

[47] Ogiela M.R., Tadeusiewicz R., *Artificial intelligence structural imaging techniques in visual pattern analysis and medical data understanding*, Pattern Recognition, 36 (2003), 2441-2452

[48] Ogiela M.R., Tadeusiewicz R., *Picture Languages in Intelligent Retrieval of Visual Data Semantic Information*, PAKM 2004, LNAI, 3336 (2004), 389-396

[49] Ogiela M.R., Tadeusiewicz R., *Picture Languages in Medical Pattern Knowledge Representation and Understanding*, MDAI 2005, LNAI, 3558 (2005), 442-447

[50] Ogiela M.R., Tadeusiewicz R., *Syntactic reasoning and pattern recognition for analysis of coronary artery images*, Artificial Intelligence in Medicine, 26 (2002), 145-159

[51] Omlin C.W., Giles C.L., *Extraction of Rules from Discrete-time Recurrent Neural Networks*, Neural Networks, 9 (1996), 41-52

[52] Oncina J., Garcia P., *Inferring regular languages in polynomial update time*, Pattern Recognition and Image Analysis, 1 (1992), 49-61

[53] Pałka D., *Ewolucyjne generowanie gramatyk bezkontekstowych z wykorzystaniem algorytmów genetycznych*, Półrocznik AGH Elektrotechnika i Elektronika, 23 (2004), 58-63

[54] Pałka D., *Generowanie gramatyk bezkontekstowych dla potrzeb syntaktycznego rozpoznawania wzorców*, Informatyka Teoretyczna i Stosowana, R. 4 nr 7 (2004), 139-147

[55] Pałka D., *Środowisko programowania genetycznego GPSource*, Informatyka Teoretyczna i Stosowana, R. 3 nr 5 (2003), 201-207

[56] Pałka D., Piekarczyk M., *Operatory krzyżowania w ewolucyjnym generowaniu gramatyk bezkontekstowych*, Informatyka Teoretyczna i Stosowana, w druku

[57] Parekh R., Honavar V., *An Incremental Interactive Algorithm for Regular Grammar Inference*, Proceedings of the Third ICGI-96, LNAI, 1147 (1996), 238-249

[58] Parekh R., Honavar V., *Automata Induction, Grammar Inference, and Language Acquisition*, Handbook of Natural Language Processing, New York 2000

[59] Parekh R., Honavar V., *Efficient Learning of Regular Languages Using Teacher-Supplied Positive Samples and Learner-Generated Queries*, Proceedings of the Fifth UNB Conference on AI, (1993), 195-203

[60] Parekh R., Nichitiu C., Honavar V., *A Polynomial Time Incremental Algorithm for Learning DFA*, Grammatical Inference, 4th International Colloquium ICGI-98, 1433 (1998), 37-49

[61] Pinker S., *Formal Models of Language Learning*, Cognition, 7 (1979), 217-283

[62] Pinker S., *The Language Instinct*, William Morrow and Company, New York 1994

[63] Pollack J.B., *The Induction of Dynamical Recognizers*, Machine Learning, 7 (1991), 123-148

[64] Rekers J., *Parser Generation for Interactive Environments*, PhD thesis, Amsterdam 1992

[65] Rosca J.P., *Entropy-Driven Adaptive Representation*, Proceedings of the Workshop on Genetic Programming From Theory to Real-World Applications, (1995), 23-32

[66] Shaw A.C., *A Formal Picture Description Scheme as a Basis for Picture Processing Systems*, Information and Control, 14 (1969), 9-52

[67] Skomorowski M., *Syntactic Recognition of Distorted Patterns*, IEC 2005: Prague, Czech Republic, Enformatika, 7 (2005), 173-177

[68] Skomorowski M., *Syntaktyczno-statystyczny model rozpoznawania obrazów zniekształconych*, Wydawnictwo UJ, Kraków 2000

[69] Tadeusiewicz R., Flasiński M., *Rozpoznawanie obrazów*, PWN, Warszawa 1991

[70] Tadeusiewicz R., Korohoda P., *Komputerowa analiza i przetwarzanie obrazów*, Wydawnictwo Postępu Telekomunikacji, Kraków 1997

[71] Tadeusiewicz R., Ogiela M.R., *Medical Image Understanding Technology*, Springer Verlag, Berlin, Heidelberg 2004

[72] Tadeusiewicz R., Ogiela M.R., *Picture languages in automatic radiological palm interpretation*, Int. J. Appl. Math. Comput. Sci., vol. 15 no. 2 (2005), 305-312

[73] Tadeusiewicz R., Ogiela M.R., *Structural approach to medical image understanding*, Bulletin of the Polish Academy of Sciences Technical Sciences, 52 (2004), 131-139

[74] Wyard P., *Representational Issues for Context Free Grammar Induction Using Genetic Algorithms*, ICGI '94: Proceedings of the 2nd International Colloquium on Grammatical Inference and Applications, (1994), 222-235

[75] *Języka Lua*, http://www.lua.org

[76] *Projekt GP Grammar*, http://gp-grammar.sourceforge.net

[77] *Projekt luabind*, http://luabind.sourceforge.net