Prace Naukowe Instytutu Informatyki, Automatyki
i Robotyki Politechniki Wrocławskiej **105**

Seria:
Monografie **29**

Olgierd Unold

Ewolucyjne wnioskowanie gramatyczne

Oficyna Wydawnicza Politechniki Wrocławskiej · Wrocław 2006

Recenzenci
Halina KWAŚNICKA
Zbigniew MICHALEWICZ

Opracowanie redakcyjne i korekta
Dorota RAWA

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
http://www.oficyna.pwr.wroc.pl
e-mail: oficwyd@pwr.wroc.pl

ISSN 0324-9786

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr 1098/2006.

Prace Naukowe Instytutu Informatyki, Automatyki i Robotyki
Nr 105 Politechniki Wrocławskiej Nr 105

Studia i Materiały Nr 29 2006

uczenie maszynowe, wnioskowanie gramatyczne,
uczące się systemy klasyfikujące, metody ewolucyjne,
języki formalne, przetwarzanie języka naturalnego,
bioinformatyka

Olgierd UNOLD*

EWOLUCYJNE WNIOSKOWANIE GRAMATYCZNE

W monografii została podjęta ważna i płodna zarówno teoretycznie, jak i praktycznie tematyka wnioskowania gramatycznego (maszynowego uczenia gramatyk). Zaproponowano nowy model ewolucyjnego wnioskowania gramatycznego, którego zasadniczym przeznaczeniem jest indukcja gramatyki bezkontekstowej. Konstrukcja nowego modelu ewolucyjnego wykorzystuje mechanizm uczenia stosowany w uczących się systemach klasyfikujących. W modelu klasyfikatorami są produkcje gramatyki bezkontekstowej podane w postaci normalnej Chomsky'ego, natomiast otoczeniem, do którego adaptuje się system, jest zbiór uczący składający się z przykładowych zdań opatrzonych etykietą określającą przynależność lub brak przynależności zdania do poszukiwanego języka. Celem uczenia jest poprawna klasyfikacja zdań uczących. Ponieważ zbiór klasyfikatorów tworzy zestaw produkcji gramatyki, poprawna klasyfikacja etykietowanych zdań oznacza wyindukowanie poszukiwanej gramatyki języka. Model śledzi produkcje użyte podczas analizy zbioru uczącego i po jej zakończeniu oblicza funkcję dopasowania każdej produkcji. Nowe produkcje gramatyki są odkrywane podczas procesu indukcji przez mechanizm pokrycia oraz algorytm genetyczny. W pracy można wyodrębnić dwie części.

Pierwsza część pracy wprowadza w tematykę wnioskowania gramatycznego, ewolucyjnego przetwarzania oraz uczących się systemów klasyfikujących. W szczególności zaprezentowano aktualny stań badań w zakresie indukcji gramatyki bezkontekstowej, nowy sposób kategoryzacji uczących się systemów klasyfikujących oraz ich podstawowe modele w jednolitym ujęciu.

W drugiej części pracy zaproponowano oryginalny model ewolucyjnego wnioskowania gramatycznego, dedykowany indukcji gramatyki bezkontekstowej. Architekturę i działanie nowego modelu opisano, posługując się kategoriami uczącego się systemu klasyfikującego. Wprowadzono tzw. mechanizm płodności produkcji, który wraz z mechanizmem ścisku oraz operatorem genetycznym inwersji ma przeciwdziałać wysokiej epistazie populacji produkcji modelu. Zdefiniowano nowe operatory pokrycia dostosowane do użytej metody parsowania oraz estymatory dokładności i kosztu indukcji. Przeprowadzono indukcję języków regularnych z tzw. zbioru Tomity, wybranych formalnych języków bezkontekstowych, a także obszernych korpusów językowych. Eksperymenty wykazały, że model uzyskuje dla każdej z badanych klas języka wyniki porównywalne z najlepszymi ze znanych w literaturze przedmiotu, i to nie tylko wśród metod ewolucyjnych,

* Instytut Informatyki, Automatyki i Robotyki Politechniki Wrocławskiej.

a w wielu wypadkach lepsze. Przeprowadzono badania symulacyjne modelu, których celem było eksperymentalne stwierdzenie własności proponowanego modelu ewolucyjnego. Poza wnioskami szczegółowymi osiągnięto również interesujące wyniki dotyczące ogólnych mechanizmów ewolucyjnych, jak wpływ selekcji turniejowej i ścisku na nacisk selektywny czy rola nowego operatora pokrycia pełnego w procesie ewolucji populacji uczącego się systemu klasyfikującego. Wskazano na jedno z możliwych praktycznych zastosowań modelu, poza badanym już w monografii obszarem inżynierii lingwistycznej, jakim jest genomika obliczeniowa. Rozpatrywano zadanie rozpoznawania sekwencji telomerowej u człowieka oraz poszukiwania regi onu promotorowego u bakterii *E. coli*. Model w obecnej implementacji może być zastosowany na wysokim poziomie estymatora *swoistości* do rozpoznawania regionów nienależących do sekwencji promotorowych.

Wstęp

Uczenie maszynowe (*machine learning*) obejmuje problematykę konstruowania programów komputerowych potrafiących pozyskiwać, na podstawie wprowadzonej informacji, nową wiedzę lub poprawiać wiedzę już posiadaną (Michalewicz 1996). W ostatnich latach zaobserwować można szczególnie intensywny rozwój tej dziedziny i to zarówno w sferze badań podstawowych, jak i zastosowań. Algorytmy uczenia maszynowego są podstawowymi metodami inżynierii wiedzy, systemów doradczych, eksploracji danych, inteligentnych systemów wyszukiwania i filtrowania informacji, przetwarzania obrazów i języka naturalnego, robotyki, a od niedawna również bioinformatyki. Jako dziedzina interdyscyplinarna uczenie maszynowe opiera się na koncepcjach i rezultatach m.in. statystyki, sztucznej inteligencji, filozofii, teorii informacji, biologii, psychologii, teorii decyzji, teorii złożoności obliczeniowej oraz informatyki.

Jedną z żywo rozwijających się metod uczenia maszynowego jest wnioskowanie gramatyczne (*grammar induction*), które podejmuje problematykę uczenia języka (a precyzyjniej gramatyki lub równoważnego automatu) na podstawie przykładowych zdań. Od algorytmu uczącego oczekuje się generalizacji polegającej na umiejętności generacji i akceptacji zdań wychodzących poza zbiór uczący. Problem indukcji gramatyki jest zdefiniowany przez:

- klasę indukowanego języka,
- dostępność danych uczących, które mogą należeć do języka (przykłady pozytywne), do języka nie należą (przykłady negatywne) lub też dostarczają dodatkowych informacji,
- wreszcie przez rozmiar danych uczących.

Zgodnie z twierdzeniem Golda (1967) niemożliwa jest indukcja dowolnego języka z hierarchii Chomsky'ego jedynie na podstawie przykładów poprawnych, a uzupełnienie zbioru uczącego o zdania negatywne pozwoliło dotychczas na znalezienie efektywnych, tj. działających w czasie wielomianowym, algorytmów uczących jedynie dla wyrażeń regularnych i równoważnych im automatów skończonych (*deterministic finite automaton*, DFA).

Dla klasy języków bezkontekstowych nie są znane efektywne algorytmy wnioskowania. Uzupełnianie zbioru uczącego o odpowiednie zapytania również nie gwarantuje identyfikacji gramatyki bezkontekstowej w czasie wielomianowym. Algorytmom

uczącym dostarcza się dodatkowej informacji w postaci przykładów spoza indukowanego języka (negatywnych), zbiory uczące niosą ze sobą dodatkową informację strukturalną, formułuje się alternatywne reprezentacje gramatyk bezkontekstowych, ogranicza się zadanie uczenia do pewnych podklas języka, wreszcie stosuje się metody bayesowskie. Jednak pytanie o wielomianową złożoność algorytmów indukcji gramatyk bezkontekstowych jest wciąż bez odpowiedzi (Angluin 2001). Jednocześnie indukcja gramatyk bezkontekstowych to ważne zagadnienie ze względów praktycznych, bowiem gramatyka bezkontekstowa może modelować nie tylko strukturę języków programowania, języków naturalnych, ale także danych biologicznych. Uczenie gramatyk bezkontekstowych znajduje zastosowanie m.in. w rozpoznawaniu wzorców (*pattern recognition*) oraz w rozpoznawaniu mowy (*speech recognition*). Algorytmy uczenia języka lub równoważnej gramatyki pozwalają też modelować sposób, w jaki człowiek uczy się języka (*language acquisition*). Budowa algorytmów uczących się gramatyk bezkontekstowych jest zatem dzisiaj jednym z otwartych i zarazem krytycznych problemów wnioskowania gramatycznego (de la Higuera 2000).

Cel, jaki stawia przed sobą prezentowana monografia, jest dwojaki. Po pierwsze, w monografii zaproponowano i eksperymentalnie przebadano nowy model ewolucyjny, którego zasadniczym przeznaczeniem jest indukcja gramatyki bezkontekstowej z zastosowaniem wnioskowania gramatycznego. Konstrukcja nowego modelu ewolucyjnego, nazwanego GCS (*Grammar-based Classifier System*), wykorzystuje mechanizm uczenia stosowany w uczących się systemach klasyfikujących (*learning classifier systems*). Idea uczenia uczących się systemów klasyfikujących, po raz pierwszy podana przez Hollanda i zastosowana w (Holland i Reitman 1978), opiera się na prostej zasadzie adaptacji reguł określających działanie modelu do otoczenia, w którym model działa. Adaptacja modelu polega na sprawdzaniu i poprawie efektywności istniejących reguł oraz generowaniu nowych reguł, dostosowanych do otoczenia. W modelu GCS regułami (klasyfikatorami) są produkcje gramatyki bezkontekstowej podane w postaci normalnej Chomsky'ego, natomiast otoczeniem zbiór uczący składający się z przykładowych etykietowanych zdań. Etykieta zdania określa jego przynależność lub brak przynależności do poszukiwanego języka. Celem uczenia modelu jest poprawna klasyfikacja zdań uczących. Ponieważ zbiór klasyfikatorów tworzy zestaw produkcji gramatyki, więc poprawna klasyfikacja etykietowanych zdań oznacza wyindukowanie poszukiwanej gramatyki języka. Model GCS śledzi produkcje użyte podczas analizy (parsowania) zbioru uczącego i po jej zakończeniu oblicza funkcję dopasowania każdej produkcji. Nowe produkcje gramatyki są odkrywane przez mechanizm pokrycia (*covering*) oraz algorytm genetyczny. Algorytm genetyczny, wykorzystując wybraną metodę selekcji, ścisku oraz wartości dopasowania produkcji, poszukuje efektywniejszej populacji klasyfikatorów po zakończeniu pełnej analizy zbioru uczącego. Mechanizm pokrycia, w przeciwieństwie do algorytmu genetycznego, działa w trakcie parsowania zdania uczącego. Dodaje on do populacji klasyfikatorów takie produkcje, które w danej sytuacji

środowiskowej umożliwiają dalszy rozbiór zdania. Rozbiór zdań tworzących otoczenie modelu realizowany jest przez algorytm CYK[1].

Drugim celem monografii jest próba wypełnienia luki w polskim piśmiennictwie na temat wnioskowania gramatycznego[2], w tym wnioskowania gramatycznego korzystającego z metod ewolucyjnych. Nie ma obecnie, według wiedzy autora, żadnego zwartego opracowania w języku polskim na ten temat, nie ma artykułów, które by poruszały tę problematykę[3], a internetowa przeglądarka *Google* znajduje na polskich stronach nieliczne odwołania do terminu *grammar induction*. Wydaje się zasadnym, by tak ważna i płodna teoretycznie oraz praktycznie metoda uczenia maszynowego doczekała się odrębnego opracowania w języku polskim.

Monografia podzielona jest na 6 głównych rozdziałów, poprzedzonych wstępem i zakończonych podsumowaniem oraz 3 załącznikami.

W rozdziale 1 wprowadzono w tematykę wnioskowania gramatycznego. Po zdefiniowaniu zadania, jakie stawia przed sobą wnioskowanie gramatyczne, oraz pewnych charakterystycznych dla tej metody uczenia maszynowego własności podano obszary zastosowań z licznymi referencjami bibliograficznymi. Następnie, po uprzednim przytoczeniu podstawowych pojęć z teorii automatów i języków, scharakteryzowano paradygmaty uczenia stosowane przez wnioskowanie gramatyczne. W zależności od przyjętego modelu uczenia formułuje się nie tylko różne własności procesu wnioskowania, ale również różne algorytmy uczące. Omówiono model identyfikacji w granicy, model PAC, zasadę minimalnej długości kodu i uczenie na podstawie zapytań. W podrozdziale 1.5 zebrano proponowane w literaturze podejścia do problemu indukcji gramatyki bezkontekstowej. Omówiono indukcję na podstawie nieetykietowanego tekstu, indukcję wspomaganą dodatkową informacją w postaci pewnych danych strukturalnych, indukcję pewnych podklas języków bezkontekstowych niezawierających wszystkich skończonych języków, indukcję alternatywnych reprezentacji gramatyk, obecnie bardzo często spotykaną w literaturze przedmiotu, głównie za sprawą udowodnionych interesujących własności uczenia – indukcję stochastycznych gramatyk i wreszcie indukcję z zastosowaniem metod sztucznej inteligencji. Ten ostatni typ wnioskowania został omówiony najobszerniej, ze szczególnym uwzględnieniem stosowanych metod ewolucyjnych. W podrozdziale 1.5.6 przedstawiono również proponowane przez autora niniejszej monografii oryginalne podejścia do omawianej problematyki, w których zastosowano automat ze stosem, także w wersji rozmytej, ewoluowany przy zastosowaniu różnych typów kodowania pośredniego, tablicową reprezentację produkcji gramatyki czy też kanoniczny uczący się system klasyfikacyjny. Doświadczenia uzyskane podczas stosowania różnorodnych metod i reprezentacji

[1] Algorytm CYK jest szerzej omówiony w podrozdz. 1.5.7.

[2] Wnioskowanie gramatyczne nie ma ustabilizowanej terminologii polskiej. W (Przepiórkowski i in. 2003) termin *grammar induction* przetłumaczony został na *automatyczną indukcję gramatyk*.

[3] Poza tekstami autora.

zaowocowały koncepcją nowego modelu ewolucyjnego GCS bazującego na ogólnej architekturze uczącego się systemu klasyfikującego. Model GCS jest przedmiotem trzech rozdziałów monografii. Rozdział 1 zamyka przedstawienie działającego w czasie sześciennym i wykorzystywanego przez model GCS algorytmu CYK, który bada przynależność łańcucha do języka bezkontekstowego.

W rozdziale 2 krótko scharakteryzowano ewolucyjne algorytmy, dzieląc je na programowanie ewolucyjne, strategie ewolucyjne, algorytmy genetyczne i programowanie genetyczne.

Przedmiotem rozdziału 3 są uczące się systemy klasyfikujące, zaliczane przez Goldberga (1989) do genetycznych systemów uczących się (*genetic-based machine learning systems*), a Michalewicza (1996) do genetycznych metod uczenia maszynowego. W rozdziale przedstawiono generyczną architekturę takiego systemu oraz zaproponowano własną kategoryzację istniejących implementacji (poza klasycznym podziałem na systemy stosujące podejście Michigan oraz Pitt) wg kryterium zasięgu chromosomu, miary użyteczności i metody reprezentacji klasyfikatora, stosowanego w module odkrywczym mechanizmu i wreszcie pamięci systemu. Każda kategoria została poddana dyskusji i opatrzona obszerną bibliografią. Rozdział kończy się szczegółowym opisem najczęściej cytowanych w literaturze przedmiotu modeli systemów klasyfikujących: LCS, ZCS, XCS oraz ACS wraz z obszarami zastosowań istniejących realizacji systemów.

W kolejnych trzech rozdziałach monografii przedstawiono oryginalny model GCS, stanowiący propozycję nowej metody ewolucyjnego wnioskowania gramatycznego. W rozdziale 4 przedstawiono zasadnicze elementy nowego modelu oraz opis jego komputerowej realizacji. Rozdział rozpoczyna formalnie zdefiniowane zadanie klasyfikacji, jakie stawiane jest przed modelem pracującym w środowisku zdań uczących. Następnie, tytułem wprowadzenia w problematykę, zamieszczony został nieformalny opis modelu, po którym omówiono podstawowe elementy składające się na model. Zdefiniowano formalnie klasyfikator modelu, podano wzory na jego przystosowanie oraz zinterpretowano występujące w definicji klasyfikatora parametry jego tzw. płodności. Przedstawiono w ujęciu formalnym ewoluowaną przez model gramatykę oraz zdefiniowano architekturę modelu w kategoriach uczącego się modelu klasyfikującego. Omówiono podstawowe mechanizmy modelu odpowiedzialne za uczenie, tj. metody pokrycia, ścisk oraz algorytm genetyczny wraz z metodami selekcji i operatorami genetycznymi. Prezentację składowych modelu kończy przedstawienie korekcji zbioru produkcji oraz zestawu parametrów modelu. Opis modelu GCS zamyka jego algorytmiczny zapis wraz z krótką prezentacją komputerowej realizacji.

W rozdziale 5 monografii zawarto wyniki szeroko przeprowadzonych badań symulacyjnych modelu, celem których miało być nie tylko eksperymentalne zbadanie własności modelu GCS, ale również porównanie możliwości modelu z innymi algorytmami. We wstępnej części rozdziału zdefiniowano estymatory symulacji, za pomocą których oceniano dokładność i koszt indukcji oraz dokładność generalizacji zbioru

testowego. Następnie scharakteryzowano zastosowane podczas eksperymentów zbiory uczące i testowe. Model GCS indukował gramatyki dla wyrażeń regularnych należących do tzw. zbioru Tomity, dla wybranych, najczęściej stosowanych w literaturze przedmiotu, formalnych języków bezkontekstowych oraz obszernych korpusów językowych. Wszystkie symulacje przeprowadzano dla tego samego, domyślnego zbioru parametrów modelu, który – czego dowiodły późniejsze empiryczne badania własności modelu – nie był zbiorem parametrów o optymalnych wartościach dla żadnego z indukowanych języków. Takie podejście pozwala na ominięcie oddzielnego problemu, jakim jest poszukiwanie zestawu optymalnych parametrów algorytmu ewolucyjnego przy jednoczesnym założeniu, że badany algorytm działa wystarczająco dobrze w określonym zakresie wartości poszczególnych parametrów. Eksperymenty wykazały, że model GCS uzyskuje dla każdej z badanych klas języka, tj. regularnej i bezkontekstowej, wyniki porównywalne z najlepszymi ze znanych w literaturze przedmiotu, i to nie tylko wśród metod ewolucyjnych, a w wielu wypadkach lepsze. W podrozdz. 5.6 opisano przeprowadzone badania symulacyjne modelu, których celem było eksperymentalne stwierdzenie własności proponowanego modelu ewolucyjnego. Poza wnioskami szczegółowymi osiągnięto również interesujące wyniki dotyczące ogólnych mechanizmów ewolucyjnych, jak wpływ selekcji turniejowej i ścisku na nacisk selektywny czy rola nowego operatora pokrycia pełnego w procesie ewolucji populacji uczącego się systemu klasyfikującego.

W rozdziale 6 wskazano na jedno z możliwych praktycznych zastosowań modelu GCS, chociaż już indukcja korpusów językowych, opisana w rozdz. 5, jest przykładem udanej implementacji modelu w obszarze inżynierii lingwistycznej. Zbadano również możliwości użycia modelu w genomice obliczeniowej. Rozpatrywane było zadanie rozpoznawania sekwencji telomerowej u człowieka oraz poszukiwania regionu promotorowego u bakterii *E. coli*.

W rozdziale 7 monografii podsumowano uzyskane wyniki oraz zakreślono dalsze plany badawcze.

1. Wnioskowanie gramatyczne

1.1. Wprowadzenie

Wnioskowanie gramatyczne (*grammar induction*, *automata induction*, *grammatical inference*, GI) (Gold 1967, Pinker 1979, Angluin i Smith 1983, Fu i Boot 1986) to według definicji Vasanta Honovara umieszczonej na stronach internetowych organizacji ICGI[4] (*International Community of Grammatical Induction*) *uczenie gramatyk*[5] *i języków na podstawie przykładowych danych*. Maszynowe uczenie gramatyk znajduje zastosowanie w takich dziedzinach nauki, jak:

- uczenie maszynowe (*machine learning*),
- syntaktyczne rozpoznawanie wzorców (*syntactic pattern recognition*),
- teoria automatów i języków formalnych (*automata and formal language theory*),
- lingwistyka obliczeniowa (*computational linguistics*),
- biologia obliczeniowa (*computational biology*),
- rozpoznawanie mowy (*speech recognition*),
- przetwarzanie języka naturalnego (*natural language processing*),
- drążenie danych (*data mining*)[6].

Pomimo tak szerokiego zastosowania wnioskowania gramatycznego, nie ma w literaturze zwartego, książkowego opracowania zagadnienia, są natomiast dostępne mniej lub bardziej obszerne wprowadzenia lub przeglądy (Lee 1996, Sakakibara 1997, de la Higuera 2000, Honovar i de la Higuera 2001).

Przykładowymi danymi, na podstawie których uczący, nazywany też maszyną wnioskującą (*inference machine*) lub algorytmem uczącym (*learning algorithm*), próbuje zbudować model, mogą być dane sekwencyjne lub strukturalne, tj. ciągi znaków (*string*), słowa, drzewa, grafy. Indukowana (*induced*, *inferred*) gramatyka może być

---

[4] http://eurise.univ-st-etienne.fr/gi/

[5] lub ekwiwalentnych automatów.

[6] Pojęcie *data mining*, oznaczające automatyczną analizę (eksplorację) danych w celu wykrycia istotnych motywów i wzorców, w języku polskim tłumaczy się na *drążenie danych*, *eksploracja danych*, *zgłębianie danych*, *dogłębna analiza danych*, *odkrywanie wiedzy* lub... nie tłumaczy się w ogóle.

następnie zastosowana do klasyfikacji danych wcześniej nieobserwowanych przez uczącego, do kompresji danych lub modelowania tychże danych.

Cechami charakterystycznymi wnioskowania gramatycznego są:

- dane wejściowe – zwykle utworzone na skończonym alfabecie, dyskretne i o nieograniczonym rozmiarze,
- wyniki – wynikiem jest zwykle gramatyka lub automat, których główną cechą jest możliwość interpretacji przez człowieka,
- złożoność – nawet proste problemy klasyfikowane są jako trudne obliczeniowo,
- różnorodność zastosowań,
- ciągle niewielka liczba zakończonych sukcesem wdrożeń przemysłowych.

1.2. Obszary zastosowań

Szukanie struktur, wzorców, regularności, gramatyk czy może automatów jedynie na podstawie dostępnych danych wejściowych stanowi problem wielu, często odległych, obszarów nauki. W podrozdziale przedstawiony zostanie przegląd najważniejszych zastosowań wnioskowania gramatycznego.

Robotyka i sterowanie

W pracach (Dean i in. 1992, Rivest i Schapire 1993, Kungas 2001) użyto GI w nawigacji mobilnych robotów, a Luzeaux (1996) zastosował ten typ wnioskowania w modelu inteligentnego sterowania. Bardziej teoretyczne prace dotyczą modelowania tzw. systemów krytycznych z użyciem automatów Büchiego[7] (Verdi i Wolper 1986, Saoudi i Yokomori 1993, de la Higuera i Janodet 2001).

Syntaktyczne rozpoznawanie wzorców

Syntaktyczne rozpoznawanie wzorców to jedno z pierwszych zastosowań wnioskowania gramatycznego w latach siedemdziesiątych ubiegłego wieku. Z tego też okresu pochodzi stosunkowo obszerna literatura. Z nowszych prac wymienić można (Lucas i in. 1994) opisującą uczenie konturów obrazów czy też (Ron i in. 1995) podejmującą zadanie rozpoznawania znaków. W (Ney 1992) znajduje się przegląd literatury.

Lingwistyka obliczeniowa

Wnioskowanie gramatyk regularnych (*regular grammar inference*) lub równoważnych im automatów skończonych stanowi przedmiot badań od ponad trzech dekad (Trakhtenbrot i Barzdin 1973, Gold 1978, Angluin 1981, Valiant 1984, Angluin 1987a, Schapire 1990, Oncina i Garcia 1992, Dupont 1996, Sakakibara 1997). Dobrym podsumowaniem stanu badań w tej dziedzinie jest książka (Honovar i de la Higuera 2001).

[7] Büchi J.R. (1960), *On a decision method in restricted second order arithmetic*, Proc. Conf. Logic Method and Philos. of Sci., California, Stanford Univ. Press.

Języki regularne są największą klasą języków, dla której można znaleźć efektywne algorytmy uczenia. Dla gramatyk bezkontekstowych oraz automatów ze stosem jak dotąd nie znaleziono algorytmów wielomianowych, który to fakt stymuluje coraz to nowe podejścia (Dupont 1994, Huijsen 1993, Kammeyer i Belew 1996, Keller i Lutz 1997, Keller i Lutz 2005, Lucas 1994, Wyard 1991, Zhou and Grefenstette 1986, Charniak 1993, Korkmaz i Ucoluk 2001, Lankhorst 1995, Smith i Witten 1996, Unold 2003, Unold 2005b, Unold 2005e).

Innym obszarem zainteresowania lingwistyki jest uczenie automatów tłumaczących (*transducers*) (Mohri 1997, Mohri 2000, Oncina i in. 1993, Casacuberta 1995).

Przetwarzanie języka naturalnego

Rozróżnienie pomiędzy pracami z zakresu lingwistyki obliczeniowej a przetwarzania języka naturalnego jest w wielu wypadkach umowne. Takie same mechanizmy można bowiem częstokroć stosować zarówno dla języków sztucznych, jak i naturalnych. Zdecydowanie wyraźniejszy podział w algorytmach stosowanych dla języków naturalnych przebiega pomiędzy uczeniem na oznakowanym lingwistycznie repozytorium językowym a uczeniem na tekście w żaden sposób nieoznaczonym (*raw corpora*)[8].

W pierwszym typie uczenia materiałem źródłowym są korpusy językowe, których uprzednio lingwistycznie nie znakowano (*tagged*) (Pereira i Schabes 1992, Brill 1993, Stolcke and Omohundro 1994, Henrichsen 2002, Watkinson i Manandhar 2001, Adriaans 1999, van den Bosch 1999, Domingos 1995, Kirby 2002, Briscoe 2000, van Zaanen 2002, Paskin 2001, Roberts 2002, Solan i in. 2005).

Drugi typ uczenia na podstawie tekstu języka naturalnego indukuje gramatyki, bazując na oznakowanym morfosyntaktycznie (*part-of-speech tagged*, POS), a w większości wypadków również i syntaktycznie, korpusie językowym (Magermann 1995, Lin 1995, Collins 1999, Hwa 1999, Collins i Duffy 2002, Charniak 2000, Cyre 2002, Klein i Manning 2003, Aycinena i in. 2003, Unold 1998d, Unold 1999a, Unold 1999b, Unold 1999c, Unold 1999d, Unold 1999e, Unold 2000, Chrobak i Unold 2000a, Chrobak i Unold 2000b, Chrobak i Unold 2001, Unold i Dulewicz 2002, Dulewicz i Unold 2002, Unold 2003, Dulewicz i Unold 2004).

Innym przykładem zastosowania wnioskowania gramatycznego w tzw. „wysokopoziomowym" przetwarzaniu języka naturalnego są systemy tłumaczące z języka na język (Amengual i in. 2001, Vidal 1997).

[8] W indukcji gramatycznej odbywającej się na podstawie korpusów językowych spotyka się również klasyfikację rozróżniającą: uczenie z nadzorem (*supervised*), w którym korpusy językowe są oznakowane zarówno syntaktycznie, jak i morfosyntaktycznie; uczenie z połowicznym nadzorem (*semisupervised*), w którym korpusy mają oznaczone nawiasami grupy syntaktyczne; oraz uczenie bez nadzoru (*unsupervised*), w którym korpusy oznakowane są jedynie morfosyntaktycznie (Thanaruk i Omkumary 1995). Ten ostatni rodzaj uczenia obejmuje również uczenie na nieoznakowanym korpusie językowym (Solan 2005). Inni (Powers 1997, Clark 2001a, Klein i Manning 2003) definiują jedynie uczenie z nadzorem i bez nadzoru.

Od 10 już lat trwają próby zastosowania indukowanych modeli automatowych w rozpoznawaniu mowy (Garcia i in. 1994, Thollard i in. 2000, Thollard 2001), stosuje się również modele *n-gram* (Jelinek 1998) oraz ukryte modele Markowa (Morgan i Bourlard 1995, Picone 1990).

Niskopoziomowym – podobnie jak rozpoznawanie mowy – przetwarzaniem języka naturalnego jest uczenie automatów tłumaczących dla potrzeb fonetyki i morfosyntaktyki (Gildea i Jurafsky 1996, Mohri 1997, Oflazer 1996, Roche i Schabes 1995).

Zarządzanie dokumentami

Dokumenty możemy rozpatrywać jako obiekty o własnej, często nieznanej, a poszukiwanej strukturze. Klasycznym przykładem są słowniki (Gonnet i Tompa 1987, Meijs 1993, Atwell i in. 1993, Ahonen i in. 1994), poszukuje się również gramatyk dowolnych znakowanych dokumentów (Young-Lai 1996, Young-Lai i Tompa 2000, Kosala i in. 2002). Pojawienie się standardu XML zaowocowało ożywieniem prowadzonych w tym zakresie badań (Fernau 2000, Arimura i in. 2001, Chidlovskii 2002, Hong 2003).

Indukcyjne programowanie logiczne

Indukcyjne programowanie logiczne (*inductive logic programming*, ILP) jest przykładem uczenia maszynowego (Muggleton 1999, Cichosz 2000), które w uczeniu predykatów wykorzystuje również zbiory uczące. W systemie MERLIN, łączącym ILP z wnioskowaniem gramatycznym, indukuje się deterministyczne (Boström 1996) oraz stochastyczne automaty (Boström 1998). System GIFT uczy się automatów drzewowych (*tree automata*) (Bernard i Habrard 2001).

Bioinformatyka

Biosekwencje mogą być traktowane jako zdania języka o określonej gramatyce. Poznanie struktury języka umożliwia klasyfikację badanych po raz pierwszy sekwencji biologicznych pomiędzy należące i nienależące do określonej grupy. W (Wang i in. 1999) klasyfikowano ciągi DNA, ciągi tRNA opisywane przez stochastyczne gramatyki bezkontekstowe były rozpoznawane w (Sakakibara i in. 1994). Gramatyki bezkontekstowe można również zastosować w analizie drugorzędnej struktury proteinowej (Abe i Mamitsuka 1997), a w połączeniu z bigramami (*bi-gram*) w rozpoznawaniu tzw. struktur izolowanych (Salvador i Benedi 2002). Ewoluowane maszyny Turinga stosowano również do rozpoznawania sekwencji wirusa HIV (Vallejo i Ramos 2001).

Innym stosowanym modelem, opisującym struktury proteinowe, są ukryte modele Markowa (Lyngsø i in. 1999, Lyngsø i Pedersen 2001, Jagota i in. 2001).

Systemy agentowe

Jeżeli zakodujemy zachowanie inteligentnego agenta w postaci deterministycznego automatu, to można wyuczyć strategii jego zachowania (czyli struktury automatu) na podstawie przeprowadzonych gier w środowisku wieloagentowym. Takie podejście prezentują prace (Carmel i Markowitch 1998, 1999).

Kompresja

Algorytm SEQUITUR (Nevill-Manning i Witten 1997) jest w stanie wyuczyć się gramatyki bezkontekstowej na podstawie jednego, zwykle długiego, ciągu znaków. Tak rozpoznana gramatyka może odtworzyć później wejściowy ciąg znaków.

Dobre współczynniki kompresji na strukturach drzewiastych (na przykład plikach XML) uzyskano, ewoluując *k*-testowalne automaty drzewiaste (*k-testable tree automaton*) (Rico-Juan i in. 2002).

Muzyka

Indukowane, na podstawie serii przykładów, automaty stochastyczne mogą modelować style muzyczne, a nawet służyć do generacji melodii (Cruz i Vidal 1998).

Wspomniany już wcześniej SEQUITUR (Nevill-Manning i Witten 1997) zastosowano do wykrycia repetycji w chorałach Bacha.

Szeregi czasowe

W pracy (Giles i in. 2001) zastosowano uczone stochastyczne automaty skończone jako predykatory ceny kursu dolara na giełdzie.

1.3. Wybrane pojęcia z teorii automatów i języków

W rozdziale podane zostaną niezbędne dla zrozumienia dalszej części monografii pojęcia formułowane w teorii automatów i języków. Kompendium wiedzy z tego zakresu można znaleźć m.in. w pracach (Hopcroft i Ullman 1979, Mikołajczak 1985).

Definicja 1

Zbiór V nazywamy alfabetem, jeśli jest on niepusty i skończony. Elementy tego zbioru nazywamy symbolami.

Przykładowo, zbiór złożony z dwóch symboli 0 oraz 1 definiuje tzw. alfabet binarny $V = \{0, 1\}$.

Definicja 2

Słowem (zdaniem albo łańcuchem) nad alfabetem V nazywamy każdy skończony ciąg symboli z V. Symbolem V^* oznaczamy zbiór wszystkich słów zbioru V. Przez ε oznaczamy słowo puste, czyli słowo niezawierające żadnego symbolu.

Łańcuch *aaba* jest przykładowym słowem nad alfabetem $V = \{a, b\}$.

Definicja 3

Złożeniem (konkatenacją) zbiorów V i W nazywamy zbiór

$$VW = \{vw \mid v \in V, w \in W\}. \tag{1}$$

Definicja 4

Potęgę alfabetu V definiujemy następująco:

$$V^{(0)} = \{\varepsilon\}, V^{(k)} = VV^{(k-1)} \text{ dla } k \geq 1. \tag{2}$$

Przyjmujemy $V^+ = V^* \setminus \{\varepsilon\}$.

Przykładowo, jeżeli $V = \{a, b\}$, to $V^{(1)} = \{a, b\}$, $V^{(2)} = \{aa, ab, ba, bb\}$. Warto przy okazji zauważyć za (Hopcroft i Ullman 1979), że o ile V jest zbiorem symboli a oraz b, to $V^{(1)}$ jest już zbiorem łańcuchów, każdy o długości 1.

Definicja 5

Długością słowa w, oznaczaną przez $|w|$, nazywamy liczbę symboli, z których składa się słowo w, np. $|\varepsilon| = 0, |aa| = 2$.

Definicja 6

Językiem L nad alfabetem V nazywamy dowolny zbiór słów nad tym alfabetem,

$$L \subseteq V^*. \tag{3}$$

Przykładowym językiem nad alfabetem $V = \{a, b\}$ jest zbiór słów zawierający parzystą liczbę symboli a, $L = \{aa, aab, bbaabaa, baaaa, ababbbabba, \ldots\}$. Warto zauważyć, że zbiór pusty jest językiem nad dowolnym alfabetem.

Definicja 7

Niech L_1, L_2 będą językami nad alfabetem V. Konkatenacją języków L_1 i L_2 nazywamy zbiór

$$L_1 L_2 = \{uv \mid u \in L_1, v \in L_2\}. \tag{4}$$

Przykładowo, konkatenacją języków $L_1 = \{aa, bb\}$ i $L_2 = \{bb, aa\}$ jest język $L_1L_2 = \{aabb, aaaa, bbbb, bbaa\}$.

Definicja 8

Domknięciem Kleene'go języka L nazywamy zbiór

$$L^* = \bigcup_{i=0}^{\infty} L^i, \tag{5}$$

gdzie

$$L^0 = \{\varepsilon\}, L^i = L^{i-1}L, i > 0. \tag{6}$$

Definicja 9

Gramatyką formalną nazywamy czwórkę

$$G = (N, T, P, S), \tag{7}$$

w której:

N – zbiór skończony zwany zbiorem symboli pomocniczych (nieterminalnych),

T – zbiór skończony zwanym zbiorem symboli końcowych (terminalnych), $N \cap T = 0$,

$P \subseteq (N \cup T)^+ \times (N \cup T)^*$ jest relacją skończoną zwaną listą produkcji,

S jest wyróżnionym symbolem pomocniczym zwanym symbolem początkowym.

Jeśli $(p, q) \in P$, to będziemy stosować także zapis $p \rightarrow q$ i mówić, że słowo q jest bezpośrednio wyprowadzane ze słowa p. Przyjmuje się również następujące uproszczenie w zapisie produkcji: jeśli $(x, y), (x, z) \in P$, to piszemy $x \rightarrow y \mid z$.

Jeśli istnieje ciąg słów $p_1, p_2, ..., p_n$ taki, że $p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_n$, to mówimy, że p_n jest wyprowadzane z p_1 lub że istnieje wywód słowa p_n ze słowa p_1 i piszemy $p_1 \xrightarrow{*} p_n$.

Definicja 10

Język $L(G)$ generowany przez gramatykę formalną G jest zbiorem ciągów symboli końcowych wyprowadzalnych z S:

$$L(G) = \{x \mid x \in T^* \wedge S \xrightarrow{*} x\}. \tag{8}$$

Jeśli $x \in L(G)$, to mówimy, że x jest ciągiem (słowem, zdaniem) generowanym przez gramatykę G.

Definicja 11

Dwie gramatyki G_1 i G_2 nazywamy równoważnymi, jeśli generują ten sam język, tzn. $L(G_1) = L(G_2)$.

Nakładając stopniowo coraz większe ograniczenia na postać produkcji, możemy zdefiniować cztery typy gramatyki, a co za tym idzie cztery klasy języków formalnych (Chomsky 1956, 1959).

Definicja 12

Jeśli na gramatykę nie nakładamy żadnych ograniczeń, to mamy do czynienia z najogólniejszą klasą gramatyk formalnych G^0 zwanych gramatyką typu 0, a języki przez nie generowane, językami klasy 0 (rekurencyjnie przeliczalnymi).

Definicja 13

Gramatykę $G^1 = (N, T, P, S)$ taką, że jej produkcje są postaci

$$x\alpha y \rightarrow xzy, \tag{9}$$

przy czym $x, y, z \in (N \cup T)^*, \alpha \in N,$ nazywamy gramatyką typu 1 (kontekstową). Jeśli G jest gramatyką kontekstową, to $L(G)$ nazywa się językiem kontekstowym (klasy 1).

Definicja 14

Gramatyka $G^2 = (N, T, P, S)$ jest gramatyką typu 2 (bezkontekstową, *context-free grammar*, CFG), jeśli wszystkie jej produkcje są postaci

$$\alpha \rightarrow x, \tag{10}$$

przy czym $\alpha \in N, x \in (N \cup T)^*$. Język generowany przez gramatykę typu 2 nazywa się językiem klasy 2 (bezkontekstowym).

Definicja 15

Gramatykę bezkontekstową $G^2 = (N, T, P, S)$ nazywamy prawostronnie liniową, jeśli każda jej produkcja jest postaci

$$\alpha \rightarrow a \text{ lub } \alpha \rightarrow a\beta, \tag{11}$$

przy czym $\alpha, \beta \in N, a \in T^*$.

Definicja 16

Gramatykę bezkontekstową $G^2 = (N, T, P, S)$ nazywamy lewostronnie liniową, jeśli każda jej produkcja jest postaci

$$\alpha \rightarrow a \text{ lub } \alpha \rightarrow \beta a, \tag{12}$$

przy czym $\alpha, \beta \in N, a \in T^*$.

Definicja 17

Gramatyki prawostronnie liniowe i lewostronnie liniowe nazywają się gramatykami regularnymi G^3 (typu 3). Język generowany przez gramatykę typu 3 nazywa się językiem klasy 3 (regularnym).

Warto przy okazji zauważyć, że $G^3 \subset G^2 \subset G^1 \subset G^0$, tj. gramatyka klasy i jest jednocześnie gramatyką klasy j, dla $0 \leq j \leq i$. Odwrotne twierdzenie nie jest prawdziwe.

Definicja 18

Gramatyka bezkontekstowa $G = (N, T, P, S)$ jest w postaci normalnej Chomsky'ego (PNC), jeśli każda produkcja ze zbioru P jest jednej z trzech postaci

$$S \rightarrow \varepsilon \mid A \rightarrow a \mid A \rightarrow BC, \tag{13}$$

gdzie $A \in N, a \in T, B, C \in N \setminus \{S\}$.

Definicja 19

Gramatyka bezkontekstowa $G = (N, T, P, S)$ jest w postaci normalnej Greibach[9] (PNG), jeśli każda produkcja ze zbioru P jest jednej z trzech postaci

$$S \rightarrow \varepsilon \mid A \rightarrow a \mid A \rightarrow aA_1A_2 \ldots A_n, \tag{14}$$

gdzie $A \in N, a \in T, A_1, A_2, ..., A_n \in N \setminus \{S\}$.

[9] W polskiej literaturze przedmiotu można spotkać się z zapisem *postać normalna Greibacha*, ale nie jest on poprawny ze względu na Sheilę Greibach, od nazwiska której wywodzi się nazwa PNG (patrz http://pl.wikipedia.org/wiki/Postać_normalna_Greibach).

Twierdzenie 1

Dla każdej gramatyki bezkontekstowej istnieje równoważna gramatyka bezkontekstowa w postaci normalnej Chomsky'ego oraz gramatyka bezkontekstowa w postaci normalnej Greibach.

1.4. Paradygmaty uczenia

Uczenie się jest wnioskowaniem indukcyjnym[10]. W przypadku uczenia języków (inaczej mówiąc wnioskowania gramatycznego) algorytmowi uczącemu się prezentowane są dane, na podstawie których algorytm musi wywnioskować reguły gramatyczne generujące obserwowany język. Co więcej, oczekuje się od algorytmu generalizacji polegającej na umiejętności generacji i akceptacji zdań wychodzących poza dotychczasowe doświadczenie językowe[11].

Języki mogą być opisane przez różne reprezentacje, takie jak gramatyki bezkontekstowe, wyrażenia regularne i inne. Problem indukcji gramatyki jest więc zdefiniowany przez klasę reprezentacji indukowanego języka, ale także dostępność danych uczących, które mogą należeć do języka (przykłady pozytywne), do języka nie należą (przykłady negatywne) lub też dostarczają dodatkowych informacji. Zadanie wnioskowania gramatycznego może być zdefiniowane jako uczenie na podstawie danych trenujących gramatyki G, która prawidłowo identyfikuje język L, to jest $L(G) = L$.

1.4.1. Identyfikacja w granicy

Jednym z pierwszych, który badał metody indukcji gramatyki był Solomonoff (1959, 1964), ale formalne podstawy teorii uczenia na podstawie przykładów zostały sformułowane w pracy Golda z 1967 r. (Gold 1967), w której wprowadzono pojęcie identyfikacji w granicy (*identification in the limit*). Pojęcie to odnosi się do ograniczonego (ograniczonego w czasie) działania algorytmu rozpoznającego nieskończoną sekwencję przykładów uczących. Algorytm po każdej prezentacji kolejnego przykładu

[10] Powyższe stwierdzenie w literaturze przyjmuje się zazwyczaj jako oczywiste (Nowak i in. 2002), chociaż Cichosz (2000) wyraźnie rozróżnia od uczenia indukcyjnego uczenie przez wyjaśnianie, uczenie automatów oraz uczenie ze wzmocnieniem.

[11] Chomsky zauważył, iż językowe otoczenie dziecka nie precyzuje dokładnie reguł gramatycznych (Chomsky 1972), a paradoksalnie (jest to tzw. paradoks nabywania języka, *paradox of language acquisition*) dzieci tej samej społeczności językowej potrafią nauczyć się poprawnej gramatyki rodzimego języka (Jackendoff 2001). Zjawisko to jest jednym z dowodów wchodzących w skład tzw. argumentu ubóstwa bodźców (*poverty of stimulus*) (Wexler i Culicover 1980). Próbą wyjaśnienia tego fenomenu jest tyle popularna, co kontrowersyjna koncepcja gramatyki uniwersalnej (*universal grammar*, UG) (Chomsky 1965), w której zakłada się, iż dzieci uczą się poprawnej gramatyki poprzez jej wybór z ograniczonego zbioru potencjalnych, genetycznie uwarunkowanych gramatyk.

zgaduje gramatykę G (buduje hipotezę g) i jeżeli proces uczenia przebiega „prawidłowo", owe hipotezy zbiegają się do poszukiwanego rozwiązania.

Formalnie, kompletną prezentacją (*complete presentation*) języka L nad alfabetem V jest nieskończona sekwencja uporządkowanych par $(w, l) \in V^* \times \{0, 1\}$, gdzie $l = 1$ jeżeli $w \in L$ i 0 w przeciwnym wypadku i każde słowo $w \in V^*$ pojawia się co najmniej raz (*learning from informant*). Jeżeli metoda wnioskowania M działa na coraz to większym fragmencie kompletnej prezentacji L, to generuje nieskończoną sekwencję hipotez g_1, g_2, g_3 itd. Mówimy, że M identyfikuje w granicy L, jeżeli istnieje taka liczba n, że wszystkie hipotezy g_i są takie same dla $i \geq n$ i $L(g_n) = L$. Zbiór języków jest identyfikowany w granicy, czyli nauczalny (*learnability*), jeżeli każdy język ze zbioru jest nauczalny. Interesuje nas odpowiedź na pytanie, jaki zbiór języków może być indukowany przez algorytm uczący.

Gold pokazał, że wszystkie języki aż do języka kontekstowego[12] mogą być identyfikowane w granicy na podstawie ich kompletnej prezentacji. Jest jednak druga strona medalu – jeżeli zbiór uczący nie jest kompletny i zawiera tylko przykłady pozytywne (*learning from text*), to wtedy żaden zbiór języków składający się z języków skończonych i przynajmniej jednego języka nieskończonego (*super-finite languages*) nie może być identyfikowany w granicy. Oznacza to m.in., że nie istnieje algorytm uczący się jedynie na podstawie przykładów pozytywnych nawet dla zbioru wyrażeń regularnych, nie mówiąc o językach stojących wyżej w hierarchii Chomsky'ego[13].

1.4.2. Model PAC

Podstawowym założeniem identyfikacji w granicy jest wymóg indukcji gramatyki opisującej dokładnie poszukiwany język. Jednak w wielu praktycznych zastosowaniach może być wystarczająca inferencja gramatyki „prawie dokładnej"[14]. W modelu PAC (*probably approximately correct*) wymaga się od uczącego identyfikacji gramatyki jedynie „prawdopodobnie w przybliżeniu poprawnej", w miejsce pełnej jej identyfikacji (Valiant 1984, Li i Vitanyi 1993, Li i Vitanyi 1995). Co więcej oczekuje się, że identyfikacja nastąpi na podstawie określonej liczby przykładów[15]. W modelu maszyna ucząca indukuje gramatykę na podstawie etykietowanych przykładów gene-

[12] Język kontekstowy jest akceptowany przez niedeterministyczną liniowo ograniczoną Maszynę Turinga, dla której problem przynależności jest rozstrzygalny (Harrison 1978).

[13] Podejście Golda do identyfikacji języka jest według niektórych badaczy (Bertolo 2001) zbyt wyidealizowane w odniesieniu do wnioskowania języka naturalnego. Zakłada bowiem, że zdania uczące się są kompletne i wiarygodne, a nie rozpatruje informacji niekompletnej, wieloznacznej i zaszumionej (*noise data*).

[14] Argument ten jest również przytaczany w literaturze zajmującej się zagadnieniem nabywania języka przez dziecko (*first language acquistion*) (Clark 2004).

[15] Definicja nauczalności Golda nie nakłada żadnych restrykcji na liczbę przykładów oraz czas, w jakim następuje identyfikacja. Li i Vitanyi (1995) zauważyli, że nałożenie na indukcyjny model uczenia restrykcji czasowych powoduje, że uczenie staje się praktycznie niemożliwe, poza przypadkami trywialnymi.

rowanych według określonego rozkładu prawdopodobieństwa. Celem uczącego jest indukcja hipotezy (gramatyki), która odrzuca jedynie te przykłady, które są mało prawdopodobne. Szybkość uczenia zależy od wielkości przyjętego błędu nieprawidłowej klasyfikacji; im mniejszy błąd, tym wymagana jest większa liczba przykładów uczących. Ponieważ zakłada się, że nie wszystkie dostępne przykłady będą prezentowane uczącemu – poszukiwana hipoteza będzie jedynie przybliżona. Odległość pomiędzy poszukiwaną gramatyką a gramatyką wyindukowaną może być wyrażona poprzez prawdopodobieństwo zbioru przykładów, który obydwie gramatyki inaczej klasyfikują. Istnieje więc określona i co najważniejsze skończona liczba przykładów, dla których owa odległość przyjmuje założoną z góry wartość.

Chociaż model PAC jest mniej restrykcyjny od indukcji w granicy, zastosowanie jego w praktyce nie prowadzi wcale do lepszych rozwiązań. Okazuje się bowiem, że zbiór języków jest nauczalny wtedy i tylko wtedy, gdy ma skończony wymiar VC, charakteryzujący przestrzeń hipotez ze względu na ich złożoność (Vapnik i Chervonenkis 1971). Jeżeli więc zbiór możliwych języków jest arbitralny i w związku z tym ma nieskończony wymiar VC, uczenie jest niemożliwe. Wynika stąd m.in. wniosek, że zbiór wszystkich wyrażeń regularnych (a nawet zbiór wszystkich skończonych języków) ma nieskończony wymiar VC i nie może być indukowany w ramach statystycznej teorii uczenia[16].

Jeszcze innym problemem uczenia (i to nie tylko statystycznego) jest złożoność obliczeniowa algorytmów uczących. Istnieje bowiem cała klasa języków nauczalnych w sensie PAC (mających skończony wymiar VC), dla których nie ma algorytmów działających w czasie wielomianowym.

1.4.3. Minimalna długość kodu

Innym modelem uczenia stosowanym we wnioskowaniu gramatycznym jest zasada minimalnej długości kodu (*minimum description length*, MDL) (Rissanen 1989, Li i Vitanyi 1995). W modelu MDL zadaniem uczącego jest wybór takiej hipotezy, która nie tylko opisuje poprawnie zbiór uczący, ale również sama jest jak najprostsza lub ujmując to w kategoriach statystycznych – hipotezy prostsze są uznawane za bardziej prawdopodobne. Bardzo często indukcję MDL opisuje się jako kompresję danych – jakiekolwiek regularności wykryte w zbiorze uczącym mogą posłużyć do jego kompresji, tj. opisania go w krótszy sposób (Grünwald 2005)[17]. Główna zasada modelu MDL może być sformułowana następująco: najlepszą hipotezą jest ta, która minimali-

[16] Podzbiory wyrażeń regularnych generowane przez automaty skończone z n stanami mają już skończony wymiar VC i w związku z tym można na przykład określić granice liczby wymagań zdań uczących.

[17] Kompresja, opisywana właśnie w kategoriach MDL, może być podstawą ogólnej teorii obliczeń (Wolff 2003).

zuje sumę długości opisu hipotezy oraz długości przykładów zakodowanych za jej pomocą (obie długości podawane są w bitach)[18].

MDL implikuje bardzo praktyczne wnioski dla algorytmów uczących, stąd też z powodzeniem jest stosowany w indukcji gramatyk (Grünwald 1996, Osborne 1999, Hong i Clark 2001b, Jonyer i in. 2004), szczególnie gramatyk stochastycznych (Stolcke 1994, Chen 1995, Clark 2001b, Keller i Lutz 1997, Keller i Lutz 2005).

1.4.4. Uczenie się na podstawie zapytań

Uczenie na podstawie zapytań (*query learning*), nazywane również aktywnym uczeniem (*active learning*), zostało zaproponowane na początku lat osiemdziesiątych ubiegłego wieku przez Angluin (1981). W podejściu tym uczący ma dostęp do wyroczni (*oracle*), która może odpowiadać na pytania dotyczące języka. Owe pytania mogą dotyczyć przynależności (*membership queries*), czy przykład jest rozpoznawany przez indukowaną gramatykę, oraz równoważności (*equivalence queries*), w której uczący pyta, czy aktualna hipoteza zgodna jest z poszukiwaną gramatyką[19]. Odpowiedzi na pytania o przynależność przykładu do docelowego języka mogą mieć charakter binarny. W przypadku pytań o równoważność przyjmuje się, że przeczącej odpowiedzi wyroczni towarzyszy kontrprzykład – przykład rozpoznawany przez hipotezę uczącego, ale nienależący do indukowanego języka, lub odwrotnie, przykład należący do języka, ale przez hipotezę uczącego nierozpoznawany. Angluin udowodniła, że posługując się pytaniami zgodnie z tzw. algorytmem L^*, można indukować języki regularne w czasie wielomianowym względem długości najdłuższego kontrprzykładu dostarczanego przez wyrocznię w odpowiedzi na pytanie o równoważność (Angluin 1987a). Wielomianowa złożoność algorytmu indukcji wynika z zakładanej dostępności do wyroczni i obszernej informacji uczącej. W (Balcazar i in. 1994) udowodniono, że zastępowanie pytań o równoważność pytaniami o przynależność wymaga wykładniczego wzrostu tych ostatnich. Całkowita rezygnacja z pytań o równoważność jest możliwa w probabilistycznej wersji algorytmu L^*, która umożliwia indukcję gramatyki z tym większym prawdopodobieństwem, im czas obliczeń jest większy (Ron i Rubinfeld 1995).

1.5. Indukcja gramatyk bezkontekstowych

Głównym obszarem zainteresowań naukowców zajmujących się wnioskowaniem gramatycznym jest niewątpliwie uczenie gramatyk regularnych oraz równoważnych

[18] Zasada MDL zaliczana jest również do probabilistycznych metod uczenia i jest często definiowana poprzez twierdzenie Bayesa (Cichosz 2000).

[19] W literaturze ten model uczenia określany jest również jako model MAT (*Minimally Adequate Teacher*).

im automatów skończonych. Wynika to z faktu, iż formułowane problemy mogą być stosunkowo proste, a jednocześnie wymagają stosowania zaawansowanych metod wychodzących poza klasyczne metody uczenia maszynowego[20]. Co więcej, gramatyki regularne i równoważne automaty są jedyną klasą, dla której znane są pozytywne wnioski dotyczące możliwości uczenia. W statystycznej teorii uczenia rozpatruje się bowiem pewne podklasy automatów mających skończony wymiar VC, czyli nauczalnych w sensie PAC: automaty skończone z n-stanami (Ishigami i Tani 1997) czy też acykliczne automaty stochastyczne (Ron i in. 1994). W ramach paradygmatu uczenia na podstawie zapytań znany jest natomiast algorytm L^* znajdujący automat skończony w czasie wielomianowym. Poszukiwania efektywnych algorytmów uczenia dla deterministycznych automatów skończonych doprowadziły do sformułowania modelu identyfikacji w granicy w wielomianowym czasie i danych (*identification in the limit from polynomial time and data*) (Gold 1978, Higuera 1997). W modelu tym wymaga się, aby indukowana gramatyka posiadała pewien właściwy sobie zbiór uczący o wielomianowym rozmiarze. Dołączenie tego zbioru do etykietowanych danych uczących powoduje wyuczenie równoważnej gramatyki[21].

Indukcja gramatyk bezkontekstowych jest trudna niezależnie od przyjętego modelu uczenia. Oczywiście, z teoretycznego punktu widzenia klasa języków bezkontekstowych jest nauczalna – jak wszystkie klasy do kontekstowej włącznie – w sensie paradygmatu Golda, to znaczy jest identyfikowana w granicy na podstawie kompletnej prezentacji. Jednak, jak już wspominano o tym wcześniej, nie są znane w tym ujęciu efektywne algorytmy wnioskowania[22]. Klasa języków bezkontekstowych nie ma skończonego wymiaru VC, nie jest więc indukowana w sensie PAC. Badania prowadzone w ramach modelu zapytań pozostawiają kwestię wielomianowej złożoności algorytmów indukcji gramatyk bezkontekstowych wciąż otwartą (Angluin 2001)[23]. W pracy (Higuera 1997) udowodniono natomiast, że gramatyki bezkontekstowe oraz automaty niedeterministyczne nie są wielomianowo identyfikowane w czasie i danych.

Pomimo tych raczej negatywnych rezultatów, proponowane są w literaturze różnorakie podejścia do indukcji gramatyk bezkontekstowych, polegające najczęściej na wyposażeniu algorytmu uczącego w dodatkowe informacje, takie jak przykłady negatywne, czy też informację strukturalną; proponuje się alternatywne reprezentacje gramatyki, ogranicza indukcję do podklas gramatyki bezkontekstowej, analizuje gra-

[20] Gold udowodnił, że poszukiwanie najmniejszego automatu DFA zgodnego ze zbiorem uczącym jest problemem *NP*-trudnym (Gold 1978).

[21] de la Higuera (1997) udowodnił równoważność tak zdefiniowanej nauczalności tzw. modelowi nauczania (Goldman i Mathias 1996).

[22] Model Golda w ogóle nie porusza problemu złożoności algorytmu identyfikacji. Pitt odrzuca możliwość wielomianowego w czasie uczenia w tym podejściu, ze względu na całkowity brak kontroli nad wielkością uczących przykładów (Pitt 1989).

[23] W (Pitt i Warmuth 1988) udowodniono, że uczenie gramatyk CFG jest tak złożone, jak obliczanie pewnych predykatów kryptograficznych.

matyki probabilistyczne, wreszcie poszukuje się algorytmów opartych o metody sztucznej inteligencji.

1.5.1. Indukcja na podstawie tekstu

W podrozdziale tym zostaną zaprezentowane algorytmy stosowane w indukcji CFG na podstawie tzw. *tekstu*, który jest sekwencją ciągów nad alfabetem języka uczonego przez gramatykę. Tekst może zatem zawierać również ciągi, które nie należą do poszukiwanego języka. Uczenie na podstawie nieetykietowanego tekstu nie jest wg definicji Golda zbieżne w granicy.

Jednym z pierwszych algorytmów uczenia na podstawie tekstu, a dokładnie jedynie zdań pozytywnych, był algorytm zaproponowany przez Solomonoffa (1959). W algorytmie tym uczący ma dostęp do przykładów pozytywnych R^+ języka L oraz pytań o przynależność. Zadaniem algorytmu jest szukanie powtarzających się wzorców w ciągach; dla każdego ciągu $w \in R^+$ tworzy się nowe ciągi w' poprzez kasowanie z w podciągów, a następnie pyta się wyroczni czy nowo utworzony $w' \in R^+$. Jeżeli odpowiedź jest pozytywna, algorytm wstawia do ciągu usunięte wcześniej, a teraz powtórzone wielokrotnie, podciągi i znowu pyta o przynależność wynikowego ciągu do języka. Jeżeli i tym razem odpowiedź jest pozytywna, oznacza to wywnioskowanie reguły rekursywnej. Jeżeli zatem w zbiorze uczącym jest wiele ciągów postaci $a^n b^n$, można wnioskować, że jedną z reguł gramatyki jest $A \rightarrow aAb$.

Algorytm Solomonoffa jest oczywiście nieefektywny, silnie zależy od postaci ciągów w zbiorze uczącym i zgodnie z twierdzeniem Golda nie jest w stanie indukować pełnej klasy języków bezkontekstowych. Tym niemniej, idea poszukiwania wzorców w ciągach odpowiadających symbolom nieterminalnym została zastosowana w wielu późniejszych algorytmach.

W (Knobe i Knobe 1976) również rozważano sytuację, w której uczący ma dostęp do przykładów pozytywnych oraz pytań o przynależność. Algorytm jest w istocie kolekcją oczywistych heurystyk, a jego działanie zależy od porządku, w którym prezentowane są przykłady uczące.

W algorytmie zaprezentowanym przez Tanatsugu (1987) uczenie odbywa się na zbiorze zdań pozytywnych i negatywnych. Metoda polega na usuwaniu zagnieżdżonych struktur z ciągu, indukcji gramatyki liniowej z przykładu, a następnie składaniu tak wywnioskowanych gramatyk liniowych w gramatykę bezkontekstową. Algorytm jest w stanie indukować pełną klasę CFG, nie jest jednak podana jego złożoność obliczeniowa.

1.5.2. Indukcja z danych strukturalnych

Alternatywną w stosunku do uczenia na podstawie tekstu metodą inferencji gramatyk bezkontekstowych jest uczenie wspomagane dodatkową informacją w postaci

danych strukturalnych. Częstokroć bowiem jesteśmy zainteresowani nie tylko ciągami, które indukowana gramatyka rozpoznaje, ale również drzewami rozbioru (*parse tree*, *derivation tree*), które gramatyka przypisuje do analizowanych ciągów.

Jedną ze stosowanych metod reprezentacji danych strukturalnych są gramatyki nawiasowe (*parenthesis grammar*) (McNaughton 1967). Gramatykę nawiasową tworzy się przez zastąpienie każdej produkcji $A \rightarrow \alpha$ produkcją $A \rightarrow (\alpha)$, gdzie nawiasy nie należą do liter alfabetu. Crespi-Reghizzi podał algorytm przyrostowy identyfikujący gramatykę z pierwszeństwem operatorowym (*operator precedence grammars*) w granicy na podstawie nawiasowanych pozytywnych przykładów (Crespi-Reghizzi 1971) oraz algorytm uczenia *k*-rozróżnialnych i jednorodnych gramatyk (*k-distinct and homogeneous grammars*) z nawiasowanych przykładów pozytywnych oraz przykładów negatywnych, które również mogą, ale nie muszą, być oznaczone nawiasami (Crespi-Reghizzi 1974).

Dane strukturalne mogą być również pamiętane w postaci szkieletów (*skeleton*), które są drzewami rozbioru z usuniętymi etykietami nieterminali (Levy i Joshi 1978). Drzewa szkieletowe są akceptowane przez pewien rodzaj automatów skończonych nazywanych automatami drzew szkieletowych (*skeletal tree automata*, *tree automata*, SA). Automat drzew szkieletowych *A*, po otrzymaniu drzewa *T* na wejściu, rozpoczyna analizę od przypisania stanów do liści drzewa. Kolejne stany są przypisywane do węzłów drzewa głównie na podstawie stanów węzłów – dzieci. *A* akceptuje *T* wtedy i tylko wtedy, gdy przypisze stan końcowy do węzła – korzenia. Problem indukcji CFG może być więc sprowadzony do problemu indukcji SA. Sakakibara rozszerzył metodę uczenia na podstawie zapytań na algorytm identyfikujący SA w czasie wielomianowym (Sakakibara 1987, 1990). Metoda ta wymaga zadawania zapytań o strukturalną przynależność (*structural membership queries*) oraz strukturalną równoważność (*structural equivalence queries*). Sakakibara podał również metodę indukcji odwrotnej gramatyki bezkontekstowej (*reversible* CFG) w granicy na podstawie jedynie pozytywnych danych strukturalnych (Sakakibara 1992), bazując na algorytmie indukcji automatów odwrotnych (Angluin 1982). Podobny algorytm zaproponowała Fass, chociaż jej podejście zakłada odpowiednią selekcję przykładów (Fass 1983). Fernau (2002) rozszerzył algorytm (Sakakibara 1992) na indukcję δ–rozróżnialne języki drzew (δ*-distinguishable tree languages*). Inny algorytm (Seginer 2003), podobnie jak metoda Fernaua, dzieli zbiór uczący na nauczalne podklasy (*context set*), które razem pokrywają całą klasę CFG. Sakakibara rozpatrywał również problem uczenia gramatyki bezkontekstowej na podstawie jedynie częściowo strukturalnych danych (Sakakibara i Muramatsu 2000)[24]. Zaproponowana metoda uczenia jest wspomagana przez algorytm genetyczny.

[24] Częściowo strukturalne dane (*partially structured example*) mają usunięte niektóre pary nawiasów.

1.5.3. Indukcja podklas gramatyk bezkontekstowych

Najbardziej powszechną metodą eliminacji negatywnych wniosków płynących z paradygmatu Golda jest indukcja takich klas gramatyk, które są w jakiś sposób ograniczone i nie zawierają wszystkich skończonych języków. Metody bazujące na takim podejściu mają dostęp do wyroczni, potrafiącej odpowiadać na szczegółowe zapytania dotyczące przykładowo wyboru potencjalnych kandydatów na najlepsze rozwiązania.

Uczenie języków liniowych[25] (*linear languages*) (Takada 1987, Mäkinen 1990) oraz równych języków liniowych (*even linear languages*) (Takada 1988, Sempere i Garcia 1994, Mäkinen 1996) można sprowadzić do problemu uczenia języków regularnych, a zatem zadania wielomianowego. W ramach tej klasy języków badano nauczalność hierarchii języków liniowych (Takada 1994), przypadek gdy są dostępne tylko przykłady pozytywne (Koshiba i in. 1997) oraz gdy przykłady pozytywne mają postać strukturalną (Sempere i Nagaraja 1998).

Angluin udowodniła, że *k*-ograniczone gramatyki bezkontekstowe (*k-bounded* CFG) są identyfikowalne w czasie wielomianowym przy użyciu pytań o równoważność oraz pytań o przynależność nieterminali (*nonterminal membership queries*) (Angluin 1987b). Zapytania o przynależność terminali pozwalają ustalić, czy dany ciąg jest wyprowadzany z określonego nieterminala, co w efekcie umożliwia uczącemu określenie struktury gramatyki.

W algorytmie indukcji prostych deterministycznych języków (*simple deterministic languages*, SDL) zamiast pytań o przynależność nieterminali stosowane są rozszerzone pytania o równoważność (*extended equivalence queries*), które mogą dotyczyć również gramatyk równoważnych (Ishizaka 1990). Tego typu pytania nie dostarczają w sposób bezpośredni takiej wiedzy strukturalnej, jak pytania o przynależność nieterminali, ale ze względu na nierozstrzygalność problemu równoważności gramatyk bezkontekstowych, odpowiedzi wyroczni muszą nieść złożoną informację.

Yokomori podał wielomianowy algorytm dla indukcji SDL, którego efektem działania nie jest docelowa gramatyka, lecz jedynie jej hipoteza (Yokomori 1988a). Algorytm stosuje prefiksowe pytania o przynależność (*prefix membership queries*) oraz pochodne pytania o przynależność (*derivative membership queries*). Pierwsza grupa pytań dotyczy odpowiedzi na pytanie, czy prefiks zadanego ciągu należy do poszukiwanego języka. Druga grupa proponuje dwie pary ciągów (u, v) oraz (u', v'); odpowiedź jest pozytywna wtedy i tylko wtedy, gdy $\{w \mid uwv \in L^*\} = \{w \mid u'wv' \in L^*\}$.

Z innych nauczalnych podklas klasy języków bezkontekstowych warto jeszcze wymienić strukturalnie odwrotne języki (*structurally reversible languages*) (Burago 1994), jednolicznikowe języki, tj. akceptowane przez deterministyczne automaty jed-

[25] Gramatyki liniowe są podklasą gramatyk bezkontekstowych; w produkcjach języków liniowych zawsze mamy tylko jeden symbol nieterminalny – słowo produkowane jest liniowym sposobem.

nolicznikowe (*one-counter languages*) (Berman i Roos 1987), języki osiowe (*pivot languages*) (Feldman i in. 1969), czy tzw. bardzo proste języki (*very simple languages*) (Yokomori 1991).

1.5.4. Indukcja alternatywnych reprezentacji gramatyk bezkontekstowych

Większość z wymienionych w poprzednim podrozdziale podklas nie ma większego znaczenia lingwistycznego. Nie jest to już natomiast prawdą w odniesieniu do pewnych podklas gramatyki kategorialnej (*categorial grammar*), która jest równoważna gramatyce bezkontekstowej. Kanazawa w pracy doktorskiej przeprowadził analizę nauczalności szeregu podklas gramatyki kategorialnej w ujęciu paradygmatu Golda (Kanazawa 1995).

W literaturze znane są również próby indukcji języków bezkontekstowych, które nie są reprezentowane przez gramatykę.

Yokomori rozszerzył algorytm identyfikujący w granicy wyrażenia regularne na podstawie przykładów pozytywnych i negatywnych, na algorytm uczący się bezkontekstowych wyrażeń (*contex-free expression*) (Yokomori 1988b). Niestety algorytm nie działa w czasie wielomianowym.

Arikawa i in. (1992) także próbowali redukować problem indukcji gramatyki bezkontekstowej do problemu, dla którego znane jest już rozwiązanie. W tym celu sformułowali tzw. elementarny system formalny (*regular elementary formal system*), który składa się ze zbioru symboli, predykatów, klauzul oraz systemu wnioskowania Shapira. Również i ten algorytm nie jest wielomianowo efektywny.

1.5.5. Indukcja stochastycznych gramatyk bezkontekstowych

W 1969 roku Horning podał algorytm wyliczeniowy, który identyfikował stochastyczne gramatyki bezkontekstowe (*stochastic context-free grammar*, SCFG) w granicy z prawdopodobieństwem równym 1, na podstawie samych danych stochastycznych, czyli wygenerowanych przez SCFG (Horning 1969). Oznacza to, że gramatyki stochastyczne są nauczalne na podstawie jedynie przykładów pozytywnych.

Formalizm SCFG jest wariantem gramatyki bezkontekstowej, w którym każda produkcja ma przypisane prawdopodobieństwo z przedziału [0, 1]. Dla stochastycznej gramatyki bezkontekstowej żądamy, aby suma prawdopodobieństw wszystkich reguł wyprowadzanych z tego samego symbolu nieterminalnego była równa jeden[26]. Prawdopodobieństwo drzewa rozbioru ciągu należącego do języka generowanego przez SCFG jest definiowane jako produkt prawdopodobieństw wszystkich reguł zastoso-

[26] W takim wypadku mówimy, że gramatyka jest odpowiednia (*proper*).

wanych w wyprowadzeniu ciągu[27]. Prawdopodobieństwo ciągu jest sumą prawdopodobieństw wszystkich jego wyprowadzeń. Gramatyka jest zgodna (*consistent*), gdy suma prawdopodobieństw wszystkich ciągów języka jest równa jeden[28].

W uczeniu SCFG można wyróżnić zasadniczo dwa podejścia, w zależności od posiadanej początkowej wiedzy. Jeżeli reguły gramatyczne są już wiadome, a zatem znamy strukturę gramatyki, możemy się skoncentrować na poszukiwaniu wartości prawdopodobieństw przypisywanych produkcjom. Najczęściej stosowany jest w tym przypadku algorytm *inside-outside* (Baker 1979, Lari i Young 1990). Alternatywne metody estymacji prawdopodobieństw zaproponowano w (Ra i Stockman 1999) oraz (Sakakibara i in.1994).

W podejściu drugim proces indukcji gramatyki stochastycznej jest podzielony na dwa etapy: w etapie pierwszym następuje uczenie reguł, w etapie drugim indukowane są prawdopodobieństwa reguł. Jeżeli mamy dodatkową informację o poszukiwanej gramatyce w postaci pewnych danych o jej strukturze, możemy zastosować metody uczenia gramatyk drzew (Sakakibara 1990, 1992). Jeżeli takiej wiedzy nie posiadamy, poszukiwany jest uproszczony automat opisujący język ograniczony (lokalny), a następnie estymowane są prawdopodobieństwa (Rico-Juan i in. 2002). Bezpośrednie indukowanie gramatyki bezkontekstowej jest trudne i wymaga stosowania inteligentnych metod obliczeniowych, takich jak algorytmy genetyczne (Kammeyer i Belew 1996, Keller i Lutz 1997, 2005).

Oddzielnym zagadnieniem w indukcji stochastycznych gramatyk bezkontekstowych jest ich parsing (Stolcke 1995).

1.5.6. Indukcja z zastosowaniem metod sztucznej inteligencji

Kombinatoryczna złożoność problemów indukcji gramatycznej wynika głównie z olbrzymiej przestrzeni potencjalnych rozwiązań, na które składają się potencjalne gramatyki lub automaty. W przeszukiwaniu przestrzeni rozwiązań pomocne mogą być techniki stosowane w sztucznej inteligencji.

VanLehn i Ball zaproponowali, by przestrzeń poszukiwań dla gramatyki bezkontekstowej była definiowana przez tzw. przestrzeń wersji (*version space*) (VanLehn

[27] Przypisywanie prawdopodobieństwa do każdej struktury językowej jest w istocie wskazaniem, na ile ta struktura należy do rozpatrywanego języka. Idea ta jest bliska rozumieniu gramatyczności języka naturalnego; w odpowiednim kontekście niemal każda struktura językowa może być gramatycznie poprawna (czy też bardziej prawdopodobna).

[28] W istocie warunek zgodności stochastycznej gramatyki bezkontekstowej nie jest tak decydujący jak to, żeby była odpowiednia. Jednak definicja zgodności gramatyki może powodować niekontrolowany rozrost drzew wyprowadzeń przy jednoczesnym zachowaniu warunku zgodności. W przypadku prawdopodobieństw estymowanych na podstawie danych (Sànchez i Benedi 1997) zdefiniowane w (Booth i Thompson 1973) warunki zgodności są wystarczające dla ograniczenia ekspansji drzew wyprowadzeń.

i Ball 1987)[29]. Giordano wprowadził częściowy porządek w przestrzeni wersji (Giordano 1994), a Langley i Stromsten (2000)[30] ukierunkowywali poszukiwania w przestrzeni, posiłkując się warunkiem prostoty gramatyki oraz uzyskiwanych w niej wyprowadzeń.

W wielu pracach zasadniczym mechanizmem indukcji są algorytmy ewolucyjne. Ewolucji może podlegać bezpośrednio gramatyka bezkontekstowa, jak również jej stochastyczna wersja; przedmiotem wnioskowania są także równoważne gramatyce bezkontekstowej automaty.

Jedną z pierwszych prób ewolucji CFG była praca Wyarda z 1991 r. (Wyard 1991). Udanej ewolucji podlegała jedynie gramatyka zdań nawiasowych, nie udało się natomiast wyewoluować gramatyki dla języka złożonego z jednakowej liczby symboli *a* i *b* (język AB). W następnej pracy Wyard analizował różne reprezentacje gramatyki, począwszy od notacji BNF (*Backus Naur Form*), przez postać normalną *Greibach* (*Greibach Normal Form*, GNF), postać normalną Chomsky'ego (*Chomsky Normal Form*, CNF) oraz reprezentację bitową (Wyard 1994). Rozkład gramatyczny zdań zbioru uczącego był wykonywany przez tzw. *chart parser*[31], algorytm o dużej złożoności obliczeniowej. Sukcesem zakończyła się indukcja języka palindromicznego złożonego maksymalnie z 4 symboli oraz uproszczonej gramatyki języka naturalnego, opisującej frazy czasownikowe i rzeczownikowe. Funkcja oceny nie uwzględniała częściowo poprawnie przeanalizowanych zdań uczących. W 1993 roku Lucas zastosował ciąg bitowy do reprezentacji pojedynczej reguły, z powodzeniem indukując dwuliterowe palindromy (Lucas 1993). W roku następnym Lucas opublikował pracę, w której skoncentrował się na metodach strukturalizacji chromosomów zwiększających efektywność stosowanego w ewolucji algorytmu genetycznego (Lucas 1994). Zaproponowana metoda reprezentacji umożliwiła efektywne uczenie trójliterowej gramatyki palindromicznej. W tym samym roku Lankhorst zastosował „niskopoziomową" binarną reprezentację pojedynczej reguły gramatyki CFG oraz algorytm genetyczny w ewolucji zdań nawiasowych, zbioru języków regularnych, języka AB oraz „mikro" języka naturalnego (Lankhorst 1994). W odróżnieniu od Wyarda i Lucasa, Lankhorst użył stosunkowo złożonej funkcji oceny, uwzględniającej nie tylko poprawnie sklasyfikowane całe przykłady, ale również ich najdłuższe podciągi oraz poprawność predykcji następnego symbolu. Losee (1996) ewoluował gramatykę bezkontekstową opisującą abstrakty dokumentów w języku naturalnym, stąd też jego funkcja oceny uwzględniała poprawność wyszukiwania i filtrowania informacji. W podejściu Smitha i Wittena ewolucji sterowanej przez algorytm genetyczny podle-

[29] Przestrzeń wersji definiowana jest jako zbiór wszystkich możliwych generalizacji gramatyki akceptującej przykłady pozytywne, takich generalizacji, które są zgodne z przykładami uczącymi. W ogólnym przypadku przestrzeń ta jest nieskończona.

[30] Praca ta w istocie opiera się na modelu MDL.

[31] Algorytm *chart parser* dedykowany jest analizie zdań języka naturalnego, a więc również zdań wieloznacznych.

gał chromosom reprezentujący całą gramatykę (Smith i Witten 1995, 1996)[32]. Funkcją dostosowania była zdolność gramatyki do rozbioru zdań uczących. Metodę trenowano na niewielkim zbiorze uczącym fraz języka naturalnego, indukując gramatykę opisującą grupę rzeczownikową oraz czasownikową. Gramatyka bezkontekstowa może być reprezentowana poprzez swoje drzewo rozbioru. W (Korkmaz i Ucoluk 2001) do indukcji uproszczonej gramatyki języka angielskiego zastosowano programowanie genetyczne, które operuje właśnie na strukturach drzewiastych. Aby uwzględnić zależności występujące pomiędzy poszczególnym fragmentami drzewa rozbioru, zaproponowano dodatkowy moduł sterujący ewolucją. Pewną odmianą programowania genetycznego jest gramatyczna ewolucja (*grammatical evolution*), która może produkować kod dowolnego języka programowania (Ryan i in. 1998). Chromosomy są reprezentowane nie jak w klasycznym programowaniu genetycznym przez drzewa rozbioru, ale wektory liczb całkowitoliczbowych. Każda liczba oznacza regułę w gramatyce BNF. Tsoulos i Lagaris zastosowali z powodzeniem gramatyczną ewolucję w indukcji zbioru języków regularnych oraz nawiasowych (Tsoulos i Lagaris 2005). Innym przykładem zastosowania programowania genetycznego w ewolucji gramatyki bezkontekstowej jest podejście rozwijane przez zespół (Mernik i in. 2003, Javed i in. 2004). Zasadniczym przeznaczeniem budowanego systemu jest indukcja parsera dziedzinowo zorientowanego języka programowania. Mechanizm programowania genetycznego został rozszerzony o heurystyczne operatory opcjonalności i iteracji oraz operator jednopunktowego krzyżowania gramatyk[33]. Gramatyka reprezentowana jest w notacji BNF. Dla każdej wyewoluowanej gramatyki generowany jest parser typu LR(1) przez autorski kompilator LISA (Mernik i in. 2002), który następnie jest weryfikowany (oceniany) w środowisku uczących programów. Oryginalną i przede wszystkim efektywną metodę indukcji gramatyk bezkontekstowych z zastosowaniem algorytmu genetycznego zaproponował Sakakibara (Sakakibara i Kondo 1999, Sakakibara i Muramatsu 2000, Sakakibara 2005). Rosnąca wykładniczo wraz z długością wejściowego ciągu przestrzeń potencjalnych rozwiązań, czyli gramatycznych struktur nieznanej gramatyki, zapisywana jest w tablicy rozbioru. Wielkość tablicy, której konstrukcja wzorowana jest na tablicy rozbioru stosowanej przez algorytm CYK (Kasami 1965), jest wielomianowo zależna od długości ciągu wejściowego[34]. Algorytm genetyczny jest odpowiedzialny za łączenie rozłącznych zbiorów nieterminali w taki

[32] W uczeniu maszynowym metodę, w której osobnik w ewoluowanej populacji reprezentuje cały zestaw reguł (w przypadku wnioskowania gramatycznego zestaw produkcji gramatyki bezkontekstowej), nazywamy podejściem Pitt (Pittsburgh), w podejściu Michigan ewolucji podlegają pojedyncze reguły. Obydwie nazwy określają również amerykańskie uniwersytety, na których obie metody były niezależnie rozwijane.

[33] Jednopunktowy operator krzyżowania krzyżuje dwie gramatyki, wymieniając jedynie ich drugie połowy. Aby zapewnić poprawność uzyskanych osobników, punkt krzyżowania jest wybierany pośrodku mniejszej gramatyki oraz zostaje sprawdzone, czy nie wypada w środku produkcji.

[34] Dla ciągu wejściowego o długości n, liczba pamiętanych w tablicy nieterminali nie przekracza n^3, a liczba produkcji n^5.

sposób, by powstała gramatyka była spójna zarówno z przykładami pozytywnymi, jak i negatywnymi[35]. Badano język $a^m b^m c^n$[36] oraz sumę dwóch języków ac^m *i* bc^m. Efektywność algorytmu uczenia jest wyższa, gdy przykłady są częściowo strukturalne, co więcej jest również wyższa od wcześniejszego algorytmu Sakakibary pracującego na danych w pełni strukturalnych (Sakakibara 1992).

Równolegle z pracami dotyczącymi bezpośredniej indukcji gramatyk bezkontekstowych przebiegają badania nad indukcją równoważnych gramatyce automatów. W (Sen i Janakiraman 1992) zastosowano algorytm genetyczny do indukcji deterministycznego automatu ze stosem (*deterministic pushdown automaton*, PDA). Akceptacja słowa wejściowego następowała pod warunkiem osiągnięciu stanu końcowego i pustego stosu. Takie założenia ograniczały ewoluowany język do deterministycznego języka bezkontekstowego. Chromosom reprezentował symbol wejściowy, symbol stosowy, następny stan oraz bit reprezentujący akcję na stosie. W pracy opisano zakończone sukcesem próby ewolucji języka nawiasowego oraz języka $a^n b^n$. O ile Sen i Janakiraman wyprowadzali wnioski na podstawie pojedynczych uruchomień algorytmu genetycznego, o tyle Huijsen (1993) w swoich eksperymentach uwzględnił stochastyczną naturę przetwarzania ewolucyjnego i obliczał średnią z wielu przebiegów. Ewolucji podlegał nie tylko automat PDA, ale również niedeterministyczny automat ze stosem (*non-deterministic pushdown automaton*, NPDA) oraz gramatyka bezkontekstowa. Tej ostatniej nie udało się poprawnie wyewoluować. Badane były języki nawiasowe, język AB oraz dwuliterowy język palindromiczny o parzystej długości. Zbiór zdań uczących zawierał wszystkie zdania języka do określonej długości. Huijsen zastosował tablicę liczb całkowitoliczbowych do reprezentacji chromosomu z wyróżnionymi liczbami-markerami, oddzielającymi poszczególne przejścia automatu. Lankhorst również indukował automat NPDA, używając dwóch metod reprezentacji zadania: binarnej oraz całkowitoliczbowej (Lankhorst 1995). Analizowanych było w sumie 10 języków, w tym zbiór języków regularnych, język nawiasowy, dwuliterowe palindromy, język AB oraz uproszczony język naturalny. Funkcja oceny uwzględniała poprawność klasyfikacji zdań uczących, częściowo poprawne analizowane podciągi oraz stopień wypełnienia stosu. Zbiór zdań uczących był dobierany losowo z zastosowaniem rozkładu Poissona dla ich długości. Wyniki były uśredniane po 10 przebiegach. Wszystkie eksperymenty zakończyły się sukcesem. Zomorodian nie indukował bezpośrednio automatu DPA, lecz język jego opisu (Zomorodian 1995). Ewoluowaną populację stanowił zbiór programów, który podlegał regułom programowania genetycznego. Ocena programu była dokonywana pośrednio, poprzez ocenę dostosowania utworzonego przez program automatu do zbioru uczącego. Kon-

[35] Łączenie rozłącznych zbiorów nieterminali jest tożsame z zadaniem podziału zbioru nieterminali. Zadanie podziału zawiera zadanie znalezienia automatu kanonicznego spójnego ze zbiorem uczącym, które jest przykładem problemu *NP* trudnego (Gold 1978).

[36] Język $a^m b^m c^n$ zawiera wszystkie charakterystyczne własności języka bezkontekstowego, tj. rozgałęzienia ($A \rightarrow BC$), nawiasy ($A \rightarrow aBb$) i iterację ($A \rightarrow aA$).

cepcja pośredniej notacji została poprawnie zweryfikowana na języku $a^n b^n$ oraz języku nawiasowym.

W literaturze znane są również próby poszukiwań w drodze ewolucji gramatyk i automatów wychodzących poza klasę języków bezkontekstowych, np. maszyn Turinga (Tanomaru 1997, Vallejo i Ramos 2001).

Pierwszą pracą, w której udokumentowano ewolucję stochastycznej gramatyki bezkontekstowej, była (Kammeyer i Belew 1996). Jej autorzy rozpatrywali zagadnienie indukcji dwóch stosunkowo prostych języków bezkontekstowych, tj. języka nawiasowego oraz języka AB. Ewolucją reguł gramatyki sterował algorytm genetyczny, natomiast prawdopodobieństwa przypisywane poszczególnym produkcjom były wyszukiwane przy użyciu algorytmu *inside-outside* (IO). Keller i Lutz (1997, 2005) w swoim podejściu zrezygnowali z algorytmu IO, którego efektywność w silnym stopniu zależy od wartości początkowych. Produkcje gramatyki oraz ich prawdopodobieństwa poszukiwał algorytm genetyczny, który operował nie bezpośrednio na SCFG, ale pewnej jej odmianie, nazwanej BWG[37] (*biased weighted grammars*). Ewolucji poddano język AB, $a^n b^n$, język nawiasowy z jednym i dwoma typami nawiasów, dwuliterowy palindrom o parzystej długości oraz trójliterowy palindrom.

W ewolucji gramatyk bezkontekstowych próbuje się również zastosować sieci neuronowe. Das i in. (1992, 1993) zaproponowali indukcję deterministycznego CFG, wykorzystującą rekurencyjną sieć neuronową ze stosem (*recurrent neural network pushdown automata*, NNPDA)[38]. NNPDA składa się z rekurencyjnej sieci neuronowej zintegrowanej z zewnętrznym stosem poprzez funkcję błędu. System może uczyć się jednocześnie funkcji przejść automatu stosowego oraz akcji kontrolujących stos. W sieci neuronowej wyróżnia się neurony stanów i wejścia, neurony czytające wierzchołek stosu, pojedynczy neuron akcji wskazujący jedną z operacji stosowych oraz pojedynczy neuron wyjściowy. Po prezentacji sekwencji wejściowej porównywany jest stan neuronu wyjściowego z etykietą sekwencji (0 lub 1). Jeżeli porównanie jest negatywne, uruchamiana jest funkcja poprawy wag sieci, aby zminimalizować wartość funkcji błędu. Argumentami funkcji błędu jest wartość neuronu wyjściowego oraz długość stosu. W przypadku sekwencji pozytywnych oczekuje się, by wartość neuronu wyjściowego była bliska 1, a długość stosu zerowa, w przypadku sekwencji negatywnej wartość neuronu wyjściowego bliska 0, a stos niepusty. NNPDA jest w stanie nauczyć się języka nawiasowego oraz języka $a^n b^n$. W nowszych pracach uczeniu podlega również i sam mechanizm stosowy, odpowiedzialny za wyodrębnianie podsekwencji w sekwencjach wyjściowych. Prosta sieć rekurencyjna (*simple recurrent network*, SRN) trenowana przez (Wiles i Elman 1995) jest w stanie wyuczyć się języka $a^n b^n$ z jedenastoma poziomami zagnieżdżeń, a sekwencyjna sieć kaskadowa

[37] BWG można potraktować jako pewne uogólnienie gramatyk obciążonych (*weighted grammar*) (Salomaa 1969).

[38] Sieć neuronowa pełni funkcję uczącego się kontrolera automatu ze stosem.

(*sequential cascaded network*, SCN) uczy się języka kontekstowego $a^n b^n c^n$ (Bodén i Wiles 2000, 2002).

Spośród wielu metod i algorytmów sztucznej inteligencji (Kwaśnicka i Spirydowicz 2004) obiecującym modelem, i to zarówno ze względu na jego prostotę, uniwersalność, jak i efektywność, jest uczący się system klasyfikujący (*learning classifier system*, LCS). LCS[39] został zaproponowany przez twórcę algorytmów genetycznych Johna Hollanda w 1976 r. (Holland 1976). Systemy klasyfikujące uczą się prostych syntaktycznie reguł, zwanych klasyfikatorami, w celu koordynacji swoich działań w dowolnym środowisku. Stanowią zatem uproszczony model zachowania organizmu dostosowującego się do otoczenia[40]. Jeżeli środowiskiem systemu klasyfikującego będzie zbiór zdań uczących, to system może uczyć się poprawnie je interpretować poprzez indukcję odpowiednich klasyfikatorów, tworzących w konsekwencji opis gramatyki. Pomimo intensywnego rozwoju w ostatnich latach badań nad systemami klasyfikującymi (Lanzi i Riolo 2000), próby zastosowania systemów klasyfikujących w indukcji gramatyk są nieliczne. W tym miejscu należy wymienić pracę Bianchiego (Bianchi 1996), w której na podstawie eksperymentów przeprowadzonych na gramatykach nawiasowych, palindromach oraz prostej gramatyce języka naturalnego, wykazano wyższą efektywność systemu klasyfikującego od podejścia ewolucyjnego[41]. Cyre (2002) również indukował gramatykę dla podzbioru języka naturalnego, używając system LCS, jednak porównywanie uzyskanych przez niego wyników z innymi jest utrudnione, ze względu na wykorzystywanie przez niego korpusu językowego chronionego znakiem towarowym (opis patentów amerykańskich na sterowniki DMA).

Przedmiotem wieloletnich już, intensywnych badań autora niniejszej monografii[42] jest adaptacyjny model parsera języka naturalnego oraz języków formalnych. Indukcja parsera, sterowana przez algorytmy ewolucyjne, odbywa się w środowisku zdań poprawnych, czyli należących do poszukiwanej gramatyki oraz zdań niepoprawnych. Badane były parsery konstruowane w oparciu o model automatu ze stosami, tablicowe reprezentacje reguł gramatyki bezkontekstowej, wreszcie mechanizmy stosowane w systemach klasyfikujących.

---

[39] Skrótem LCS przyjęto oznaczać pierwszą wersję uczących się systemów klasyfikujących zaproponowaną przez Hollanda (*Holland-style classifier system*).

[40] W (Wierzchoń 2001) znajduje się zestawienie cech wspólnych systemu klasyfikującego z systemem immunologicznym.

[41] Należy zauważyć, że co prawda systemy klasyfikujące są najczęściej traktowane jako przykład genetycznych systemów uczących się (*genetic-based machine learning*, GMBL) (Goldberg 1989), to jednak w literaturze pojawia się coraz większa liczby systemów klasyfikujących, w których odkrywanie nowych reguł odbywa się bez udziału przetwarzania ewolucyjnego, jak chociażby w systemie ACS (Stolzmann 2000). Kovacs proponuje zatem, by LCS w zależności od implementacji postrzegać jako odmianę systemu genetycznego bądź systemu uczącego się ze wzmocnieniem (Kovacs 2002b).

[42] Innym obszarem badawczym leżącym w kręgu zainteresowań autora jest modelowanie obliczeń biomolekularnych (DNA *computing*) (Unold i in. 2003, Unold i Troć 2003, Unold i in. 2004, Unold i Troć 2005) oraz morfologia matematyczna (Köppen i in. 2001).

Adaptacja parsera automatowego polega na ewolucyjnym poszukiwaniu grafu przejść automatu, zdolnego do odpowiedniej akceptacji bądź też odrzucania zdań ze zbioru uczącego. Parser implementowany jest w oparciu o pewien niedeterministyczny model automatu ze stosami PDAMS (*nondeterministic PushDown Automaton with associative Memory accesS*)[43] (Unold 1992, 1994, 1995a, 1996a, 1997b, 1998a), rozszerzany również na model rozmytego automatu ze stosami fPDMAS (*fuzzy* PDAMS) (Unold 1997c, 1998b, 1998c, 1999a). Ewolucja grafu przejść może odbywać się zgodnie z zasadami ewolucyjnego programowania lub też programowania genetycznego (Unold 1999d, 2000). W tym drugim przypadku konieczne jest zastosowanie kodowania pośredniego, umożliwiającego ewolucję struktur drzewiastych, stanowiących swoisty język konstrukcji grafu przejść automatu (Unold i Dulewicz 2002, 2004). Ewoluowany automat działa w stratyfikacyjnym modelu reprezentacji wiedzy, w którym poszczególne jednostki językowe (słowa, grupy słów, zdania) są położone w węzłach wielowarstwowej sieci semantycznej (Unold 1996b, 1997a).

Poszukiwana gramatyka może być reprezentowana bezpośrednio jako tablica reguł gramatyki bezkontekstowej[44]. W (Chrobak i Unold 2000a, 2000b, 2001) gramatyka bezkontekstowa reprezentowana była przez tablicę o zadanych z góry wymiarach, odpowiadających liczbie produkcji wyprowadzanych z jednego nieterminala (kolumny) oraz liczbie stosowanych nieterminali (wiersze). Postać gramatyki ograniczono do postaci normalnej Greibach. Ze względu na statyczną reprezentację, wprowadzony został tzw. symbol nieważny, który oznaczał niewykorzystywane miejsce po prawej stronie produkcji. Symbol ten został następnie użyty w dedykowanych operatorach genetycznych. Ewolucji podlegała populacja gramatyk reprezentowanych przez tablice-chromosomy. Eksperymenty przeprowadzono na zbiorach zdań języków: angielskiego i niemieckiego – jako przykładach języków o względnie stałej i uporządkowanej składni zdania oraz polskiego – jako przykładowym języku o bardzo luźnym reżimie składni zdania. Zdania trenujące były kompletowane na zasadzie losowego wyboru zdań z podręczników do nauki danych języków na poziomie podstawowym. Uzyskano ciekawe wyniki, wskazujące m.in. na podobną „genetyczną podatność" gramatyk języków angielskiego i polskiego. W przypadku języka niemieckiego zaobserwowano wyraźną tendencję wzrostową jakości generowanych gramatyk. Należy

[43] Automat ze stosami równoważny jest co najmniej gramatyce bezkontekstowej (Hopcroft i Ullman 1979).

[44] Pytanie o miejsce języków naturalnych w hierarchii Chomsky'ego jest wciąż otwarte. Przede wszystkim języki naturalne są nieskończone, gdyż trudno sobie wyobrazić skończoną listę wszystkich zdań języka chociażby polskiego czy też angielskiego. Konstrukcje warunkowe typu „jeżeli..., to..." mogą reprezentować odległe zależności językowe, zatem do ich opisu z pewnością nie wystarczą gramatyki skończone. Zdania w konstrukcjach warunkowych mogą być dowolnie długie, a same konstrukcje mogą być wielokrotnie zagnieżdżone. Sytuacja taka może być reprezentowana przez wyrażenie 0^n1^n, którego opis wymaga już gramatyki bezkontekstowej ($S \rightarrow 0S1$, $S \rightarrow \varepsilon$, gdzie ε to tzw. symbol pusty). Trwa zatem debata, czy gramatyka bezkontekstowa wystarczy do opisu fenomenów języków naturalnych (Pullum i Gazdar 1982, Shieber 1985).

jednak pamiętać o silnym wpływie zastosowanej statycznej reprezentacji gramatyki na otrzymane wyniki.

Reprezentacja tablicowa gramatyki bezkontekstowej jest przykładem implementacji tzw. paradygmatu uczenia Pittsburgh (Pitt), w którym osobnik podlegający ewolucji reprezentuje cały zestaw reguł (gramatykę), a ocena pojedynczej reguły (produkcji CFG) jest nieistotna (Smith 1980, 1983)[45]. Alternatywnym paradygmatem uczenia jest tzw. podejście Michigan, w którym osobnik reprezentuje pojedynczą regułę, a populacja jest zbiorem konkurujących ze sobą reguł (Holland i Retman 1978, Booker 1982). Pierwszą udaną implementacją paradygmatu Michigan był system klasyfikujący Hollanda LCS (Holland i Reitman 1978). W (Unold i Dąbrowski 2003a, 2003b) LCS został zaadaptowany do zadania indukcji gramatyki bezkontekstowej. Funkcję klasyfikatorów systemu pełni lista produkcji gramatyk bezkontekstowych, pogrupowanych w populację gramatyk. Każdy klasyfikator jest postaci: *produkcja_gramatyki*: *komunikat*, gdzie *produkcja_gramatyki* to jedna z produkcji danej gramatyki, a *komunikat* oznacza komunikat wysyłany do kolejki komunikatów w przypadku zadziałania klasyfikatora. Każda produkcja jest w postaci normalnej Greibach. Aktywacja klasyfikatora następuje wtedy, gdy produkcja jest zdolna do częściowej lub całkowitej analizy aktualnego fragmentu zdania. Każdemu klasyfikatorowi jest przyporządkowana pewna „siła". Jej zadaniem jest określanie przydatności produkcji/klasyfikatora dla danej gramatyki. Początkowo wszystkie reguły mają przyporządkowane takie same siły, jednak w procesie analizy każdy klasyfikator jest nagradzany bądź karany za podejmowane akcje zgodnie z algorytmem „kubełkowym" przyznawania nagród (*bucket brigade algorithm*) (Holland 1986). Każde zdanie języka naturalnego jest analizowane po kolei przez całą populację gramatyk. Początkowa postać gramatyk używanych przez system jest generowana losowo. Kolejka komunikatów pełni funkcję pamięci systemu, w której przechowywane są efekty zadziałania poszczególnych klasyfikatorów. Pierwszym elementem kolejki komunikatów w systemie jest zawsze zdanie dostarczane przez środowisko systemu. Na kolejnych miejscach kolejki umieszczane są nieprzeanalizowane jeszcze w danym kroku analizy fragmenty zdania. Komunikaty znajdujące się w kolejce są kolejno poddawane działaniu klasyfikatorów. Te klasyfikatory, dla których spełniony jest warunek dopasowania produkcji do zdania, mają prawo do umieszczenia w kolejce własnych komunikatów. Kolejka jest tworzona do momentu, aż żaden klasyfikator nie jest w stanie odpalić swojej akcji, tzn. żadna produkcja nie pasuje do aktualnie analizowanego fragmentu zdania. Każdy klasyfikator umieszczający swój komunikat w kolejce jest zobowiązany do wpłacenia umownej opłaty, stanowiącej pewien procent jego siły. Opłata ta jest następnie przekazywana do klasyfikatora (lub rozdzielana pomiędzy większą liczbę klasyfikatorów), który

[45] W istocie bardzo często ocena osobnika, czyli zestawu reguł, jest funkcją ocen pojedynczych reguł. W (Chrobak i Unold 2000b) ocena pojedynczych reguł (produkcji) miała wpływ na działanie specjalizowanego operatora mutacji, nie wpływała natomiast na ocenę całej gramatyki.

umieścił w kolejce komunikat będący przyczyną odpalenia akcji aktualnej reguły. Klasyfikator, który kończy analizę zdania, otrzymuje nagrodę od środowiska systemu. Ponadto wszystkie klasyfikatory biorące udział w pełnej i częściowej analizie danego zdania mogą być dodatkowo nagradzane ustaloną sumą punktów siły. Opisany algorytm jest prawdziwy w przypadku zdań poprawnych. Dla zdań niepoprawnych jedyną modyfikacją w stosunku do powyższego mechanizmu są ujemne wartości „nagród". Algorytm genetyczny traktuje poszczególne gramatyki jako chromosomy, genami są w tym przypadku produkcje gramatyk. Krzyżowanie i selekcja jest przeprowadzana w odniesieniu do całych gramatyk (traktowanych jako wektory produkcji), natomiast mutacja powoduje modyfikowanie (dodawanie, usuwanie, zamianę) pojedynczych symboli poszczególnych produkcji. System został przetestowany na palindromach oraz izolowanych zdaniach języka angielskiego, osiągając sprawność 90% akceptacji zdań pozytywnych języka naturalnego przez wyewoluowaną gramatykę.

Podejście Michigan zostało zastosowane również w zupełnie nowym modelu systemu klasyfikującego GCS (*Grammar-based Classifier System*), którego wybrane charakterystyki można znaleźć w pracach autora (Unold i Cielecki 2005a, Unold 2005a, Unold 2005b, Unold 2005c, Unold i Cielecki 2005b, Unold 2005d, Unold 2005e). Nazwa systemu określa zarówno jego architekturę, w której wiedza o otoczeniu reprezentowana jest w postaci produkcji gramatyki, jak i zasadnicze przeznaczenie, jakim jest automatyczna indukcja gramatyki bezkontekstowej. Zasada pracy systemu GCS jest podobna do klasycznego LCS pracującego w reżimie zadania wieloetapowego, ale różni się od niego reprezentacją populacji klasyfikatorów, metodą dopasowywania klasyfikatorów do stanu środowiskowego, metodami odkrywania nowych klasyfikatorów i wreszcie środowiskiem. Zaproponowany przez autora nowy model systemu klasyfikującego jest przedmiotem następnych rozdziałów monografii.

1.5.7. Algorytm CYK

Indukcja gramatyki bezkontekstowej, niezależnie od przyjętego modelu uczenia, a w ramach modelu zastosowanej metody inferencji, wymusza odpowiedź na tzw. pytanie o przynależność (*membership query*): dla danej gramatyki bezkontekstowej $G = (N, T, P, S)$ i łańcucha $x \in T^*$, czy $x \in L(G)$? W literaturze znanych jest kilka algorytmów udzielających odpowiedzi na powyższe pytanie, złożoność obliczeniowa najszybszych[46] jest proporcjonalna do sześcianu długości łańcucha x (Hopcroft i Ullman 1979). Najbardziej znanymi metodami testowania należenia łańcucha do gramatyki CFG

[46] Teoretycznie najszybszym asymptotycznie algorytmem parsowania bezkontekstowego jest algorytm Valianta (1975), który redukuje zadanie parsowania do problemu mnożenia macierzy. Najlepszy znany obecnie algorytm mnożenia macierzy wymaga $\Theta(n^{2,376})$ kroków, jednak praktyczne stosowanie metody Valianta nie jest już tak efektywne (Lee 2002).

są algorytm CYK (Kasami 1965, Younger 1967) oraz algorytm Earleya (Earley 1970). Złożoność obliczeniowa obydwu algorytmów jest $\Theta(gn^3)$, gdzie g jest rozmiarem CFG, a n długością łańcuchów (Graham i in. 1980).

Algorytm Cocke'a–Yongera–Kasamiego, znany jako algorytm CYK[47], opiera się na metodzie programowania dynamicznego i wymaga podania gramatyki bezkontekstowej w postaci normalnej Chomsky'ego[48]. Działanie algorytmu polega na sprawdzaniu, czy dla dowolnego symbolu nieterminalnego A istnieje wyprowadzenie $A \xrightarrow{*} x_{ij}$, gdzie x_{ij} jest podłańcuchem o długości j, rozpoczynającym się od i-tej pozycji łańcucha x. Jeżeli takie wyprowadzenie istnieje, to algorytm zapamiętuje, że z danego symbolu nieterminalnego można wyprowadzić określony podłańcuch. Stosując indukcję po j, czyli rozpatrując coraz dłuższe podłańcuchy, zastępujemy w każdym kroku prawe strony wyprowadzeń symbolami ze strony lewej. Po osiągnięciu $j = n$ możemy rozstrzygnąć, czy $S \xrightarrow{*} x_{1j}$. Ponieważ $x_{1n} = x$, zatem $x \in L(G)$ wtedy i tylko wtedy, gdy $S \xrightarrow{*} x_{1n}$.

Poniżej przedstawiono formalny zapis algorytmu CYK, posługując się notacją zaczerpniętą z pracy (Hopcroft i Ullman 1979). Symbol V_{ij} oznacza zbiór symboli nieterminalnych A, dla których $A \xrightarrow{*} x_{ij}$:

```
procedure CYK
begin
   for i := 1 to n do
      V_i1 := {A | A → a jest produkcją, a jest i-tym symbolem łańcucha x};
   for j := 2 to n do
      for i := 1 to n – j + 1 do
      begin
            V_ij := 0;
            for k := 1 to j – 1 do
            V_ij := V_ij ∪ {A | A → BC jest produkcją, B ∈ V_ik i C ∈ V_i+k,j–k }
      end
end.
```

Algorytm CYK zapisuje wyniki swego działania w trójkątnej tablicy o wymiarach $n \times n$, gdzie n jest długością analizowanego łańcucha x. Jeżeli w komórce tablicy o współrzędnych i, j znajdzie się symbol nieterminalny A, oznacza to, że istnieje wy-

[47] W literaturze anglojęzycznej spotyka się też skrót CKY (Lee 2002).

[48] Ograniczenie gramatyki do PNC nie zawęża zakresu zastosowania algorytmu CYK, gdyż znane jest twierdzenie mówiące, że każdą gramatykę bezkontekstową niegenerującą zdania pustego, można przekształcić do postaci PNC (Hopcroft i Ullman 1979).

prowadzenie $A \xrightarrow{*} x_{ij}$, gdzie x_{ij} jest podłańcuchem o długości j, rozpoczynającym się od i-tej pozycji łańcucha x. Wszystkie symbole z komórki $[i, j]$ tworzą zbiór V_{ij}. Tablica algorytmu CYK wypełniana jest zgodnie z rosnącym indeksem j, a zatem wzdłuż kolejnych wierszy tablicy, z których każdy reprezentuje podciągi łańcucha x o długości j. Podczas wypełniania pierwszego wiersza tablicy wyszukiwane są w gramatyce produkcje przepisujące symbole terminalne typu $A \rightarrow a$. Wstawienie symbolu nieterminalnego A do komórki tablicy kolejnych wierszy jest możliwe tylko wtedy, gdy w zbiorze produkcji gramatyki istnieje taka produkcja $A \rightarrow BC$, gdzie B wyprowadza pierwszą część podciągu, a C część drugą. Symbole B oraz C są poszukiwane w odpowiednich komórkach tablicy CYK, reprezentujących możliwe odpowiednio pierwsze i drugie części wyprowadzanego podciągu. Zapis symbolu startowego gramatyki S do komórki $[1, n]$ oznacza pozytywną odpowiedź na pytanie o przynależność łańcucha x do $L(G)$. Jednocześnie, wypełniona tablica CYK zawiera wszystkie możliwe rozbiory analizowanego łańcucha.

Jako przykład działania algorytmu CYK rozważmy następującą gramatykę bezkontekstową:

$G = (\{A, B, C, S\}, \{a, b, c\}, \{S \rightarrow AB, S \rightarrow AC, C \rightarrow SB, C \rightarrow a, B \rightarrow BB, B \rightarrow b, A \rightarrow a\}, S)$

oraz łańcuch wejściowy *aabb*. Tablicę algorytmu CYK pokazano na rys. 1.

		a	*a*	*b*	*b*
			$i \rightarrow$		
		1	2	3	4
	1	*A, C*	*A, C*	*B*	*B*
$j\downarrow$	2	*S*	*S*	*B*	
	3	*C*	*C, S*		
	4	*C, S*			

Rys. 1. Wypełniona tablica CYK dla łańcucha *aabb*
Fig.1. CYK table filled for the input string *aabb*

Przykładowo, aby wypełnić komórkę [2, 3], należy znaleźć wszystkie symbole nieterminalne, które wyprowadzają podciąg łańcucha wejściowego rozpoczynający się od pozycji numer 2 i długości 3 (podciąg *abb*). Dzielimy podciąg na dwie części. Część pierwsza podciągu może zaczynać się od pozycji 2 i mieć długość 1 lub 2, części drugie muszą być uzupełnieniem części pierwszej do ciągu o długości 3. Rozpoczynamy od analizy $V_{21} = \{A, C\}$ oraz $V_{32} = \{B\}$. Możliwymi prawymi stronami produkcji reguły V_{21} V_{32} są AB i CB. W zbiorze produkcji gramatyki istnieje reguła $S \rightarrow AB$, a zatem do komórki [2, 3] (zbioru V_{23}) wpisujemy symbol S. Następnie analizujemy drugi z możliwych podziałów ciągu, czyli $V_{22} = \{S\}$ oraz $V_{41} = \{B\}$. Ciąg SB jest prawą stroną produkcji $C \rightarrow SB$, a więc dodajemy symbol C do zbioru V_{23}. Obecność symbolu startowego S w komórce [1, 4] dowodzi przynależności łańcucha do gramatyki.

		a	*a*	*a*	*a*
			$i \rightarrow$		
		1	2	3	4
	1	*S*	*S*	*S*	*S*
$j\downarrow$	2	*S*	*S*	*S*	
	3	*S*, *S*	*S*, *S*		
	4	*S*, *S*, *S*, *S*, *S*			

Rys. 2. Tablica CYK z powielonymi symbolami nieterminalnymi
Fig. 2. CYK table with duplicate nonterminal symbols

Złożoność obliczeniowa algorytmu CYK może wzrosnąć do $\Theta(\exp n)$ w przypadku, gdy komórki tablicy będą zawierały powielone symbole nieterminalne. Na rysunku 2 zilustrowano zawartość tablicy CYK dla gramatyki $G = (\{S\}, \{a\}, \{S \rightarrow SS, S \rightarrow a\}, S)$ podczas rozbioru ciągu *aaaa*. Możliwy wykładniczy wzrost złożoności algorytmu CYK wynika z faktu, że liczba wszystkich możliwych struktur gramatycznych (alternatywnych rozbiorów) dla ciągu o długości n jest funkcją wykładniczą n (Sakakibara 2005).

2. Metody ewolucyjne

Pod pojęciem ewolucyjnych metod uczenia się rozumiemy systemy uczące się działające na zasadach, jakie można zaobserwować w ewolucji żywych organizmów[49]. Zazwyczaj do dziedziny ewolucyjnych algorytmów zalicza się następujące metody (Michalewicz 1996, Arabas 2001, Kwaśnicka i Spirydowicz 2004)[50]:

- programowanie ewolucyjne,
- strategie ewolucyjne,
- algorytmy genetyczne,
- programowanie genetyczne.

Wymienione wyżej metody były historycznie rozwijane przez niezależne grupy badaczy, obecnie jednak obserwuje się zacieranie granic między oddzielnymi jeszcze do niedawna metodami (De Jong 1998, Fogel i Michalewicz 1999)[51]. Cechą charakterystyczną metod ewolucyjnych jest stosowana terminologia, która wywodzi się wprost z genetyki i ewolucji. Każdy osobnik składa się z chromosomów[52], a chromosom z genów. Zestaw wszystkich genów osobnika tworzy jego genotyp. Zestaw zewnętrznych cech osobnika, z których każda może zależeć od jednego lub kilku genów, nazywany jest fenotypem osobnika (genotyp osobnika koduje jego fenotyp).

---

[49] Michalewicz proponuje stosowanie wspólnego terminu *programy ewolucyjne* (*evolution programs*) dla wszystkich metod korzystających z zasad ewolucji (Michalewicz 1996).

[50] Wielu autorów, jak chociażby Goldberg (1989) czy Michalewicz (1996), do metod algorytmów ewolucyjnych zalicza również *uczące się systemy klasyfikujące*. Jednak ich konstrukcja, odbiegająca w sposób istotny od budowy metod ewolucyjnych, a przede wszystkim fakt, że istnieją udane aplikacje systemów klasyfikujących, które nie stosują algorytmów ewolucyjnych, sugeruje odrębne ich traktowanie.

[51] Obecnie większość praktycznie wykorzystywanych ewolucyjnych metod uczenia się trudno zakwalifikować w sposób jednoznaczny do którejkolwiek z wymienionych metod. Powstają natomiast nowe techniki ewolucyjne, jak nieuporządkowane algorytmy genetyczne (*messy genetic algorithms*) (Goldberg i in. 1989), komórkowe algorytmy genetyczne (*cellular genetic algorithms*) (Gruau i in. 1996), hybrydowe strategie genetyczne (*hybrid genetic strategies*) (Guerra-Salcedo i in. 1999), połączenia sieci neuronowych i algorytmów genetycznych (Schaffer i in. 1992) czy wyspowy model algorytmów genetycznych (*island model genetic algorithms*) (Whitley i in. 1997).

[52] Najczęściej w metodach ewolucyjnych osobnik ma tylko jeden chromosom, tzw. *chromosom haploidalny*.

Wszystkie metody ewolucyjne symulują ewolucję osobników w zadanym środowisku, używając selekcji i reprodukcji. Podobnie jak w naturze, o przeżyciu i/lub pozostawieniu potomka decyduje jakość przystosowania osobnika do środowiska (*fitness*)[53]. Przystosowanie osobnika oceniane jest na podstawie jego fenotypu.

Ewolucyjne algorytmy poszukują rozwiązania zadanego problemu poprzez przetwarzanie populacji osobników, z których każdy koduje rozwiązanie zadania. Dla każdego osobnika (a ściślej mówiąc jego fenotypu) obliczana jest funkcja przystosowania – osobniki o większej wartości funkcji przystosowania mają większe szanse reprodukcji (co odpowiada procesowi eksploatacji). Potomstwo różni się od rodziców dzięki zastosowaniu operatorów genetycznych, z których najczęściej używa się operatora mutacji (*mutation*) działającego na jednym osobniku (tzw. reprodukcja aseksualna) oraz krzyżowania[54] (*crossover*) – operatora działającego na co najmniej dwóch osobnikach (tzw. reprodukcja seksualna). Obydwa operatory, poprzez zaburzenie osobników rodzicielskich, zapewniają niezbędną w metodzie eksplorację przestrzeni rozwiązań.

```
procedure Algorytm Ewolucyjny
begin
   inicjuj populację
   oceń populację
   while (not kryterium stopu) do
   begin
      wybierz osobniki do reprodukcji
      zastosuj operatory genetyczne
      oceń populację
      wybierz osobniki do następnej generacji
   end
end
```

Rys. 3. Pseudokod algorytmu ewolucyjnego
Fig. 3. Pseudocode of evolutionary algorithm

Na rysunku 3 przedstawiony został pseudokod typowego algorytmu ewolucyjnego. Po zainicjowaniu populacji osobników, następuje w pętli ewolucja osobników poprzez następujące po sobie: reprodukcję, operacje genetyczne, ocenę i sukcesję. Zadaniem

[53] Według Russella i Norviga (1995) algorytmy ewolucyjne są metodami uczenia ze wzmocnieniem, w którym potomek jest traktowany jako nagroda.

[54] Zdaniem Arabasa (2001) odpowiedniejszym tłumaczeniem słowa *crossover* jest *krosowanie*, gdyż termin *krzyżowanie* w biologii odnosi się do gatunków, odmian i ras, a nie do procesu molekularnego zachodzącego podczas płciowego rozmnażania. Z tego względu bardziej odpowiednim terminem zdaje się być *rekombinacja.*

reprodukcji jest wybór z populacji (bazowej) osobników rodzicielskich, które następnie poddawane są operacjom genetycznym. Uzyskana w ten sposób populacja (potomna) poddawana jest ocenie środowiska, po czym następuje wybór osobników, które przechodzą do następnej generacji algorytmu. Nowa populacja bazowa może składać się zarówno z populacji potomnej, jak i poprzedniej populacji bazowej.

Inicjacja populacji polega zazwyczaj na losowym wygenerowaniu zadanej liczby osobników, chociaż zaleca się, jeżeli jest taka możliwość, wykorzystanie wiedzy o problemie przy tworzeniu początkowej populacji, np. generując osobniki częściowo dostosowane do środowiska. Funkcją oceny może być zarówno prosta funkcja matematyczna, jak i obliczeniowo złożone, osobne zadanie obliczeniowe[55]. Pętla ewolucji może zakończyć się wówczas, gdy przystosowanie osobników osiągnie zadawalającą wartość, minie zadana liczba iteracji lub gdy stwierdzi się, że algorytm osiągnął stan stagnacji, czyli przystosowanie osobników nie jest zadawalające, ale też od dłuższego czasu nie zmienia się.

2.1. Programowanie ewolucyjne

Programowanie ewolucyjne (*evolutionary programming*) zostało zaproponowane w 1966 r. przez Lawrenca Fogela i współpracowników (Fogel i in. 1966) jako metoda ewolucji populacji automatów skończonych w celu odkrycia gramatyki nieznanego języka, gdy znane są etykietowane przykłady zdań. Funkcja przystosowania osobnika, czyli automatu skończonego reprezentującego poszukiwaną gramatykę, zliczała liczbę poprawnych klasyfikacji przykładów podanych na wejście automatu. Każdy osobnik w populacji generuje jednego potomka, używając wyłącznie mutacji, która może polegać na:

- dodaniu stanu automatu,
- usunięciu stanu,
- zmianie stanu początkowego,
- zmianie funkcji wyjść,
- zmianie funkcji przejść.

Podczas reprodukcji każdego osobnika populacja chwilowo składa się z podwojonej liczby osobników: rodziców i potomków. Do następnego pokolenia wybierane są osobniki z podwojonej populacji, ale tak, że zachowany jest stały rozmiar populacji w kolejnych generacjach. Oznacza to, że najlepszy osobnik zawsze przetrwa, czyli

[55] Przykładem osobnego zadania obliczeniowego może być liczenie funkcji dopasowania dla tak złożonych struktur, jak sieci neuronowe czy systemy rozmyte w inteligentnych systemach obliczeniowych (Rutkowska 1997), czy też obliczanie przystosowania w hybrydowych systemach harmonogramowania (Pawlak 1999) lub w ewolucyjnych parserach języka naturalnego (Unold 2000).

znalezione raz optimum już nie jest gubione[56]. Wyboru dokonuje się zwykle z zastosowaniem stochastycznej turniejowej metody selekcji (*tournament selection*)[57].

Warto zauważyć, że w programowaniu ewolucyjnym nie stosuje się żadnego kodowania rozwiązań, a reprezentacja wynika z rozwiązywanego problemu. Poza tym, siła mutacji – jedynego w tej metodzie operatora genetycznego, często bywa ograniczana w miarę zbliżania się do optimum globalnego[58].

2.2. Strategie ewolucyjne

Również w latach sześćdziesiątych ubiegłego wieku, dwaj studenci Politechniki Berlińskiej Ingo Rechenberg i Hans-Paul Schwefel, podczas poszukiwania metody optymalizacji kształtów ciał w przepływach, wpadli na pomysł strategii ewolucyjnych (*evolutionary strategy*) (Rechenberg 1973, Schwefel 1977). Podobnie jak ma to miejsce w programowaniu ewolucyjnym, ta metoda ewolucyjna również nie stosuje żadnego kodowania osobników, używając najczęściej wektora liczb rzeczywistych. Genotyp osobnika wzbogacony jest o drugi chromosom zawierający wektor standardowych odchyleń wykorzystywany podczas obliczania mutacji[59]. Pierwotny algorytm, nazwany strategią (1 + 1), opierał się na schemacie mutacji i selekcji z wykorzystaniem tylko jednego osobnika. Jeżeli po mutacji potomek okazywał się lepiej przystosowany do środowiska, to on przechodził do następnej generacji, jeżeli lepszym był rodzic, to pozostawał w kolejnym pokoleniu. W algorytmie tym zastosowano mechanizm adaptacji zasięgu mutacji, zwany regułą 1/5, według której na pięć mutacji jedna powinna kończyć się sukcesem. Ponieważ strategia (1 + 1) przejawia niewielką odporność na lokalne minima, wprowadzono strategie wieloelementowe, w których ewoluuje się populację μ-elementową, gdzie $\mu > 1$ [60]. Poza rozszerzeniem liczby przetwarzanych w jednej generacji osobników w strategiach wieloelementowych wprowadzono krzyżowanie. W strategii zwanej $(\mu + \lambda)$ populacja licząca μ elementów produkuje w jednej generacji λ po-

[56] Jest to tzw. *sukcesja elitarna* (*elitism*).

[57] W turniejowej metodzie selekcji stosuje się dwustopniową selekcję osobników. W każdym kroku reprodukcji wybiera się n osobników z populacji składającej się z m osobników ($m \geq n$). Wszystkie n elementowe kombinacje z populacji są jednakowo prawdopodobne. Następnie przeprowadza się turniej między wybranymi n osobnikami, wygrywa osobnik o największym przystosowaniu. Proces wyboru i turnieju przeprowadza się wielokrotnie, aż do wypełnienia populacji następnego pokolenia.

[58] Choć warto zauważyć za (Pawlak 1999) pewną tautologię: ogranicza się siłę działania mutacji przy zbliżaniu się do optimum globalnego w sytuacji, gdy w istocie nie znamy jeszcze owego optimum.

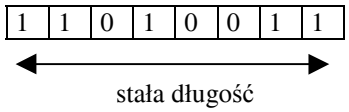
[59] W tym miejscu warto wspomnieć o pracach Davida Fogela, syna Lawranca, który w latach osiemdziesiątych ubiegłego wieku „odmłodził" programowanie ewolucyjne, między innymi przez wprowadzenie takich mechanizmów wziętych ze strategii ewolucyjnych, jak reprezentacja rzeczywistoliczbowa oraz drugi chromosom zawierający wektor zmiennych określających strategię przeszukiwań.

[60] Liczności ewoluowanej populacji przyjęto oznaczać symbolem μ. Może powodować to pewne nieporozumienia, gdyż często ten symbol w algorytmach ewolucyjnych oznacza mutację.

tomków. Do następnej populacji wybieranych jest μ osobników ze złączonej populacji rodziców i potomków o liczności $\mu + \lambda$. W tej strategii zrezygnowano z reguły 1/5 na korzyść samoczynnej adaptacji mutacji, będącej wynikiem mechanizmu selekcji. W strategii (μ, λ) (gdzie $\lambda > \mu$) nowa populacja tworzona jest jedynie na podstawie λ potomków[61]. W strategiach wieloelementowych krzyżowanie może przebiegać z wykorzystaniem dwóch osobników lub rodzicami potomka mogą być wszystkie osobniki w populacji.

2.3. Algorytmy genetyczne

Chyba najpopularniejszymi algorytmami ewolucyjnymi są algorytmy genetyczne (*genetic algorithms*) Johna Hollanda (1975), spopularyzowane przez Davida Goldberga (1989). W ich pierwotnej postaci każdy osobnik, niezależnie od rozwiązywanego problemu, jest kodowany przez łańcuch znaków binarnych 0 i 1, a chromosomy mają stałą długość (rys. 4). Na poziomie genotypu, czyli łańcucha znaków binarnych, realizowane są operatory genetyczne krzyżowania i mutacji, z których dominującym jest rekombinacja.



Rys. 4. Binarna reprezentacja genomu w algorytmie genetycznym
Fig. 4. Binary representation of genome in genetic algorithm

Najpowszechniej stosowanym typem krzyżowana jest tzw. krzyżowanie jednopunktowe (*one-point crossover*), rys. 5. Operacja ta polega na losowym wyborze punktu przecięcia w osobnikach rodzicielskich, a następnie wymianie genów leżących za punktem przecięcia pomiędzy rodzicami.

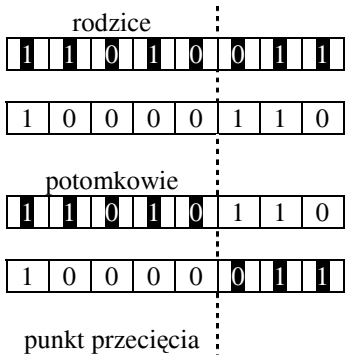
Inną charakterystyczną cechą standardowego algorytmu genetycznego jest reprodukcja proporcjonalna do przystosowania (*fitness-proportional selection*), zwana inaczej metodą ruletki (*roulette wheel selection*). W metodzie tej każdemu osobnikowi przydziela się wycinek koła ruletki o polu powierzchni proporcjonalnym do wartości funkcji przystosowania osobnika. Całe koło ruletki odpowiada sumie wartości funkcji przystosowania wszystkich osobników w populacji.

Prawdopodobieństwo wyboru i-tego osobnika z populacji o rozmiarze n można wyrazić wzorem

[61] W strategii (μ, λ) każdy osobnik żyje tylko jedno pokolenie, strategia $(\mu + \lambda)$ pozwalała na dowolnie długie życie osobnika w pętli ewolucyjnej.

$$p_i = \frac{przystosowanie_i}{\sum_{i=1}^{n} przystosowanie_i}, \tag{15}$$

w którym $przystosowanie_i$ oznacza wartość funkcji przystosowania i-tego osobnika w populacji, a mianownik jest sumą przystosowań wszystkich osobników w populacji. Selekcja osobnika może być interpretowana jako obrót kołem ruletki, w wyniku którego wybrany zostaje osobnik należący do wylosowanego wycinka koła. Populacja potomna jest tworzona w wyniku n-krotnego uruchomienia koła ruletki. Oczywiście osobnicy o mniejszej wartości funkcji przystosowania mają mniejsze szanse na wylosowanie, osobnicy o większej wartości przystosowania, czyli należący do większego wycinka koła, są statystycznie wybierani częściej.



Rys. 5. Krzyżowanie jednopunktowe w algorytmie genetycznym
Fig. 5. One-point crossover in genetic algorithm

Ważne własności algorytmów genetycznych zostały wyprowadzone przez ich twórcę w oparciu o teorię schematów. W najogólniejszym rozumieniu schemat (*schema*) jest pewnym szablonem pozwalającym badać podobieństwa osobników reprezentowanych przez łańcuchy binarne. Ze względu na przyjęte w teorii schematów silne ograniczenia, takie jak binarna reprezentacja chromosomu, charakterystyczne operatory genetyczne czy nieskończona populacja, jej praktyczna użyteczność jest niewielka, a do analizy działania algorytmu genetycznego stosuje się teorię procesów Markowa (Nix i Vose 1992).

2.4. Programowanie genetyczne

Najmłodszą z wymienionych w poprzednich punktach metod ewolucyjnych jest programowanie genetyczne (*genetic progamming*), zaproponowane przez Johna Koza

na początku lat dziewięćdziesiątych ubiegłego wieku (Koza 1992). Metoda ta została stworzona w celu automatycznego tworzenia programu komputerowego poprzez stopniową jego ewolucję. Osobnikiem jest program komputerowy zapisany w postaci struktury drzewiastej. Najczęściej stosuje się język programowania LISP, w którym zarówno program, jak i dane do programu pamiętane są właśnie w postaci drzewa. Węzły drzewa reprezentują bądź to funkcje, bądź operatory jedno- lub wieloargumentowe, w liściach umieszczane są symbole terminalne (np. wartości stałe, argumenty). Programowanie genetyczne jest w swej istocie rozszerzonym modelem algorytmu genetycznego, w którym zamiast kodowania binarnego stosuje się genom o strukturze drzewiastej. Ponieważ ewoluujące programy mają różną długość, więc reprezentują ich chromosomy o zmiennej długości, a operatory genetyczne są dostosowane do przetwarzanych struktur genotypowych. Krzyżowanie polega na wymianie całych poddrzew pomiędzy wybranymi osobnikami, mutacja może dotyczyć:

- losowej zmiany funkcji w węźle,
- losowej zmiany wartości liścia,
- zamiany poddrzewa na losowo wygenerowane,
- tzw. permutacji, czyli zamiany miejscami dwóch podrzew wychodzących z jednego węzła.

Współczesne zastosowanie programowania genetycznego nie ogranicza się do ewolucji programów komputerowych, ale również klasyfikacji i wyszukiwania tekstów, klasyfikacji protein, przetwarzania obrazów, projektowania obwodów elektrycznych, identyfikowania obiektów (Banzhaf i in. 1998). Jest to możliwe również dzięki odseparowaniu drzewa-genotypu od fenotypu, tj. szukanego rozwiązania[62].

[62] Na przykład w tzw. *kodowaniu krawędziowym* (*edge encoding*) (Luke i Spector 1996) lub *komórkowym* (*cellular encoding*) (Gruau 1992) drzewo stanowi metajęzyk umożliwiający rozkodowanie genotypu na dowolną inną strukturę, np. graf przejść parsera (Dulewicz i Unold 2004).

3. Uczący się system klasyfikujący

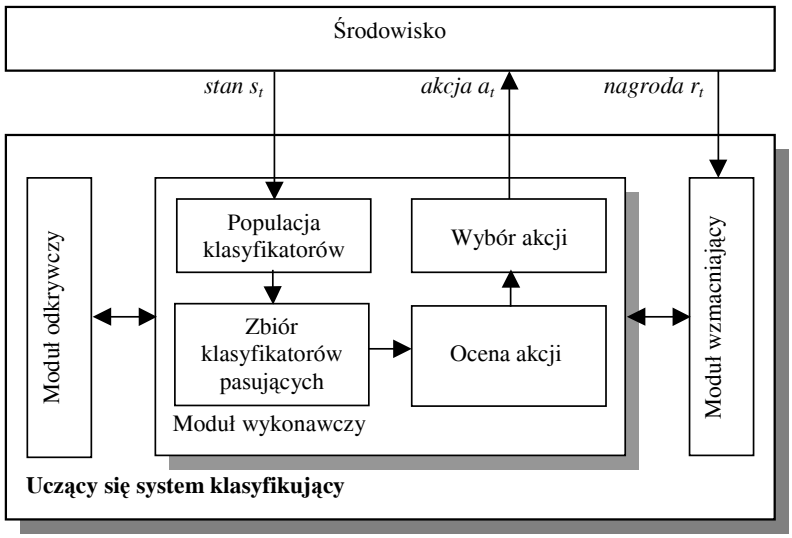
Uczący się system klasyfikujący może być zdefiniowany jako model zachowania prostego organizmu, który adaptuje się do otoczenia poprzez właściwe reagowanie na przychodzące bodźce. Zwrotne sprzężenie z otoczeniem jest możliwe dzięki zestawowi reguł typu „warunek-akcja", a adaptacja systemu polega na sprawdzaniu i poprawie efektywności istniejących reguł oraz generowaniu nowych reguł, dostosowanych do środowiska. Odkrywanie nowych reguł jest wynikiem indukcyjnego algorytmu uczenia, który jest swoistą kombinacją uczenia ze wzmocnieniem (Kaelbling i in. 1996) i przetwarzania ewolucyjnego.

3.1. Architektura

Uczący się system klasyfikujący uczy się poprzez interakcję z otoczeniem, od którego otrzymuje odpowiedzi w postaci numerycznej nagrody. Zasadą uczenia jest próba maksymalizacji otrzymywanych nagród. W chwili obecnej znanych jest kilka różnych modeli systemów LCS i w związku z tym funkcjonuje również kilka odmiennych definicji uczącego się systemu klasyfikującego. Pomijając w tej chwili nieistotne szczegóły, można przyjąć, że system klasyfikujący składa się z czterech zasadniczych modułów (rys. 6):

- populacji klasyfikatorów, reprezentujących bieżącą wiedzę systemu;
- modułu wykonawczego (*peformance component*), zarządzającego interakcją z otoczeniem;
- modułu wzmacniającego (*reinforcement component*), zwanego również modułem przyznawania ocen (*credit assignment component*), dystrybuującego otrzymaną od otoczenia nagrodę do klasyfikatorów, dzięki którym nagroda została przyznana;
- modułu odkrywczego (*discovery component*), odpowiedzialnego za poprawę istniejących klasyfikatorów oraz odkrywanie nowych, najczęściej wykorzystującego do tego celu algorytm genetyczny.

Do oceny przydatności klasyfikatorów służą najczęściej dwie miary: predykcja (*prediction*) oraz przystosowanie (*fitness*)[63]. Predykacja szacuje użyteczność klasyfikatora w systemie w funkcji wielkości nagrody, jaką system otrzyma po zastosowaniu danego klasyfikatora. Przystosowanie klasyfikatora estymuje jego jakość wiedzy o zadaniu i służy modułowi odkrywczemu w sterowaniu ewolucją populacji klasyfikatorów. Im lepsze przystosowanie klasyfikatora – czyli większa wiedza o rozwiązywanym problemie, tym większe szanse jego reprodukcji, jeżeli przystosowanie jest niewielkie, to obecność takiego klasyfikatora nie jest pożądana w systemie, a jego szanse reprodukcji powinny maleć.



Rys. 6. Uczący się system klasyfikujący w interakcji ze środowiskiem
Fig. 6. Learning classifier system with environment interaction

Zgodnie z rys. 6, w każdej dyskretnej chwili czasu t system klasyfikujący otrzymuje od otoczenia (za pomocą detektorów) jego aktualny stan s_t, na podstawie którego buduje zbiór klasyfikatorów pasujących (*match set*[64]), zawierający te klasyfikatory z populacji wszystkich klasyfikatorów, których warunek pasuje do stanu otoczenia. Następnie system ocenia użyteczność akcji ze zbioru klasyfikatorów pasujących. Na podstawie przyjętych kryteriów zostaje wybrana ze zbioru akcja a_t i przesłana jako odpowiedź systemu do otoczenia (za pośrednictwem efektorów). W zależności od stanu otoczenia s_t i podjętej akcji a_t, system otrzymuje nagrodę r_t. Następnie moduł wzmacniający dystrybuuje nagrodę r_t pomiędzy klasyfikatorami, które przysłużyły się

[63] W pierwotnym podejściu Hollanda ocenę klasyfikatora stanowi jego *siła* (*strength*), tożsama z miarą dopasowania.

[64] Kazimierz Grygiel, polski tłumacz książki Goldberga (Goldberg 1989), używa tłumaczenia *grupa zgodna*.

do jej otrzymania. Sam mechanizm dystrybucji nagrody może opierać się bądź to na specjalnie zaprojektowanym algorytmie dla systemów klasyfikujących, tzw. algorytmie kubełkowym, bądź na algorytmach stosowanych w uczeniu ze wzmocnieniem, jak w modyfikowanej wersji *Q-learning* (Wilson 1995). W klasycznej, czyli najstarszej, wersji uczącego się systemu klasyfikującego moduł odkrywczy jest algorytmem genetycznym, który traktuje klasyfikatory jak osobniki mogące podlegać standardowym operatorom genetycznym.

Otoczenie systemu klasyfikującego definiuje jego zasadnicze zadanie. W przypadku zadań klasyfikacyjnych, np. (Holmes 1996, Unold i Dąbrowski 2003a), otoczeniem systemu jest zbiór uczący zawierający etykietowane przykłady. Każdy przykład jest opisany przez wektor atrybutów oraz jednoznaczne przypisanie do określonej klasy. Celem uczenia jest otrzymanie takich klasyfikatorów, które będą w stanie z odpowiednią dokładnością sklasyfikować nieznane przykłady. W sterowaniu autonomicznych robotów otoczeniem jest fizyczne środowisko robota, a zadaniem systemu jest nauczenie określonego zachowania, np. (Katagami i Hamada 2000). W przypadku zastosowania systemu klasyfikacyjnego w ekonomii obliczeniowej otoczeniem jest rynek, a celem uczenia może być opracowanie reguł zwiększających zyski (Judd i Tesfatsion 2005).

3.2. Klasyfikacja

Zainteresowanie uczącymi się systemami klasyfikującymi trwa nieprzerwanie od momentu ich powstania, czyli obecnie niemal od trzech dziesięcioleci. Wydaje się, że jest ono spowodowane m.in. atrakcyjnością dosyć swobodnej interpretacji populacji klasyfikatorów reprezentujących wiedzę w systemie. Przykładowo, osobnikiem podlegającym operacjom genetycznym może być pojedynczy klasyfikator albo cały zbiór klasyfikatorów, system klasyfikujący można traktować jako system o zmiennym (ewolucyjnym) kodzie wykonywalnego programu, wreszcie ogólna definicja klasyfikatora nie narzuca metody jego reprezentacji.

Model zaproponowany przez Johna Hollanda w latach siedemdziesiątych ubiegłego wieku doczekał się wielu, często diametralnie różnych, implementacji, dlatego też próbując usystematyzować wiedzę na ich temat, można dokonać kategoryzacji uczących się systemów klasyfikujących wg poniższych kryteriów[65]:

- zasięgu chromosomu,
- miary użyteczności klasyfikatora,
- metody reprezentacji klasyfikatora,
- mechanizmu stosowanego w module odkrywczym,
- pamięci systemu.

[65] Zaproponowana klasyfikacja uczących się systemów klasyfikujących jest propozycją autora monografii.

3.2.1. Kryterium zasięgu chromosomu

W pierwszym zrealizowanym systemie klasyfikującym CS-1 (Holland i Reitman 1978), zaprogramowanym w Fortranie na maszynie IBM 1800 należącej do Uniwersytetu w Michigan, przedmiotem operacji genetycznych były indywidualne klasyfikatory. Wszystkie klasyfikatory systemu tworzą populację osobników-chromosomów ewoluujących w czasie. W każdym kolejnym kroku obliczeniowym algorytm oceny określa wartość poszczególnych klasyfikatorów, a algorytm genetyczny stanowiący podstawę modułu odkrywczego operuje na pojedynczym chromosomie-klasyfikatorze. Zgodnie z mechanizmem ewolucyjnym klasyfikatory słabe mają mniejsze szanse na reprodukcję, nowe klasyfikatory tworzone są z najlepszych.

Niemalże w tym samym czasie na Uniwersytecie w Pittsburgu powstał program LS-1 (Smith 1980), którego architektura w sposób istotny różniła się od propozycji Hollanda i Reitmana. W propozycji Smitha chromosomami podlegającymi operacjom genetycznym nie są pojedyncze klasyfikatory, lecz ich zestawy. W takiej sytuacji praktycznie nie ma potrzeby oceny pojedynczej reguły i stosowania specjalizowanych algorytmów dystrybucji otrzymanej ze środowiska nagrody, gdyż wystarcza miara przydatności całego zestawu reguł.

Systemy klasyfikujące, w których zasięg chromosomu ogranicza się do pojedynczego klasyfikatora, przyjęto nazywać podejściem Michigan (*Michigan approach*), natomiast systemy, w których zasięg chromosomu obejmuje grupę klasyfikatorów, podejściem Pitt (*Pitt approach*).

Porównując działanie obydwu typów systemów klasyfikujących, należy stwierdzić, że podejście Pitt jest bliższe idei metod ewolucyjnych. W przetwarzaniu ewolucyjnym poszukuje się najlepszego osobnika z populacji możliwych rozwiązań, a przecież rozwiązanie stawianego systemowi klasyfikującemu zadania reprezentowane jest nie przez pojedynczy klasyfikator, ale poprzez cały ich zestaw. Praktycznie dopiero pojawienie się systemu XCS Wilsona (1995, 1998), który używa niszowania populacji[66], umożliwiło systemom klasyfikującym stosującym metodę Michigan znalezienie równowagi pomiędzy współzawodnictwem klasyfikatorów a ich współdziałaniem. Rezygnacja z przyznawania indywidualnych ocen klasyfikatorów w podejściu Pitt powoduje jednak istotne spowolnienie efektów uczenia, ponieważ otrzymywana od otoczenia nagroda jest „wysokopoziomowa" i dotyczy całej grupy reguł. Również stosowana operacja krzyżowania musi analizować sensowność tworzonych klasyfikatorów. Może się bowiem zdarzyć, że podczas rozcinania chromosomów, z których każdy składa się z całego zestawu klasyfikatorów, nastąpi przecięcie chromosomu na przykład na granicy klasyfikatora.

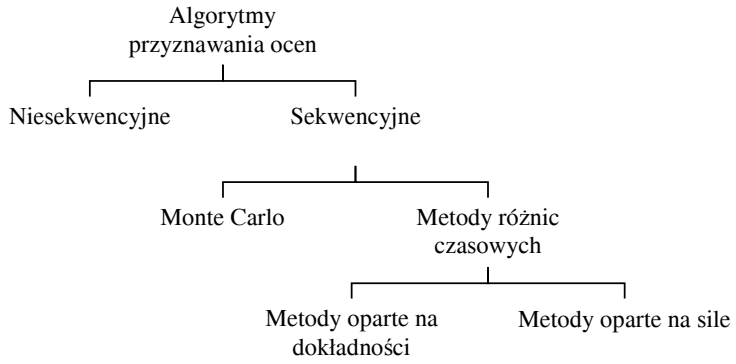
[66] W podejściu Michigan wszystkie klasyfikatory konkurują ze sobą podczas selekcji. Prowadzi to do pozostawienia w populacji jedynie najlepszych osobników, o podobnie wysokiej funkcji przystosowania, co nie zawsze jest pożądane. Zapobiega się temu zjawisku między innymi przez zmniejszenie zasięgu selekcji. W tym celu w całej populacji wyodrębnia się podpopulacje, tzw. *nisze*, w ramach których odbywa się konkurencja osobników.

W takiej sytuacji również i drugi rodzic-chromosom powinien być rozcinany na granicy reguły. Problemy może także stwarzać operowanie na chromosomach różnej długości, czyli różnej liczbie klasyfikatorów w zestawie. Podejście Pitt stosuje się obecnie w tych zadaniach, w których poszukuje się kooperujących populacji klasyfikatorów (Carse i Fogarty 1994, Nagasaka i Taura 1997).

Warto też w tym miejscu wspomnieć o systemach RUDI Grefenstetta (1988) oraz jego rozwinięciu SAMUEL (Grefenstette i in. 1990), w których zaproponowano wielopoziomowy algorytm przyznawania ocen, łącząc podejście Michigan i Pitt. Obydwie metody stosowane są również w hierarchicznym systemie klasyfikującym HCS (Shu i Schaeffer 1991).

3.2.2. Kryterium miary użyteczności klasyfikatora

W poprzednim podrozdziale rozróżniono systemy klasyfikujące według zasięgu chromosomu, dzieląc je na dwie grupy: podejście Michigan oraz podejście Pitt. Innym powszechnie uznanym kryterium podziału systemów klasyfikujących jest sposób oceny użyteczności[67] klasyfikatorów w populacji, który wiąże się z różnymi metodami oceny klasyfikatorów.



Rys. 7. Typy algorytmów przyznawania ocen, na podstawie (Kovacs 2002a)
Fig. 7. Types of rewards, adopted from (Kovacs 2002a)

Schematyczny podział algorytmów przyznawania ocen przedstawiono na rys. 7. W zadaniach sekwencyjnych wykorzystuje się najczęściej różne odmiany metod różnic czasowych (*temporal difference algorithm*) (Sutton 1988), które wchodzą w skład algorytmów uczenia ze wzmocnieniem. W tej grupie najczęściej wykorzystywane są różne odmiany algorytmu kubełkowego (Holland i in. 1986, Goldberg 1989), w któ-

[67] W kontekście tego rozdziału pojęcie *użyteczności* obejmuje określenia: przydatności, przystosowania, dopasowania, dokładności czy wreszcie siły.

rym mierzona jest siła klasyfikatora, oraz odmiany algorytmu *Q-learning* (Wilson 1994, 1995) operującego dokładnością klasyfikatora. Istnieją również udane realizacje uczących się systemów klasyfikujących, które wykorzystują uczenie typu Monte Carlo, przyjmujące w nomenklaturze systemów klasyfikujących nazwę schematu epokowego (*epochal schemes*). Do najbardziej znanych należą system CS-1 (Holland i Reitman 1978) oraz RUDI (Grefenstette 1988).

W zadaniach niesekwencyjnych, inaczej jednoetapowych, ocena klasyfikatora nie bierze pod uwagę nagrody, jaką otrzymać może system w przyszłości. Dlatego też algorytmy oceny klasyfikatorów mogą być prostsze od tych stosowanych w zadaniach sekwencyjnych, a wręcz mogą być traktowane jako ich uproszczone wersje (Kovacs 2002a).

3.2.3. Kryterium metody reprezentacji klasyfikatora

Sposób reprezentacji klasyfikatora w systemie klasyfikacyjnym ma istotny wpływ na działanie całego systemu. Do tej pory nie znaleziono reprezentacji idealnej, pasującej do wszystkich typów stawianych zadań. Reprezentacje proste, „kanoniczne”[68] nie mają odpowiednich własności generalizujących, reprezentacje o lepszych własnościach generalizujących są złożone, najczęściej dedykowane określonym zadaniom.

W kanonicznej reprezentacji klasyfikatorów używa się standardowego ternarnego języka LCS (*standard ternary* LCS *language*), w którym każda reguła ma jeden warunek i jedną akcję. Warunek reguły jest ciągiem o stałej długość, utworzonym z alfabetu {0, 1, #}, akcja oraz stan środowiska są ciągami również stałej długości, ale zbudowanymi z alfabetu {0, 1}.

Warunek reguły pasuje do otrzymywanego ze środowiska stanu (por. rys. 6) wtedy, gdy dla każdej pozycji ciągu reprezentującego stan i ciągu reprezentującego warunek występują identyczne litery alfabetu (zera albo jedynki) lub znak #. Znak ten oznacza zarówno 0, jak i 1 i jest środkiem, za pomocą którego osiąga się generalizację reguł. Na przykład 4-bitowy warunek 1#0# oznacza cztery stany środowiska, tj. 1100, 1101, 1000, 1001.

Klasyfikator jest zapisywany zwykle w postaci: *warunek* → *akcja*. Przykładowo, reguła 1001→1 oznacza następujące działanie: jeżeli na wejściu pojawi się sekwencja 1001, to podejmij akcję 1.

Ternarna reprezentacja klasyfikatorów miała zastosowanie na przestrzeni trzech dziesięcioleci w bardzo wielu systemach klasyfikujących, z których można wymienić te najbardziej znane, jak: Animat (Wilson 1985), BOOLE (Wilson 1987), SCS (Goldberg 1989), NEWBOOLE (Bonelli i in. 1990), ZCS (Wilson 1994), XCS (Wilson 1995) czy EpiCS (Holmes 1996).

[68] Pod pojęciem kanonicznej postaci uczącego się systemu klasyfikującego przyjmuje się powszechnie (patrz (Barry 2000)) model opisany przez J. Hollanda (1986).

Ternarny język zapisu klasyfikatorów ma jednak swoje ograniczenia. Niewielka liczba stosowanych liter w zapisie reguł powoduje wzrost skomplikowania reprezentacji przy próbach wyrażenia bardziej złożonych pojęć. Ternarny język nie jest też w stanie opisać dowolnych podprzestrzeni z przestrzeni tworzonej przez iloczyn kartezjański stan × akcja. Przykładowo, jedyną możliwością zapisu warunku obejmującego stany 01 i 10 jest warunek ##. Ale jednocześnie tak zapisany warunek pasuje do stanów 00 i 11, które być może nie są pożądane przez system. Warto również zauważyć, że standardowy język zapisu klasyfikatorów nie ma możliwości generalizacji akcji, gdyż w zbiorze liter alfabetu akcji nie ma znaku #. Nie można zatem w pojedynczej regule wyrazić na przykład żądania uruchomienia jednocześnie akcji 1 oraz akcji 0 w wyniku pojawienia się na wejściu systemu określonego stanu otoczenia.

Powyższe ograniczenia zainicjowały poszukiwania rozszerzeń standardowego ternarnego języka opisu klasyfikatorów. Pojawiły się wieloczłonowe warunki łączone logicznym operatorem AND (Goldberg 1983, Booker i in. 1989), negacje warunków (Smith 1980), (Booker i in. 1989) czy też litery o własnościach przenoszenia (*pass-through characters*) (Forrest 1985, Riolo 1988, Stolzmann 2000). W systemie LS-1 Smith (1980), bazując na ciągach ternarnych, zastosował złożony język zapisu, w którym m.in. wprowadził operator negacji, operator pominięcia, a akcje mogły być zależne bądź nie od wykonywanego zadania.

Poszukiwania poszły również w kierunku alternatywnych w stosunku do ternarnego metod reprezentacji klasyfikatorów. Wilson zaproponował wersję rzeczywistoliczbową systemu XCS, w której w warunku reguły występują atrybuty przyjmujące wartości z określonego przedziału (Wilson 2000) oraz wersję całkowitoliczbową (Wilson 2001). Lanzi (1999b) oraz Ahluwalia i Bull (1999) badali klasyfikatory o zmiennej długości, używając w ich przetwarzaniu programowania genetycznego. W 1991 r. Valenzuela-Rendon (1991) opisał rozmyty uczący się system klasyfikujący, w którym zastosowano logikę rozmytą w opisie warunku klasyfikatora. W (Bull i O'Hara 2002) warunki reguł oraz akcje są wielowarstwowymi perceptronami. W systemie GCS (Unold 2005a) klasyfikatory reprezentowane są bezpośrednio przez produkcje gramatyki bezkontekstowej.

3.2.4. Kryterium mechanizmu stosowanego w module odkrywczym

Zgodnie z intencją Hollanda oraz wielu innych badaczy, uczące się systemy klasyfikujące traktuje się jako przykład maszynowego uczenia bazującego na algorytmach genetycznych – *genetics based machine learning*. Zasadniczym elementem modułu odkrywczego jest algorytm genetyczny, algorytm przyznawania nagrody może być traktowany jako szczególnego rodzaju funkcja przystosowania, a stosowanie populacji klasyfikatorów jest jedynie sposobem aplikacji algorytmu genetycznego dla pewnych typów zadań.

Jednak kategoryzacja systemów klasyfikujących jako jeszcze jednego typu przetwarzania ewolucyjnego nie jest wcale taka oczywista. Świadczy o tym chociażby pytanie, które zadali sobie naukowcy na co dzień zajmujący się tą tematyką w zbiorze jedenastu krótkich esejów zatytułowanych *Czym są uczące się systemy klasyfikujące* (Holland i in. 2000).

Do stawiania takiego pytania uprawnia m.in. fakt, że istnieją uczące się systemy klasyfikujące, w których algorytm genetyczny nie jest najważniejszym i jedynym komponentem modułu odkrywczego. Powszechnie stosuje się operację pokrycia (*covering*), znane są też takie metody, jak: korporacyjne powiązania (*corporate linkage*) (Tomlinson i Bull 1998), mostkowanie (*bridging*) (Riolo 1987) czy też wyzwalane łańcuchy (*triggered chaining*) (Riolo 1989). W niektórych systemach przypisuje się algorytmowi genetycznemu rolę wręcz drugoplanową (Barry 2000).

Ostatnio coraz częściej szuka się też zastosowania innych metod ewolucyjnych w module odkrywczym, jak na przykład programowania genetycznego (Lanzi 1999b, Ahluwalia i Bull 1999).

Jedna z najnowszych – i dodajmy coraz bardziej popularnych – mutacji systemów klasyfikujących, tzw. ACS Wolfganga Stolzmanna, nie używa w ogóle metod ewolucyjnych podczas przetwarzania populacji klasyfikatorów (Stolzmann 2000).

3.2.5. Kryterium pamięci systemu

Możliwości uczenia uczącego się systemu klasyfikującego zależą od jego umiejętności percepcji środowiska, w którym działa. W wielu przypadkach bieżąca informacja zwrotna od otoczenia jest wystarczająca do podjęcia prawidłowej akcji przez system. W takim przypadku mówimy o środowisku w pełni obserwowalnym (*completely observable*) lub też środowisku spełniającym warunek Markowa[69]. Jeżeli jednak detektory uczącego się systemu klasyfikującego dostarczają jedynie częściowej informacji o środowisku i system powinien podejmować różne akcje w odpowiedzi na tę samą (pozornie) informację środowiskową, to mówimy wtedy o środowisku jedynie częściowo obserwowalnym (*partially observable*) lub środowisku niespełniającym warunku Markowa. W tym drugim przypadku uczący się system klasyfikujący spotyka się z problemem percepcyjnego utożsamiania (*perceptual aliasing problem*). Prostą ilustracją problemu utożsamiania jest zadanie wykonywane przez robota i polegające na pakowaniu prezentu do pudełka, które następnie należy zawinąć w ozdobny papier (przykład został zaczerpnięty z (Lin 1993)). Zakładamy, że pudełko jest na początku zamknięte. Zapakowanie prezentu do pudełka wiąże się zatem z otwarciem pudełka,

[69] W środowisku spełniającym warunek Markowa wszelkie procesy można ująć w formie cyklicznych zmian stanów, z tym że prawdopodobieństwo zmiany stanu w danym cyklu jest niezależne od tego, co się działo w procesie wcześniej. Ujmując tę definicję w kategoriach procesów stochastycznych, mówimy, że rozkłady warunkowe procesu Markowa zależą tylko od ostatnio zaobserwowanego warunku.

włożeniem prezentu do środka, zamknięciem pudełka i wreszcie jego zawinięciem w papier. Zauważmy, że po zamknięciu pudełka detektory systemu sterującego robotem wykonującym zadanie pakowania obserwują identyczną sytuację środowiskową jak przed rozpoczęciem pakowania, czyli – „zamknięte pudełko". Podejmowane później akcje muszą być jednak zupełnie inne.

Warto zauważyć, że rozróżnienie środowiska spełniającego warunek Markowa od niespełniającego zależy w znacznej mierze od… sensorów robota. Jeżeli bowiem wyposażylibyśmy robota w możliwość chociażby ważenia paczki, to obecność prezentu w zamkniętym pudełku mogłaby być prosto zweryfikowana.

Są jednak takie sytuacje, w których zwiększanie możliwości sensorycznych sterowanej maszyny nie rozwiązuje problemu percepcyjnego utożsamiania. Przykładem jest zadanie nawigacji robota w labiryncie lub też w lesie, które w różnych odmianach jest bardzo często stosowane jako test sprawności nowych architektur systemów klasyfikujących. Na rysunku 8 przedstawiono klasyczny przykład takiego zadania, znany jako Woods101 (Cliff i Ross 1994, McCallum 1996). Zadaniem robota jest znalezienie w lesie (*T – tree*) jedzenia (*F – food*).

T	*T*	*T*	*T*	*T*	*T*	*T*
T						*T*
T		*T*		*T*		*T*
T		*T*	*F*	*T*		*T*
T	*T*	*T*	*T*	*T*	*T*	*T*

Rys. 8. Środowisko Woods101
Fig. 8. The Woods101 environment

W środowisku Woods101 są dwie pozycje, zaznaczone na rysunku cieniowaniem, które robot rozpoznaje jako identyczne, a które w rzeczywistości wymagają różnych akcji. W pozycji po prawej stronie optymalną akcją robota będzie wybranie ruchu „zachód-południe", w pozycji po lewej stronie robot powinien wykonać ruch „wschód-południe". Wybranie prawidłowego ruchu w obydwu pozycjach na podstawie jedynie bieżącej informacji środowiskowej jest niemożliwe. Zauważmy, że problem utożsamiania mógłby być rozwiązany, gdyby robot (system) pamiętał, z jakiej części siatki osiągnął jedną z dwu utożsamianych pozycji. Dodanie pamięci do systemu jest kolejną, obok zwiększania liczby sensorów robota, propozycją uczenia w środowisku niespełniającym warunku Markowa.

Kanoniczny system klasyfikujący Hollanda posiada wewnętrzną listę komunikatów, która może być postrzegana jako dodatkowa pamięć systemu. Jednak tego typu LCS nie radzi sobie w sposób zadawalający z problemem utożsamiania. W (Robertson i Riolo 1988) zastosowano system Hollanda do predykcji litery w sekwencji składającej się z nieskończonej iteracji ciągu „*yhwh*", osiągając 70% skuteczność. Smith (1994) natomiast wykazał, że kanoniczny system klasyfikujący

nie jest zdolny do nauczenia optymalnej strategii w środowisku niespełniającym warunku Markowa. W 1995 roku Wilson zaproponował nowy typ systemu klasyfikującego XCS (Wilson 1995), który zachowywał się optymalnie w otoczeniu spełniającym warunek Markowa, ale ze względu na brak jakiejkolwiek pamięci wewnętrznej nie uczył się optymalnych zachowań w otoczeniu niespełniającym warunku. Jednak już w 1994 r. Wilson sugerował, że dodanie niewielkiego rejestru bitowego do systemu powinno umożliwić rozwiązywanie problemu utożsamiania (Wilson 1994). Warto w tym miejscu zauważyć, że dodanie dosłownie jednego bitu, który wskazywałby, z jakiej części siatki robot osiągnął jedną z utożsamianych pozycji w środowisku Woods101 (rys. 8), pozwoliłoby na wyuczenie optymalnej strategii.

Inną propozycją rozwiązania problemu percepcyjnego utożsamiania są tzw. korporacje klasyfikatorów (*classifier corporations*), w których obiektem działania algorytmu genetycznego są całe zespoły klasyfikatorów (Wilson i Goldberg 1989), tworzące pośrednią architekturę pomiędzy podejściem Michigan a Pitt. W takiej architekturze tworzy się łańcuch wykonań, w którym aktywacja klasyfikatora na jego początku tworzy kontekst – swoistą pamięć, dla klasyfikatorów aktywowanych później. Idea korporacji klasyfikatorów została zastosowana w systemie ZCS (Tomlinson i Bull 1998) oraz w XCS (Tomlinson i Bull 1999). W najlepszym przypadku osiągnięto dostosowanie do środowiska rzędu 85%.

Stolzmann zastosował również swój system ACS do uczenia w środowisku niespełniającym warunku Markowa, używając zapamiętanych wcześniej stanów do rozróżniania pojawiających się później stanów utożsamianych (Stolzmann 1999).

Optymalna strategia uczenia została wyuczona przez system XCSM, w którym dodano rejestr pamięci do architektury XCS (Lanzi 1998, Lanzi i Wilson 2000).

W 1994 roku Littman pokazał, że znalezienie optymalnej strategii, która nie używa pamięci, w środowisku niespełniającym warunku Markowa jest zadaniem *NP*-trudnym (Littman 1994). W związku z tym kryterium pamięci jest w systemach klasyfikujących jednocześnie kryterium typu otoczenia, w jakim system jest zdolny optymalnie się zachowywać.

3.3. Wybrane modele

Jak już wcześniej wspomniano, definicja systemu klasyfikującego, jako regułowego systemu adaptującego się do otoczenia, w którym działa, jest bardzo pojemna. Dlatego też funkcjonujących dzisiaj modeli, różniących się czasami drobnymi (choć należy dodać najczęściej istotnymi) szczegółami[70], jest już kilkadziesiąt. Poniżej zestawiono listę modeli najbardziej znanych, uporządkowanych według daty powstania:

[70] Na przykład XCSM różni się od XCS rejestrem pamięci, dzięki czemu XCSM działa już poprawnie w środowisku niespełniającym warunku Markowa (Lanzi i Wilson 2000).

CS-1	(Holland i Reitman 1978),
LS-1	(Smith 1980),
Animat	(Wilson 1985),
BOOLE	(Wilson 1987),
CFS-C	(Riolo 1988),
SCS (*Simple* CS)	(Goldberg 1989),
GOFER-1	(Booker 1989),
VCS (*Variable* CS)	(Shu i Schaeffer 1989),
CSM (CS *with Memory*)	(Zhou 1990),
NEWBOOLE	(Bonelli i in. 1990),
SAMUEL	(Grefenstette i in. 1990),
FCS (*Fuzzy* CS)	(Valenzuela-Rendon 1991),
HCS (*Hierarchical* CS)	(Shu i Schaeffer 1991),
ZCS (*Zeroth-level* CS)	(Wilson 1994),
DACS (*Delayed Action* CS)	(Carse 1994),
OCS (*Organizational* CS)	(Wilcox 1995),
XCS (*eXtended* CS)	(Wilson 1995),
EpiCS (CS *for Epidemiologic data*)	(Holmes 1996),
ACS (*Anticipatory* CS)	(Stolzmann 1997),
XCSM (XCS *with Memory*)	(Lanzi 1998),
CCS (*Corporate* CS)	(Tomlinson i Bull 1998),
XCSL (XCS *with Lisp s-expression*)	(Lanzi 1999a),
mXCS (*messy* XCS)	(Lanzi 1999b),
XCSR (XCS *with Real-coded variables*)	(Wilson 2000),
XCSI (XCS *with Integer-coded variables*)	(Wilson 2001),
SB-XCS (*Strength-based* XCS)	(Kovacs 2002a),
XNCS (*Neuro-Fuzzy* XCS)	(Bull i O'Hara 2002),
YACS (*Yet Another* CS)	(Gérard i in. 2002),
ICU (*I See You*)	(Quirin 2002),
GALE (*Genetic and Artif. Life Environment*)	(Llora 2002).

Praktycznie wszystkie wyżej wymienione modele[71] można pogrupować – według zasady działania, a co najmniej inspiracji ich twórców – wokół jednego z następujących systemów klasyfikujących: LCS, ZCS, XCS oraz ACS. W następnych rozdziałach zostaną omówione charakterystyczne własności właśnie tych czterech modeli[72].

[71] Dosyć charakterystyczną nazwę nosi jeden z ostatnio zaproponowanych modeli, a mianowicie YACS (*yet another classifier system*) – jeszcze jeden system klasyfikujący (Gérard i in. 2002), będący zresztą mutacją systemu ACS.

[72] Obszerniejsze omówienie systemów klasyfikujących zawarte jest w przygotowywanej do druku monografii *Uczące się systemy klasyfikujące*.

3.3.1. LCS

Kanoniczny, uczący się system klasyfikujący (*Learning Classifier System*, LCS) zaproponowany przez Hollanda (1980, 1986), rys. 9, jest w istocie wersją systemu produkcyjnego, tj. systemu obliczeniowego, używającego reguł w postaci *jeżeli...*, *to ...*, umożliwiających reagowanie na zmiany środowiska. Na system klasyfikujący można patrzeć jak na model uczenia się ze wzmocnieniem (patrz rys. 6). LCS otrzymuje od otoczenia informację o jego stanie $s \in \{0, 1\}^l$, gdzie l jest liczbą bitów reprezentujących każdy stan. W odpowiedzi system wykonuje akcję $\alpha \in \{a_1, \ldots, a_n\}$, za którą może otrzymać skalarną nagrodę r. Na LCS składa się Populacja klasyfikatorów, Lista komunikatów (*message list*), Detektory (*detectors*) i Efektory (*efectors*), Zbiór klasyfikatorów pasujących (*match set*) oraz aktywnych (*active set*), rys. 9. Klasyfikator jest trójką $\{C, A, S\}$, gdzie:

$C \in \{0, 1, \#\}^l$ – warunek klasyfikatora (*condition*),
$A \in \{0, 1, \#\}^l$ – komunikat (akcja) klasyfikatora (*action*),
S – siła klasyfikatora.

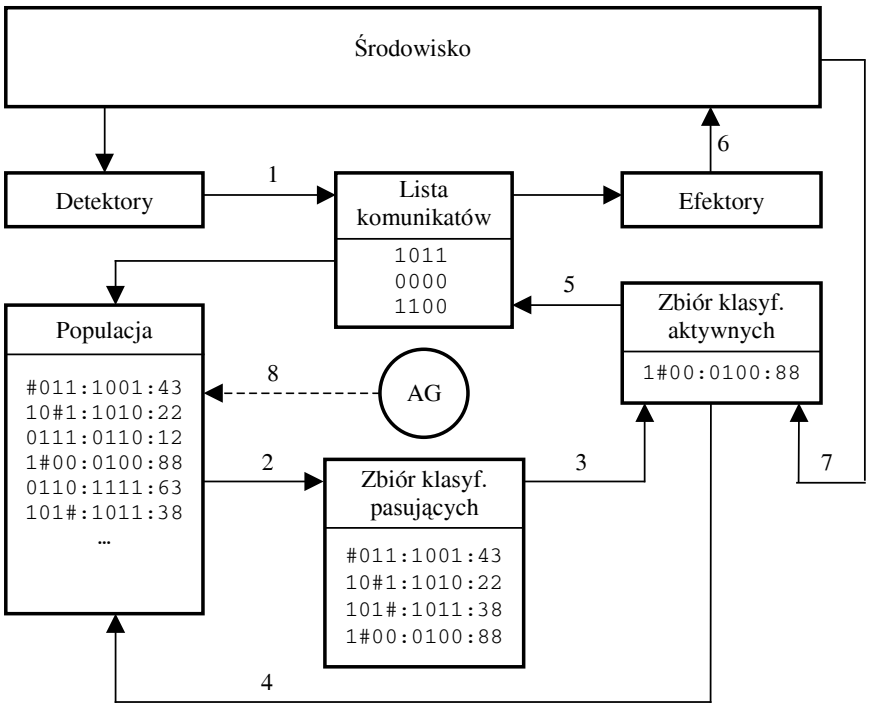
W kanonicznym systemie klasyfikującym warunek klasyfikatora oraz komunikat mają tę samą długość l, dzięki czemu komunikat wysłany do Listy komunikatów w bieżącym cyklu uczenia może być porównywany z częścią warunkową klasyfikatora w następnym cyklu uczącym. Symbol # w warunku klasyfikatora oznacza symbol uniwersalny (*don't care*), który zastępuje dowolny symbol z alfabetu komunikatów[73]. Ten sam symbol po stronie akcji klasyfikatora reprezentuje tzw. własność przenoszenia (*pass through*), polegającą na przeniesieniu wartości oznaczonego tak właśnie bitu z dopasowanego komunikatu do komunikatu wysyłanego do otoczenia[74]. Siła klasyfikatora (*strength*) jest miarą jego przystosowania do środowiska[75]. Pojedynczy cykl uczenia kanonicznego systemu klasyfikującego przedstawiono na rys. 10.

System obserwuje środowisko poprzez Detektory, zadaniem których jest przetwarzanie sygnałów zewnętrznych na komunikaty. Komunikaty te są dodawane do Listy komunikatów, która zawiera wszystkie komunikaty $M \in \{0, 1\}^l$ wysłane przez

[73] Stosowanie symbolu uniwersalnego na i-tej pozycji warunku oznacza ignorowanie wartości i-tej pozycji detektora.

[74] Stosowanie symbolu *pass through* na i-tej pozycji akcji jest ukrytym założeniem, że wartość i-tej pozycji detektora nie zmieni się po odpaleniu akcji wybranego klasyfikatora.

[75] W pierwszej implementacji LCS (Holland i Reitman 1978) siła klasyfikatora była przewidywaną miarą oczekiwanej wypłaty za zastosowanie klasyfikatora, a wypłata była tylko realizowana dla tych klasyfikatorów, dla których estymacje wypłaty nie były większe od rzeczywiście otrzymanej. W późniejszej realizacji LCS (Holland 1986) siła klasyfikatora była już tylko miarą jego użyteczności w otrzymywaniu wypłaty.



Rys. 9. Kanoniczny system klasyfikujący LCS, na podstawie (Wyatt 2004)
Fig. 9. Canonical learning classifier system LCS, adopted from (Wyatt 2004)

zwycięskie klasyfikatory w poprzednim cyklu uczenia (patrz rys. 10 – krok 1). Zbiór klasyfikatorów pasujących tworzony jest przez klasyfikatory, dla których część warunkowa pasuje do komunikatów z Listy komunikatów (krok 2). Warunek klasyfikatora pasuje do komunikatu (jest z nim zgodny), jeżeli na każdej pozycji za wyjątkiem symboli uniwersalnych jest z nim identyczny. Wszystkie klasyfikatory ze Zbioru klasyfikatorów pasujących nabywają prawo do uczestnictwa w przetargu (*auction*) na uaktywnienie (krok 3a). Podczas przetargu każdy uprawniony klasyfikator składa ofertę (*bid*) proporcjonalną do swojej siły[76]. Tylko klasyfikatory z najwyższą ofertą mają prawo przejścia do Zbioru klasyfikatorów aktywnych (krok 3b) i umieszczenia swoich komunikatów na Liście komunikatów. Siła wszystkich aktywnych klasyfikatorów zostaje pomniejszona o złożoną przez nich ofertę (krok 4a), natomiast siła klasyfikatorów odpowiedzialnych za wysłanie komunikatu aktywującego klasyfikatory zostaje powiększona o równo podzieloną wartość oferty klasyfikatorów aktywnych (krok 4b)[77].

[76] Siła klasyfikatora reprezentuje jego użyteczność w poprzednich krokach uczenia (*past usefulness*).

[77] W ogólnym przypadku poza tzw. podatkiem obrotowym, który płacą tylko klasyfikatory składające ofertę, można również pobierać podatek pogłówny, który płacą wszystkie klasyfikatory.

LCS:

1. Wstaw zdekodowane przez Detektory komunikaty na Listę komunikatów
2. Utwórz Zbiór klasyfikatorów pasujących do komunikatów z Listy komunikatów
3.
   a. Oblicz ofertę składaną przez klasyfikatory ze Zbioru klasyfikatorów pasujących
   b. Utwórz Zbiór klasyfikatorów aktywnych
4.
   a. Zmniejsz siłę klasyfikatorów ze Zbioru klasyfikatorów aktywnych o złożoną ofertę
   b. Zwiększ siłę klasyfikatorów dostarczających komunikaty dla klasyfikatorów aktywnych
5.
   a. Wyczyść Listę komunikatów
   b. Wyślij komunikaty klasyfikatorów ze Zbioru klasyfikatorów aktywnych na Listę komunikatów
6. Uaktywnij komunikaty z Listy komunikatów za pośrednictwem Efektorów
7. Podziel nagrodę pomiędzy klasyfikatory ze Zbioru klasyfikatorów aktywnych
8. Uruchom Algorytm genetyczny

Rys. 10. Pojedynczy cykl uczenia kanonicznego systemu klasyfikującego LCS
Fig. 10. Single learning cycle of LCS

Warto przy okazji zauważyć za Bookerem (1982), że takie wzmacnianie klasyfikatorów dostarczających komunikaty sprzyja wytwarzaniu się podpopulacji klasyfikatorów o odpowiednich rozmiarach. Nie jest bowiem uzasadnione poszukiwanie jednego, uniwersalnego klasyfikatora reagującego prawidłowo na zróżnicowane bodźce otrzymywane ze środowiska. Należy natomiast szukać koegzystujących ze sobą podzbiorów klasyfikatorów umożliwiających różnorodne zachowanie, jakie wymaga otoczenie systemu[78]. Algorytmiczny zapis układu oceniającego LCS, zwany algorytmem „kubełkowym" przyznawania nagród (*bucket brigade algorithm*) (Holland 1986), można znaleźć w (Piech 2003), natomiast jego równania w (Goldberg 1989) i (Wierzchoń 2001). Kolejnym krokiem wykonywanym przez uczący się system klasyfikujący jest uaktywnienie Efektorów, zdolnych do oddziaływania na środowisko, przez komunikaty znajdujące się na Liście komunikatów (krok 6). Jeżeli wynikiem oddziaływania systemu z otoczeniem jest nagroda otoczenia, to jest ona dzielona równo pomiędzy wszystkie klasyfikatory ze Zbioru klasyfikatorów aktywnych (krok 7). Zadaniem algorytmu kubełkowego jest przyznawanie ocen i wybór klasyfikatorów, natomiast za tworzenie nowych i modyfikacje istniejących klasyfikatorów odpowiedzialny jest algorytm genetyczny (AG). Algorytm genetyczny jest uruchamiany w każdym cyklu uczenia z zadanym prawdopodobieństwem

[78] W (Horn i in. 1994) zauważono, że w systemie klasyfikującym współzawodnictwo o ograniczone zasoby (czyli nagrody) prowadzi do słabej kooperacji (*weak cooperation*) klasyfikatorów, a w konsekwencji do optymalnego wykorzystania tychże zasobów. Owa współpraca polega właśnie na dzieleniu wspólnych zasobów poprzez różne „gatunki" klasyfikatorów, zajmujących odmienne „nisze ekologiczne".

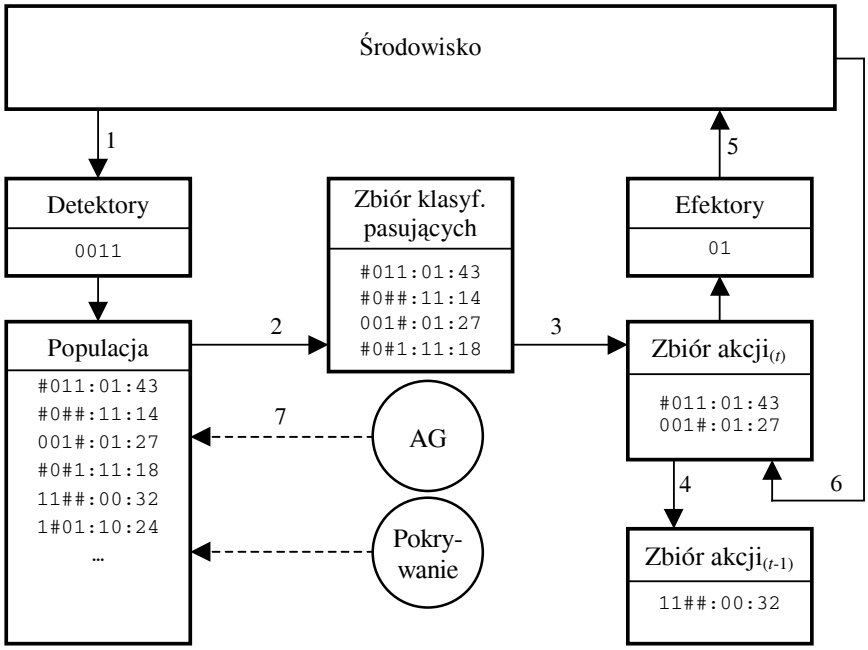
mniejszym od jeden[79]. Nowe klasyfikatory tworzone są w oparciu o reprodukcję, krzyżowanie i mutacje (krok 8). W procesie selekcji używa się najczęściej reguły ruletki, a siła klasyfikatora określa jego przystosowanie. Ze względu na oczekiwaną wysoką efektywność bieżącą uczenia w czasie, zasięg selekcji jest ograniczany do części populacji (*steady-state selection*).

W 1975 roku Holland zaproponował, by klasyfikatory organizowały się w pewną strukturę, zwaną hierarchią domniemań (*default hierarchy*), w której klasyfikatory ogólne (zawierające po stronie warunku wiele symboli uniwersalnych) reprezentowałyby przypadki ogólne, a klasyfikatory szczegółowe – przypadki wyjątkowe (Holland 1975). Goldberg w swej pracy doktorskiej wskazał, jak można organizować taką strukturę klasyfikatorów, w której dla przypadku, gdy dwa klasyfikatory składają jednocześnie ofertę jako odpowiedź na ten sam komunikat, wygrywa klasyfikator szczegółowy (Goldberg 1983). W hierarchii domniemań populacja klasyfikatorów jest mniej liczna od populacji klasyfikatorów rozłącznych oraz powiększa zbiór rozwiązań, gdyż klasyfikatory szczegółowe mogą skutecznie uzupełniać już istniejące zbiory rozłącznych, ogólnych klasyfikatorów.

3.3.2. ZCS

W 1994 roku Wilson zaprezentował nowy uczący się system klasyfikujący ZCS (*Zeroth level Classifier System*) (Wilson 1994). Zasadniczym zadaniem nowej wersji systemu klasyfikującego było uproszczenie modelu kanonicznego w celu lepszego zrozumienia występujących w nim mechanizmów. ZCS podobnie jak LCS działa w pewnym środowisku, od którego otrzymuje zdekodowane za pomocą detektorów binarne sygnały. Na podstawie otrzymywanych sygnałów, system określa własną reakcję, którą za pośrednictwem komunikatów przesyłanych do efektorów przekazuje do otoczenia, zmieniając w ten sposób jego stan. Reakcja systemu jest nagradzana przez środowisko poprzez przekazanie skalarnej nagrody. Nowe reguły systemu są odkrywane przez algorytm genetyczny. ZCS jednak istotnie różni się od swego wzorca. Nie ma bowiem listy komunikatów, a klasyfikatory są bezpośrednio dopasowywane do aktualnie obserwowanego stanu otoczenia. Pojawił się dodatkowy mechanizm odkrywczy, zwany pokryciem (*covering*), odpowiedzialny za tworzenie nowych klasyfikatorów. ZCS wprowadził również nowy algorytm wzmacniający, nazwany przez Goldberga (1989) „ukrytym algorytmem kubełkowym" (*implicit bucket brigade algorithm*). Algorytm ten działa na odpowiednich zbiorach akcji.

[79] W ogólnym wypadku wywołania AG mogą być stochastyczne lub deterministyczne, czyli dokładnie co zadana parametrem liczba kroków.



Rys. 11. System klasyfikujący ZCS, na podstawie (Wyatt 2004)
Fig. 11. Learning classifier system ZCS, adopted from (Wyatt 2004)

Na architekturę ZCS (rys. 11) składa się: Populacja klasyfikatorów, Detektory, Efektory, Zbiór klasyfikatorów pasujących, Zbiór akcji[80] $[A]_t$ (*Action Set*) dla czasu t oraz Zbiór akcji $[A]_{t-1}$ dla czasu $t - 1$. Podobnie jak w systemie LCS, klasyfikator systemu ZCS jest trójką $\{C, A, S\}$, gdzie:

$C \in \{0, 1, \#\}^l$ – warunek klasyfikatora,
$A \in \{0, 1, \#\}^l$ – komunikat (akcja) klasyfikatora,
S – siła klasyfikatora.

Pojedynczy cykl uczenia ZCS przedstawiono na rys. 12. Na początku każdego cyklu uczenia Detektory generują binarny ciąg o długości l reprezentujący aktualny stan otoczenia (patrz rys. 12 – krok 1). Następnie tworzony jest Zbiór klasyfikatorów pasujących z klasyfikatorów, dla których część warunkowa zgodna jest z komunikatem zewnętrznym (krok 2a). Jeżeli Zbiór klasyfikatorów pasujących jest pusty, wywoływany jest operator pokrycia (krok 2b).

[80] W polskim przekładzie książki Goldberga (1989) pojęcie *covering* tłumaczone jest jako *operacja kreacji*, a *action set* jako *grupa wykonawcza*.

ZCS:

1. Utwórz komunikat zewnętrzny reprezentujący aktualny stan otoczenia
2.
   a. Utwórz Zbiór klasyfikatorów pasujących do komunikatu zewnętrznego
   b. Wywołaj operator pokrycia
3.
   a. Wybierz akcję w sposób deterministyczny albo stochastyczny
   b. Utwórz Zbiór akcji
4.
   a. Przekaż część siły Zbioru akcji do izby rozrachunkowej
   b. Zmniejsz wartość izby rozrachunkowej o rabat
   c. Podziel wartość izby rozrachunkowej pomiędzy klasyfikatory poprzednio aktywnego Zbioru akcji
   d. Opodatkuj klasyfikatory, które nie weszły do Zbioru akcji
5. Uaktywnij akcję w otoczeniu za pośrednictwem Efektorów
6.
   a. Zredukuj otrzymaną od otoczenia nagrodę
   b. Podziel nagrodę pomiędzy klasyfikatory ze Zbioru akcji
7. Uruchom Algorytm genetyczny

Rys. 12. Pojedynczy cykl uczenia ZCS
Fig. 12. Single learning cycle of ZCS

Przyjmuje się również często, że wywołanie operatora pokrycia może nastąpić w sytuacji, gdy sumaryczna siła klasyfikatorów pasujących jest mniejsza od określonej części (np. połowy) sumarycznej siły klasyfikatorów w populacji. Operator ten tworzy nowy klasyfikator, którego część warunkowa pasuje do komunikatu zewnętrznego lub go pokrywa. Część warunkowa nowego klasyfikatora zawiera losową liczbę symboli uniwersalnych #, akcja tworzona jest losowo, a siła jest średnią z populacji. Tak powstały nowy klasyfikator zastępuje jeden wybrany klasyfikator z populacji, najczęściej o niewielkiej sile. W następnym kroku wybierana jest akcja ze Zbioru klasyfikatorów pasujących. Wybór akcji może być probabilistyczny lub deterministyczny (krok 3a). W pierwszym podejściu stosowana jest metoda ruletki, w której wycinki koła są proporcjonalne do siły klasyfikatorów. W deterministycznej metodzie wybieramy tę akcję, dla której suma sił wywołujących ją klasyfikatorów jest największa. Po wyborze akcji formułowany jest Zbiór akcji $[A]_t$, zawierający wszystkie te klasyfikatory ze Zbioru klasyfikatorów pasujących, które wywołują wskazaną w kroku poprzednim akcję (krok 3b). Następnie akcja zostaje przesłana do Efektorów, które przekazują ją do otoczenia, zmieniając jego stan (krok 5). Mechanizmy wzmocnienia w ZCS polegają na przekazywaniu siły pomiędzy następującymi po sobie w czasie Zbiorami akcji $[A]_t$ i $[A]_{t-1}$. W pierwszej kolejności siła wszystkich klasyfikatorów z $[A]_t$ zostaje pomniejszona o tzw. współczynnik uczenia β ($0 < \beta \leq 1$) (krok 4a), a uzyskana w ten sposób siła zostaje umieszczona w swego rodzaju „izbie rozrachunkowej". Wartość przekazanej siły równa jest $\beta S_{|A|}$, gdzie $S_{|A|}$ to sumaryczna siła klasyfikatorów z $[A]_t$. Dodatkowo, zawartość izby rozrachunkowej zostaje pomniejszona

o rabat γ ($0 < \gamma \leq 1$) (krok 4b). Otrzymany w ten sposób zasób siły zostaje równo podzielony pomiędzy klasyfikatory poprzednio aktywnego Zbioru Akcji $[A]_{t-1}$, czyli siła każdego klasyfikatora tego zbioru zostaje zwiększona o wartość $\beta\gamma S_{|A|}$ (krok 4c). Goldberg (1989) zauważył, że jest to w istocie ukryty algorytm kubełkowy, rozprowadzający nagrody środowiska wzdłuż „łańcucha" aktywowanych klasyfikatorów. Siła wszystkich klasyfikatorów pasujących, które nie znalazły się w $[A]_t$ zostaje pomniejszona o podatek τ ($0 < \tau \leq 1$) (krok 4d). Zabieg ten ma wymusić eksploracje silniejszych klasyfikatorów. Jeżeli system, w wyniku wykonania akcji, otrzymał nagrodę r_{imm} od otoczenia, to jest ona w pierwszej kolejności redukowana o współczynnik β (krok 6a), a następnie każdy klasyfikator z $[A]_t$ otrzymuje dodatkową siłę o wartości $\beta r_{imm}/|A_t|$, gdzie $|A_t|$ jest mocą zbioru $[A]_t$ (krok 6b). W efekcie siła wszystkich klasyfikatorów zbioru $[A]_t$ zostaje powiększona o wartość βr_{imm}. Redukcja otrzymywanej od otoczenia nagrody ma za zadanie zapobieżenie dominacji w populacji klasyfikatorów, które tworzą wysoko nagradzane podpopulacje (tzw. nisze). ZCS stosuje dwa mechanizmy odkrywcze – omówiony już wcześniej operator pokrycia oraz algorytm genetyczny, którego zasięg działania, podobnie jak to ma miejsce w przypadku LCS, jest ograniczony do wybranej części populacji i jest uruchamiany w każdym cyklu uczenia z zadanym prawdopodobieństwem mniejszym od jeden (krok 7). Aby zachować sumaryczną siłę oraz liczebność genetycznie modyfikowanej populacji, potomkowie powstali w wyniku operacji krzyżowania i mutacji otrzymują połowę siły rodziców, a następnie zastępują dwa klasyfikatory wylosowane metodą ruletki.

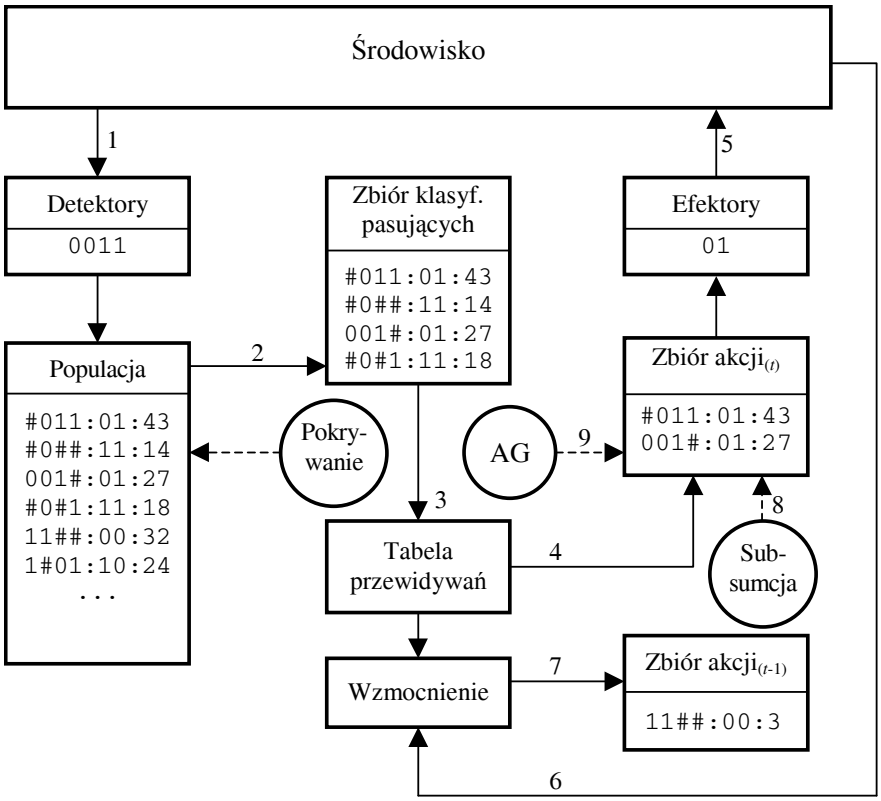
3.3.3. XCS

Zarówno LCS, jak i ZCS są przykładami systemów klasyfikujących, w których mechanizm odkrywczy bazuje na opłacalności zastosowania klasyfikatorów (*payoff--based classifier system*). Prowadzi to generalnie do dwóch problemów, które tego typu systemy napotykają.

Pierwszym z nich jest powstawanie zbyt ogólnych klasyfikatorów (*over-general rules*), drugim natomiast mechanizm zachłannego tworzenia nowych klasyfikatorów (*greedy classifier creation*). Według Kovacsa (2000) „zbyt ogólny klasyfikator" pasuje do wielu stanów, z tym że równocześnie jest nieprawidłowy dla kilku. Mechanizm zachłannego tworzenia nowych klasyfikatorów objawia się tym, że algorytm genetyczny preferuje klasyfikatory o wysokich wypłatach, co może w konsekwencji prowadzić system do posiadania zbyt małej ich liczby, a w skrajnych przypadkach nawet żadnego, dla stanów, które nie zwracają wysokich nagród (Cliff i Ross 1994). Oba przypadki prowadzą do nieoptymalnego zachowania uczącego się systemu klasyfikującego.

Zaproponowany przez Wilsona (1995) system XCS (*eXtended Classifier System*) jest systemem klasyfikującym, w którym moduł odkrywczy jako miarę przydatności klasyfikatora stosuje jego przewidywaną dokładność otrzymywanej wypłaty (*accuracy-based*

classifier system), w miejsce wartości samej wypłaty. XCS, w przeciwieństwie do swoich poprzedników, tworzy kompletną i dokładną mapę możliwych wejść, akcji i wypłat (*payoff landscape*), dzięki uwzględnianiu również klasyfikatorów przynoszących niewielką wypłatę do systemu, ale niezbędnych w swoich niszach[81] środowiskowych. Klasyfikatory zbyt ogólne nie są dokładne i zastępowane są przez mniej ogólne, ale dokładniejsze reguły, a algorytm genetyczny operuje w niszach, w których wartość dopasowania klasyfikatora jest związana z jego dokładnością, a nie spodziewaną wypłatą.



Rys. 13. System klasyfikujący XCS, na podstawie (Wyatt 2004)
Fig. 13. Learning classifier system XCS, adopted from (Wyatt 2004)

Na rysunku 13 zilustrowano architekturę systemu XCS. System XCS, podobnie jak LCS i ZCS, kontaktuje się ze światem zewnętrznym poprzez detektory, za pomocą których otrzymuje aktualny stan otoczenia oraz efektory, poprzez które wykonuje w środowisku akcje. Odpowiedzią otoczenia na akcję jest skalarna nagroda. XCS

[81] Niszę możemy zdefiniować jako podprzestrzeń par (wejście, akcja), które przynoszą taką samą wypłatę.

składa się z Populacji klasyfikatorów, Detektorów i Efektorów, Zbioru akcji $[A]_t$ dla czasu t oraz Zbioru akcji $[A]_{t-1}$ dla czasu $t-1$, Tabeli przewidywań (*prediction array*). Klasyfikator systemu XCS jest zbiorem $\{C, A, p, \varepsilon, f, exp, ts, as, num\}$, gdzie:

$C \in \{0, 1, \#\}^l$ – warunek klasyfikatora,
$A \in \{0, 1, \#\}^l$ – komunikat (akcja) klasyfikatora,
p – przewidywana wypłata (*payoff prediction*),
ε – błąd przewidywania (*prediction error*),
f – dopasowanie (*fitness*),
exp – doświadczenie (*experience*),
ts – znacznik czasu (*time-stamp*),
as – wielkość Zbioru akcji (*Action Set size estimate*),
num – liczność (*numerosity*).

Warunek C klasyfikatora definiuje możliwe stany (komunikaty) otoczenia, do których klasyfikator może się dopasować. Parametr A określa akcję proponowaną przez klasyfikator, natomiast p szacuje przewidywaną wypłatę za wykonanie akcji. Dopasowanie klasyfikatora f jest funkcją odwrotną błędu przewidywania ε, który znowu jest oszacowaniem błędu popełnionego podczas przewidywania wypłaty. Algorytm genetyczny używa dopasowania klasyfikatora podczas selekcji tych klasyfikatorów, które już udowodniły swoją dokładność w przewidywaniu wypłaty. Doświadczenie exp zlicza, ile razy klasyfikator znalazł się w Zbiorze akcji. Miara ta wykorzystywana jest przez operator subsumcji (*subsumption*) oraz technikę aktualizacji MAM (*Moyenne Adaptive Modifée*). Znacznik czasu ts jest zwiększany po zakończeniu każdego cyklu uczenia i wykorzystywany do zapamiętywania, kiedy Zbiór akcji był ostatnio ewoluowany przez algorytm genetyczny. Wielkość Zbioru akcji as szacuje średni rozmiar Zbioru akcji, do którego należał klasyfikator. Obliczanie as umożliwia alokację równych zasobów do środowiskowych nisz, reprezentowanych w XCS przez Zbiór akcji. Liczność num jest liczbą wystąpień mikroklasyfikatorów (*micro-classifier*) w makroklasyfikatorze (*macro-classifier*).

Precyzyjny opis działania XCS nie jest możliwy bez podania zestawu parametrów uczenia, które wpływają na działanie algorytmu genetycznego, obliczanie funkcji dopasowania, wybór akcji i operator pokrycia. Parametry uczenia XCS są następujące:

N – maksymalny rozmiar populacji klasyfikatorów, licząc wystąpienia wszystkich mikroklasyfikatorów (suma parametrów num poszczególnych klasyfikatorów),
β – współczynnik uczenia dla p, ε, f, as,
α, ε_0, ν – parametry używane przez funkcję dopasowania,
γ – rabat aktualizacji przewidywania wypłaty,
χ – prawdopodobieństwo krzyżowania dla algorytmu genetycznego,
μ – prawdopodobieństwo mutacji dla algorytmu genetycznego,
δ – ułamek ze średniej dopasowania w populacji klasyfikatorów, poniżej której dopasowanie klasyfikatora może być rozpatrywane w procedurze usuwania klasyfikatora,

$P_{\#}$ – prawdopodobieństwo zastosowania symbolu uniwersalnego # przez operator pokrycia,

p_I, ε_I, f_I – wartości początkowe nowych klasyfikatorów,

p_{exp} – prawdopodobieństwo losowego wyboru akcji z Tabeli przewidywań,

θ_{sub} – wartość progowa subsumcji (*subsumption threshold*), która określa minimalne doświadczenie, jakie musi posiadać klasyfikator, aby mógł brać udział w subsumcji,

θ_{del} – próg kasowania (*deletion threshold*), który określa minimalne doświadczenie, jakie musi posiadać klasyfikator, aby mógł brać udział w procedurze usuwania klasyfikatora,

θ_{GA} – prawdopodobieństwo uruchomienia algorytmu genetycznego,

θ_{mna} – minimalna liczba akcji, jaka musi być w Zbiorze akcji, aby nie uruchamiać operatora pokrycia.

Na rysunku 14 przedstawiono schematycznie pojedynczy cykl uczenia systemu XCS. Na początku cyklu uczenia system XCS otrzymuje poprzez Detektory komunikat zewnętrzny, który opisuje aktualny stan otoczenia (rys. 14 – krok 1). Komunikat zewnętrzny jest uzgadniany z warunkami klasyfikatorów znajdujących się w populacji, w wyniku czego tworzony jest Zbiór klasyfikatorów pasujących, zawierający klasyfikatory, które opisują aktualny stan otoczenia (krok 2a).

XCS:

1. Utwórz komunikat zewnętrzny reprezentujący aktualny stan otoczenia
2.
   a. Utwórz Zbiór klasyfikatorów pasujących do komunikatu zewnętrznego
   b. Wywołaj operator pokrycia
3. Skonstruuj Tabelę przewidywań
4.
   a. Wybierz akcję w sposób deterministyczny albo stochastyczny
   b. Utwórz Zbiór akcji
5.
   a. Uaktywnij akcję w otoczeniu za pośrednictwem Efektorów
   b. Przekaż otrzymaną od otoczenia nagrodę do następnego cyklu uczenia
6. Oblicz wskaźnik aktualizacji
7. Dla wszystkich klasyfikatorów w Zbiorze akcji (dla czasu t lub $t-1$)
   a. Oblicz przewidywaną wypłatę
   b. Oblicz błąd przewidywań
   c. Oblicz rozmiar Zbioru akcji
   d. Zaktualizuj dopasowanie
8. Uruchom operator subsumcji w Zbiorze akcji
9.
   a. Uruchom algorytm genetyczny w Zbiorze akcji
   b. Uruchom operator subsumcji w Zbiorze akcji
   c. Wstaw nowe klasyfikatory do Zbioru akcji

Rys. 14. Pojedynczy cykl uczenia XCS
Fig. 14. Single learning cycle of XCS

Następnie system sprawdza, czy liczba akcji w Zbiorze klasyfikatorów pasujących jest większa od parametru θ_{mna}. Jeżeli nie jest większa, to operator pokrycia działa tak długo, aż powyższy warunek stanie się prawdziwy (krok 2b). Operator pokrycia tworzy nowy klasyfikator, którego część warunkowa pasuje do komunikatu zewnętrznego bądź go pokrywa i który zawiera losową liczbę symboli uniwersalnych #, zgodnie z wartością $P_{\#}$. Akcja nowego klasyfikatora generowana jest losowo, ale w ten sposób, by nie pokrywała się z akcjami już obecnymi w Zbiorze klasyfikatorów pasujących. Parametry nowego klasyfikatora są ustawiane następująco: $p = p_I$, $\varepsilon = \varepsilon_I$, $f = f_I$, $exp = 0$, $as = 1$, $num = 1$. W kolejnym kroku dla każdej akcji ze Zbioru klasyfikatorów pasujących tworzony jest wpis w Tabeli przewidywań, w której zapisywana jest średnia z parametru przewidywania, ważona przez dopasowanie klasyfikatorów generujących akcję (krok 3). Wartości Tabeli przewidywań można traktować jako estymacje systemu możliwych zysków z zastosowania danej akcji. Następnie wybierana jest losowo bądź deterministycznie akcja ze Zbioru klasyfikatorów pasujących. Wiele implementacji XCS stosuje jednocześnie zarówno wybór probabilistyczny, oparty o prawdopodobieństwo p_{exp}, jak i deterministyczny, w którym wybrana akcja ma przypisaną najwyższą wartość w Tabeli przewidywań[82] (krok 4a).

Na podstawie wybranej akcji formułowany jest Zbiór akcji, zawierający wszystkie klasyfikatory ze Zbioru klasyfikatorów pasujących, które odpalają wskazaną w kroku poprzednim akcję (krok 4b). Wybrana akcja jest następnie wysyłana poprzez Efektory do otoczenia (krok 5a). System XCS może być stosowany dla zadań jednoetapowych (*single-step*) i wieloetapowych (*multi-step*). W zadaniach jednoetapowych otrzymana nagroda, będąca reakcją otoczenia na wywołaną akcję, kończy proces uczenia, w zadaniach wieloetapowych proces uczenia rozłożony jest na wiele pojedynczych cykli uczenia, a otrzymana nagroda w jednym cyklu przekazywana jest do kolejnego (krok 5b). W oczywisty sposób różnica pomiędzy obydwoma typami rozwiązywanych przez XCS zadań widoczna jest również w mechanizmach aktualizacji podstawowych parametrów Zbioru akcji, czyli p, ε, f. W zadaniach jednoetapowych, w których z przyczyn oczywistych nie funkcjonuje Zbiór akcji $[A]_{t-1}$ z poprzedniego cyklu uczenia, tzw. współczynnik aktualizacji P równy jest otrzymanej przez system wypłacie, a aktualizacji podlegają klasyfikatory ze zbioru $[A]_t$. W zadaniach wieloetapowych P równy jest sumie wypłaty otrzymanej w poprzednim cyklu uczenia i maksymalnej wartości z Tabeli przewidywań zmniejszonej o rabat aktualizacji γ (krok 6). Aktualizacja dotyczy zbioru $[A]_{t-1}$ z poprzedniego cyklu uczenia. Po wyznaczeniu współczynnika aktualizacji P, można zmodyfikować wartość parametru przewidywań każdego klasyfikatora zgodnie z regułą Widrowa–Hoffa (Widrow i Hoff 1960)

[82] Jednoczesne stosowanie metody deterministycznej i stochastycznej wyboru akcji pozwala na zachowanie równowagi pomiędzy eksploatacją a eksploracją przestrzeni poszukiwań.

$$p_j \leftarrow p_j + \beta(P - p_j), \tag{16}$$

gdzie $0 < \beta \leq 1$ (krok 7a).

Aktualizacja kolejnych parametrów klasyfikatorów, czyli ε i *as*, odbywa się zgodnie z techniką *Moyenne Adaptive Modifée* (Venturini 1994), która pozwala na szybkie dochodzenie do rzeczywistych wartości średnich. Jeżeli klasyfikator był aktualizowany mniej niż $1/\beta$ razy, to nowe wartości parametrów ε i *as* są średnimi z wartości poprzedniej i aktualnej (krok 7b i 7c). W przeciwnym wypadku aktualizowane są zgodnie z regułą Widrowa–Hoffa:

$$\varepsilon_j \leftarrow \varepsilon_j + \beta(|P - p_j| - \varepsilon_j), \tag{17}$$

$$as \leftarrow as + \beta\left(\sum_{j\in[A]_{t-1}} num_j - as\right). \tag{18}$$

Kluczowy parametr klasyfikatora, czyli dopasowanie *f*, obliczany jest w trzech krokach (krok 7d). W kroku pierwszym liczona jest dokładność klasyfikatora κ. Jeżeli klasyfikator okazał się dokładny, tj. oszacowana wartość błędu przewidywania ε jest mniejsza od ε_0, to $\kappa = 1$. W przeciwnym wypadku wartość κ obliczana jest jako funkcja potęgowa trzech parametrów α, ε, v:

$$\kappa_j = \begin{cases} 1 & \text{dla} \quad \varepsilon_j < \varepsilon_0 \\ \alpha(\varepsilon_j / \varepsilon_0)^{-v} & \text{dla} \quad \varepsilon_j \geq \varepsilon_0 \end{cases}. \tag{19}$$

Następnie obliczana jest względna dokładność klasyfikatora κ' w odniesieniu do sumy dokładności pozostałych klasyfikatorów w Zbiorze akcji. W kroku trzecim aktualizowana jest wartość dopasowania klasyfikatora według reguły Widrowa–Hoffa

$$f_j \leftarrow f_j + \beta(\kappa' - f_j). \tag{20}$$

Na Zbiorze Akcji $[A]_t$ działa operator subsumcji[83] (krok 8). Zadaniem tego operatora jest eliminacja z populacji klasyfikatorów reguł redundantnych, czyli takich, które mogą być wyrażone przez inne reguły, bardziej ogólne, dokładne i odpowiednio doświadczone, czyli o parametrach $\varepsilon < \varepsilon_0$ i $exp > \theta_{sub}$. W szczególności, klasyfikator R_1 może być zastąpiony klasyfikatorem R_2, jeżeli zbiór komunikatów pasujących do R_1 jest podzbiorem komunikatów pasujących do R_2. Klasyfikator, który zastępuje inny, mniej ogólny klasyfikator, ma parametr liczności *num*, który jest zwiększany każdorazowo po operacji subsumcji (krok 8). W takiej sytuacji mówimy, że klasyfikator składa się ze zbioru mikroklasyfikatorów o liczności *num*. Głównym komponentem odkrywczym systemu XCS, obok operatora pokrycia, jest algorytm genetyczny, który

[83] Angielski termin *subsumption* tłumaczony jest w polskim wydaniu książki (Goldberg 1989) jako *obejmowanie*.

działa na populacji klasyfikatorów zawartych w Zbiorze akcji. Wywoływany jest, gdy zachodzi poniższa formuła, gdzie ts_{actual} to aktualny znacznik czasu

$$ts_{actual} - \frac{\sum_{j \in [A]} ts_j * num_j}{\sum_{j \in [A]} num_j} > \theta_{GA}. \tag{21}$$

Algorytm genetyczny przypisuje nowo utworzonym klasyfikatorom wartości $num = 1$ oraz $exp = 0$, parametry p, ε, f otrzymują za początkowe wartości średnie ich rodziców, z tym że dodatkowo dwa ostatnie parametry są zmniejszane odpowiednio o współczynnik 0,25 i 0,1 (krok 9a). Subsumcja sprawdza, czy warunek klasyfikatorów-potomków jest zawarty w warunku klasyfikatorów-rodziców. Jeżeli taka sytuacja zachodzi, to potomek nie jest dodawany do populacji, a jedynie zwiększa się liczność rodzica (krok 9b). Dodawanie nowych klasyfikatorów do populacji następuje do momentu osiągnięcia przez populację rozmiaru N. W populacji o większym rozmiarze dodawaniu nowego klasyfikatora towarzyszy procedura usuwania klasyfikatora istniejącego (krok 9c). Wybór klasyfikatora do usunięcia odbywa się za pomocą ruletki, z tym że prawdopodobieństwo wybrania klasyfikatora jest proporcjonalne do rozmiaru Zbiorów akcji, do jakich należał (określanego parametrem *as*) i wzrasta dla klasyfikatorów doświadczonych (spełniających warunek $exp > \theta_{del}$), ale o słabym przystosowaniu.

3.3.4. ACS

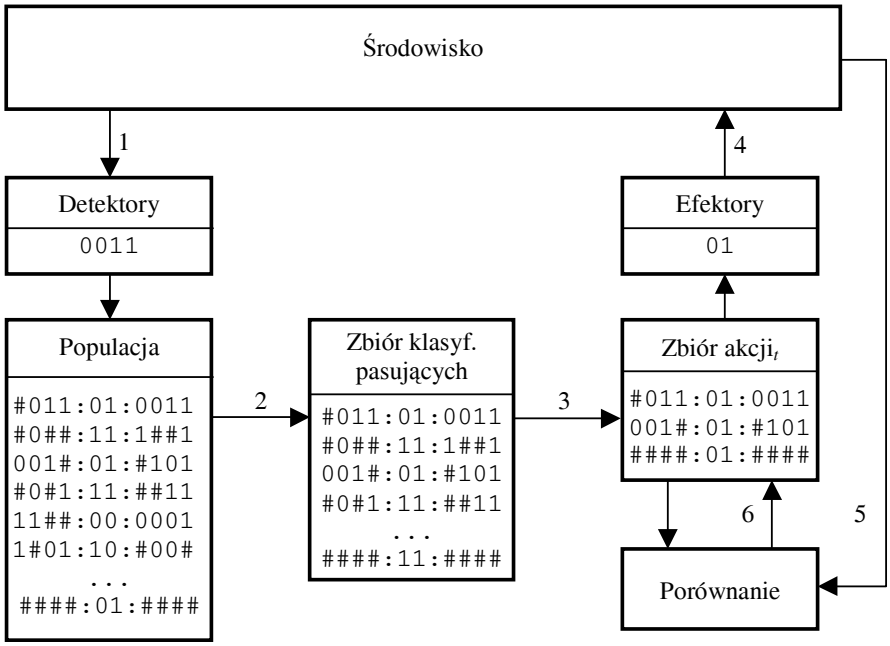
System ACS (*Anticipatory Classifier System*) został zaproponowany w 1997 r. przez Stolzmanna (1997). ACS wykazuje wiele podobieństw do prezentowanych wcześniej uczących się systemów klasyfikujących, różniąc się jednak od nich istotnie w rozszerzonej strukturze klasyfikatora oraz zastosowanym algorytmie uczenia. Podobnie jak LCS, ZCS i XCS, system ACS wchodzi w interakcje z otoczeniem, w którym wykonuje akcje dostosowywane do obserwowanego stanu otoczenia. Za wykonane akcje ACS otrzymuje skalarną nagrodę. Zasadniczym elementem różniącym ten system klasyfikujący od pozostałych jest sposób uczenia, którego podstawą nie jest otrzymana nagroda (jak w przypadku LCS i ZCS) lub precyzja jej przewidywania (XCS), lecz predykacja (antycypacja) wywołanego akcją stanu otoczenia. ACS jest próbą, dodajmy udaną, implementacji uczenia antycypacyjnego (Hoffmann 1993), które z kolei jest przykładem tzw. uczenia się utajonego[84] (*latent learning*). Teoria Hoffmanna zakłada, że każdemu zachowaniu towarzyszy przewidywanie jego konsekwencji i jeżeli owo przewidywanie jest poprawne, to relacja pomiędzy zachowaniem a przewidywaniem jest wzmacniana, w przeciwnym natomiast wypadku osłabiana; nieścisłe przewidywanie prowadzi do dalszego rozróżniania warunków samego za-

[84] Uczeniem utajonym określa się skojarzenia wyuczone dzięki doświadczeniom czy obserwacji, w czasie której nie występuje żadna zmiana w zachowaniu zewnętrznym.

chowania. W związku z tym klasyfikator systemu ACS, poza częścią warunkową i akcją, zawiera również przewidywany skutek swojego odpalenia w otoczeniu, czyli przewidywany nowy stan otoczenia. Na ACS składa się: Populacja klasyfikatorów, Detektory, Efektory, Zbiór klasyfikatorów pasujących i Zbiór akcji (rys. 15).

Klasyfikator systemu ACS jest zbiorem $\{C, A, E, M, q, r\}$, gdzie:

$C \in \{0, 1, \#\}^l$ – warunek klasyfikatora,
$A \in \{0, 1, \#\}^l$ – komunikat (akcja) klasyfikatora,
$E \in \{0, 1, \#\}^l$ – przewidywany stan otoczenia (*effect part*, *expectation part*),
M – ślad (*mark*)[85],
q – dokładność przewidywania (*quality*),
r – przewidywana wypłata (*reward prediction*).



Rys. 15. System klasyfikujący ACS (opracowanie własne)
Fig. 15. Learning classifier system ACS (own study)

Warunek C klasyfikatora definiuje możliwe stany otoczenia, do których klasyfikator może się dopasować. Zarówno w warunku klasyfikatora, jak i jego komunikacie może pojawić się symbol uniwersalny, który po stronie akcji klasyfikatora reprezentuje własność przenoszenia. Przewidywany stan otoczenia E specyfikuje atrybuty antycypowanego, nowego stanu, do którego powinno przejść otoczenie po wykonaniu

[85] Ślad został wprowadzony do klasyfikatora dopiero w późniejszych publikacjach dotyczących systemu ACS.

akcji klasyfikatora. Ślad M rejestruje wartości atrybutów stanów, dla których klasyfikator niepoprawnie zgadywał. Klasyfikator o dokładności przewidywania $q \in [0, 1]$ większej od progu θ_r nazywany jest wiarygodnym (*reliable*) i nie zostaje usunięty z populacji, klasyfikator o wartości q mniejszej od progu θ_i jest usuwany z populacji. Przewidywana przez klasyfikator wypłata po wykonaniu akcji jest zapisywana w r.

Parametry uczenia ACS są następujące:

β_r – współczynnik wzmocnienia (*reinforcement learning rate*) stosowany w aktualizacji przewidywanej wypłaty,

β_q – współczynnik antycypacji (*anticipatory learning rate*) stosowany w aktualizacji dokładności przewidywania,

γ – rabat aktualizacji (*discount factor*) przewidywania maksymalnej wypłaty,

θ_i – próg niedopasowania klasyfikatora (*inadequacy threshold*), poniżej którego klasyfikator zostaje usunięty z populacji,

θ_r – próg wiarygodności (*reliability threshold*), powyżej którego klasyfikator pozostaje w populacji,

$u_{\max}$ – próg specyficzności (*specificity threshold*), definiujący maksymalną liczbę określonych atrybutów w warunku klasyfikatora, które pozostają niezmienione w przewidywanym stanie otoczenia,

ϵ – prawdopodobieństwo eksploracji (*exploration probability*), stosowane podczas losowego wyboru akcji.

System ACS rozpoczyna swoje działanie od wypełnienia Populacji klasyfikatorów klasyfikatorami najbardziej ogólnymi z możliwych, tj. zawierającymi po stronie warunku C i przewidywanego stanu otoczenia E jedynie uniwersalne symbole #. Każda możliwa akcja w systemie reprezentowana jest przez jeden ogólny klasyfikator. Taki sposób inicjacji Populacji wynika z założenia, że na początku żadna akcja systemu nie wywoła zmiany otoczenia. Aby zapewnić obecność klasyfikatora dla każdej akcji systemu, początkowe klasyfikatory nie są nigdy usuwane z populacji.

ACS:

1. Utwórz komunikat zewnętrzny reprezentujący aktualny stan otoczenia
2. Utwórz Zbiór klasyfikatorów pasujących do komunikatu zewnętrznego
3.
   a. Wybierz akcję w sposób deterministyczny albo stochastyczny
   b. Utwórz Zbiór akcji
4. Uaktywnij akcję w otoczeniu za pośrednictwem Efektorów
5.
   a. Odbierz nagrodę od otoczenia
   b. Utwórz komunikat zewnętrzny reprezentujący następny stan otoczenia
6. Dla wszystkich klasyfikatorów w Zbiorze akcji
   a. Oblicz przewidywaną wypłatę
   b. Zastosuj uczenie antycypacyjne

Rys. 16. Pojedynczy cykl uczenia ACS
Fig. 16. Single learning cycle of ACS

Pojedynczy cykl uczenia ACS, rys. 16, rozpoczyna się od obserwacji stanu otoczenia σ_t (rys. 16 – krok 1). Następnie tworzony jest Zbiór klasyfikatorów pasujących $[M]_t$, który zawiera wszystkie klasyfikatory, ich część warunkowa pasuje do aktualnego stanu otoczenia (krok 2). Zasady uzgodnienia warunku klasyfikatora i stanu otoczenia (komunikatu zewnętrznego) są takie same, jak dla wcześniej omawianych systemów klasyfikujących, tzn. wszystkie symbole specyficzne (nie uniwersalne) są identyczne na odpowiadających sobie pozycjach. W kolejnym kroku wybierana jest akcja z $[M]_t$. Wybór akcji zależy od przyjętej w danej implementacji ACS strategii, w (Butz i in. 1999) proponowano metodę ruletki lub wybór losowy, w (Butz i in. 2002) strategię zachłanną z parametrem ϵ (krok 3a). W strategii zachłannej z parametrem ϵ wybiera się z prawdopodobieństwem ϵ akcję klasyfikatora, którego iloczyn dokładności przewidywania q i przewidywanej wypłaty r jest największy. Po wyborze akcji α_t formowany jest Zbiór akcji zawierający te klasyfikatory z $[M]_t$, które proponują akcję α_t (krok 3b). Następnie, za pośrednictwem Efektorów, wykonywana jest akcja w otoczeniu (krok 4), za którą system otrzymuje skalarną nagrodę ρ_t (krok 5a). Otoczenie przechodzi do nowego stanu σ_{t+1} (krok 5b)[86]. Nowy stan otoczenia σ_{t+1} porównywany jest z przewidywanym stanem otoczenia E klasyfikatorów ze Zbioru akcji. Na podstawie wyników porównania aktualizowana jest lista klasyfikatorów aktywnych ze zbioru $[A]_t$ za pomocą mechanizmów uczenia ze wzmocnieniem oraz uczenia antycypacyjnego (*anticipatory learning process*, ALP).

Aktualizacja parametru przewidywanej wypłaty r klasyfikatorów ze Zbioru akcji prowadzona jest zgodnie z ideą algorytmu kubełkowego (krok 6a) i opiera się na wartości otrzymanej wypłaty oraz zawartości Zbioru klasyfikatorów pasujących w kroku następnym $[M]_{t+1}$[87]

$$r \leftarrow r + \beta_r \left(\rho_t + \gamma \max_{cl \in [M]_{t+1}} (q_{cl} r_{cl}) - r \right). \tag{22}$$

Procedura uczenia ALP porównuje stan otoczenia σ_{t+1} z przewidywanym stanem otoczenia E każdego klasyfikatora należącego do Zbioru akcji. Rezultatem porównania oraz aktualnej struktury klasyfikatora jest jego modyfikacja oraz możliwa generacja nowego klasyfikatora (krok 6b). Jeżeli wygenerowany nowy klasyfikator już istnieje w populacji, to nie jest do populacji dodawany, ale zwiększana jest dokładność przewidywania q istniejącego już klasyfikatora zgodnie z regułą Widrowa–Hoffa

$$q \leftarrow q + \beta_q (1 - q). \tag{23}$$

[86] Nowy stan otoczenia σ_{t+1} odbierany jest oczywiście przez detektory systemu. Na rysunku 15 pominięto je w kroku 5 ze względu na czytelność ilustracji.

[87] Obecność we wzorze iloczynu parametrów q oraz r zbioru $[M]_{t+1}$ wynika z przyjęcia strategii zachłannej z parametem ϵ jako metody wyboru akcji.

Parametry q oraz r nowego klasyfikatora są dziedziczone po rodzicu, z tym że q nie może być niższe od wartości 0,5, gdyż wartość niższa może szybko wyeliminować nowy klasyfikator z populacji. Rozważając możliwe przypadki wyników porównania σ_{t+1} z antycypowanym stanem otoczenia E klasyfikatora, można wyróżnić trzy przypadki.

Przypadek pierwszy, nazywany bezużytecznym (*useless case*) pojawia się wtedy, gdy po wykonaniu akcji otoczenie nie przechodzi do nowego stanu, tj. $\sigma_{t+1} = \sigma_t$. W takiej sytuacji ACS zmniejsza wszystkim klasyfikatorom aktywnym dokładność przewidywania według reguły Widrowa–Hoffa

$$q \leftarrow q - \beta_q q\,. \tag{24}$$

Dodatkowo, każdy klasyfikator zapamiętuje stan, który nie przyniósł zmiany, po to by w przyszłości ACS mógł stosować tylko klasyfikatory mające szanse przynieść zmianę w otoczeniu (z reguły tej wyłączone są ogólne klasyfikatory początkowe)[88]. Za zapamiętywanie stanów odpowiada ślad klasyfikatora M zapamiętujący na poszczególnych pozycjach wartości atrybutów kolejnych stanów, dla których klasyfikator niepoprawnie zgadywał.

Kolejnym przypadkiem, jaki może się pojawić podczas porównywania stanu otoczenia i przewidywanego przez klasyfikator stanu otoczenia, może być tzw. przypadek nieoczekiwany (*unexpected case*), w którym klasyfikator niepoprawnie przewidział stan otoczenia. Zachodzi on wtedy, gdy jedna lub więcej wartości atrybutów stanu otoczenia σ_{t+1} różnią się od przewidywanych. W takim wypadku stan jest zapamiętywany przez klasyfikator w swoim śladzie, a dokładność przewidywania zmniejszana jest analogicznie jak w przypadku bezużytecznym. Po modyfikacji klasyfikatora może zostać wygenerowany nowy klasyfikator, jeżeli tylko, po zastąpieniu symboli uniwersalnych w części C oraz E klasyfikatora-rodzica przez symbole określone, można otrzymać prawidłową antycypację (jest to tzw. przypadek dających się skorygować założeń i oczekiwań (*case of correctable assumptions and expectations*)).

Ostatnim rezultatem porównania jest przypadek poprawnej antycypacji przez klasyfikator kolejnego stanu otoczenia (*expected case*). Jeżeli klasyfikator nie miał wypełnionego śladu, to jego dokładność przewidywania q zwiększa się zgodnie z równaniem (23). Wypełniony ślad klasyfikatora świadczy o istnieniu stanów, w których klasyfikator poprawnie nie przewidywał. Jeżeli zatem można znaleźć różnicę pomiędzy śladem klasyfikatora a aktualnym stanem systemu, to generowany jest nowy klasyfikator, który różni się od stanu lub stanów zapisanych w śladzie klasyfikatora-rodzica. To różnicowanie polega na zamianie maksymalnie $u_{\max}$ symboli uniwersalnych w warunkach klasyfikatora (części C oraz E) na wartości określone.

Kończąc omawianie systemu ACS, warto zauważyć, że jest to przykład uczącego się systemu klasyfikującego, który odkrywa nowe reguły nie za pomocą algorytmu

[88] To założenie wynika z teorii uczenia, leżącej u podstaw ALP, wedle której każdej akcji powinna towarzyszyć zmiana w percepcji środowiska.

genetycznego, ale pewnych heurystyk. Ostatnie propozycje architektury ACS zawierają już połączenie uczenia ALP oraz GA (Butz i in. 2002).

3.4. Zastosowania

Środowisko autonomicznych robotów zawsze było uważane za dobry obszar testowy dla wszelkich algorytmów uczenia ze wzmocnieniem. Właśnie w tym obszarze obserwuje się najwięcej zastosowań uczących się systemów klasyfikujących. W latach 90. ubiegłego wieku został opracowany, a następnie sprawdzony w wielu wersjach i aplikacjach, równoległy system uczący Alecsys, autorstwa zespołu kierowanego przez Dorigo i Colombetti (1998). W systemie tym zaimplementowano oryginalną ideę uczenia BAT (*Behavior Analysis and Training*), polegającą na przyrostowym uczeniu się autonomicznego agenta. Detale implementacyjne systemu Alecsys można znaleźć w (Dorigo i Sirtori 1991, Dorigo 1991, Dorigo i Schnepf 1993, Dorigo 1995), metodologia uczenia BAT oraz właściwa jej technika uczenia „Robot Shaping" zostały przedstawione w (Dorigo i Colombetti 1994, Colombetti i Dorigo 1994, Colombetti i in. 1996, Colombetti i Dorigo 1999). Przykładem innego systemu klasyfikującego sterującego autonomicznym agentem jest MonaLysa (Donnart i Meyer 1994, 1996). MonaLysa jest hierarchicznym systemem, łączącym system klasyfikujący z innymi modułami, np. modułem planowania. Stosunkowo wiele systemów autonomicznych robotów zrealizowano z udziałem rozmytych systemów klasyfikujących, np. (Bonarini 2000).

Innym ważnym obszarem zastosowań LCS jest uczenie z nadzorem (*supervised classification*). W tym typie uczenia informacją trenującą jest seria etykietowanych przykładów reprezentujących poszukiwany model zachowania. Zadaniem algorytmu uczącego jest indukcja ogólnego i kompaktowego modelu, opisującego nie tylko obserwowane przykłady, ale również klasyfikującego poprawnie przykłady wcześniej nieobserwowane. Stosowanie uczących się systemów klasyfikujących w zadaniach uczenia z nadzorem ma na celu bądź modelowanie kognitywistyczne, bądź też analizę i eksplorację danych. W pierwszej grupie zastosowań zadaniem systemów klasyfikujących jest modelowanie procesów kategoryzacji wykonywanych przez człowieka. System Riola (1990) symulował proces planowania i tzw. uczenia się utajonego (*latent learning*). Jakościowo podobne rezultaty uzyskał Sen (1996), który również badał problemy filtrowania i kondensacji. Hartley (1999) porównywał natomiast działanie dwóch różnych systemów klasyfikujących NEWBOOLE i XCS, w tym samym zadaniu kategoryzacji. Stwierdził, że metoda liczenia dostosowania klasyfikatora ma istotny wpływ na ewoluowaną wiedzę, a obliczanie dokładności klasyfikatora (jak w systemie XCS) daje bardziej kompletny model przestrzeni poszukiwań[89] niż obliczanie jego siły (jak w syste-

[89] Wilson (1995) używa pojęcia kompletnego mapowania (*complete mapping*).

mie NEWBOOLE). Stosowanie systemów klasyfikujących w problemach eksploracji danych daje możliwość uzyskania zwartych opisów wielowymiarowych najczęściej danych, które poprawiają efektywność przeszukiwań[90]. W (Bonelii i Parodi 1991) pokazano, że system NEWBOOLE działa znacząco lepiej od algorytmu CN2 i sieci neuronowej w drążeniu danych medycznych. Równie dobre rezultaty osiągnął Sen (1993), porównując działanie zmodyfikowanego NEWBOOLE'a z innymi metodami uczenia maszynowego na zestawie powszechnie używanych danych testowych. Saxon i Barry (1999) zbadali natomiast działanie systemu XCS w rozwiązywaniu znanego w uczeniu maszynowym problemu Monk. Również i w tym przypadku system klasyfikujący okazał się efektywniejszy od większości znanych technik uczenia. Obecnie chyba najbardziej znanym systemem klasyfikującym dedykowanym zadaniom analizy danych jest EpiCS Holmesa (1996). EpiCS jest modyfikacją NEWBOOLE'a przeznaczoną do analizy danych epidemiologicznych, choć obecnie znajduje także zastosowanie w odkrywaniu wiedzy w różnorodnych bazach klinicznych (Holmes 2000).

Kolejnym obszarem zastosowań uczących się systemów klasyfikujących, po robotyce i uczeniu z nadzorem, jest ekonomia obliczeniowa (*computational economics*). Modelowanie adaptacyjnych agentów w środowisku sztucznych rynków giełdowych było przedmiotem prac (Arthur i in. 1996, LeBaron i in. 1999). Vriend (2000) porównywał działanie systemów klasyfikujących typu Michigan oraz Pitt w tym samym modelu ekonomicznym. W (Marimon i in. 1990) symulowano rozwój pieniądza przy użyciu populacji systemów klasyfikujących, a Bull (1999) zastosował zmodyfikowany system ZCS w symulacji rynku ciągłych aukcji CDA (*continuous double-auction*). W grupie aplikacji systemów klasyfikujących w modelach ekonomicznych warto jeszcze wspomnieć o (Miller i Holland 1991) oraz (Mitlöhner 1996).

Uczące się systemy klasyfikujące stosuje się również w modelowaniu ludzkiego procesu uczenia: w środowisku dwujęzycznym (Satterfield 1999) oraz uczenia się języków (Druhan i Mathews 1989); Davis (2000) zaproponował natomiast obliczeniowy model teorii afektów (*Affect Theory*), stosowanej w psychiatrii i psychologii.

Z innych zastosowań warto wymienić pracę (Federman i in. 1998), w której system klasyfikujący przewidywał następną nutę dla różnych typów muzyki. W (Richards i in. 1992, 1996) zastosowano LCS w optymalizacji kształtów dwu- i trzywymiarowych, optymalizacją kształtów zajmowali się również (Nagasaka i Taura 1997). Smith i in. (1999) ewoluowali strategie walki powietrznej; Sanza i in. (1998) zastosowali agenta wyposażonego w system klasyfikujący w nawigacji w wirtualnym otoczeniu. Frey i Slate (1991) rozpoznawali litery, a Cao i in. (1999) oraz Escazut i Fogarty (1997) konstruowali sterowniki sygnalizacji miejskiej.

[90] XCS w wersji Greenvera (2000) zajął trzecie miejsce w zawodach w eksploracji danych The 2000 European Network on Computational Intelligence (COIL), ulegając jedynie technikom stosującym dedykowane zadaniom heurystyki.

4. Model GCS

Zasadniczym przeznaczeniem prezentowanego tu nowego modelu ewolucyjnego jest indukcja gramatyk bezkontekstowych. Wnioskowanie gramatyk bezkontekstowych to ważne zagadnienie ze względów praktycznych, bowiem gramatyka bezkontekstowa może modelować zarówno strukturę języków programowania (Aho i in. 1986), języków naturalnych (Jurafsky i Martin 2000), jak i dane biologiczne, np. sekwencje nukleotydów tworzących DNA i RNA (Durbin i in. 1998). Z teoretycznego punktu widzenia problem indukcji CFG stanowi wciąż prawdziwe wyzwanie, ze względu na udowodnione ograniczenia możliwości uczenia. Zgodnie ze wzmiankowanym już wcześniej twierdzeniem Golda, niemożliwa jest indukcja dowolnego języka z hierarchii Chomsky'ego jedynie na podstawie przykładów poprawnych. Co więcej, uzupełnianie zbioru uczącego o odpowiednie zapytania nie gwarantuje identyfikacji gramatyki bezkontekstowej w czasie wielomianowym. Algorytmom uczącym dostarcza się zatem dodatkowej informacji w postaci przykładów spoza indukowanego języka (negatywnych), zbiory uczące niosą ze sobą dodatkową informację strukturalną, formułuje się alternatywne reprezentacje gramatyk bezkontekstowych, ogranicza się zadanie uczenia do pewnych podklas języka, wreszcie stosuje się metody bayesowskie. Jednak efektywne, wielomianowe algorytmy istnieją jak dotąd jedynie dla wyrażeń regularnych. Tak więc konstrukcja metod uczenia gramatyk bezkontekstowych jest obecnie krytycznym i wciąż otwartym problemem wnioskowania gramatycznego. W indukcji gramatyk bezkontekstowych próbuje się wykorzystać m.in. różne warianty metod ewolucyjnych czy też metod statystycznych. Publikowane wyniki badań dotyczą najczęściej języków sztucznych, rzadko korpusów językowych.

Przedmiotem następnych rozdziałów monografii jest autorski model GCS, wykorzystujący wnioskowanie gramatyczne w indukcji gramatyk bezkontekstowych.

4.1. Zadanie klasyfikacji

GCS (*Grammar-based Classifier System*, System klasyfikujący zorientowany na gramatykę) jest nowym modelem ewolucyjnym przeznaczonym do indukcji CFG. Nazwa systemu określa zarówno jego zasadnicze zadanie, jak i architekturę, gdyż

wiedza o rozwiązywanym zadaniu reprezentowana jest bezpośrednio, w postaci zbioru produkcji gramatyki bezkontekstowej.

W przypadku zadań klasyfikacyjnych, a do takich można zaliczyć zadanie uczenia gramatyki nieznanego języka, otoczeniem systemu jest zbiór uczący zawierający etykietowane przykłady. Każdy przykład jest opisany przez wektor atrybutów oraz jednoznaczne przypisanie do określonej klasy. Celem uczenia jest otrzymanie takich klasyfikatorów, które będą w stanie z odpowiednią dokładnością sklasyfikować nieznane wcześniej przykłady. Jeżeli klasyfikator przedstawimy w postaci pojedynczej produkcji gramatyki bezkontekstowej, a zbiór klasyfikatorów potraktujemy jako gramatykę, to poprawna klasyfikacja etykietowanych zdań oznaczać będzie wyindukowanie poszukiwanej gramatyki języka. Zgodnie z twierdzeniem Golda (1967) indukcja gramatyki języka, począwszy od wyrażeń regularnych, nie jest możliwa jedynie na podstawie przykładów pozytywnych, zatem przykłady muszą obejmować zarówno klasę zdań należących do języka, jak i spoza języka.

Formalnie, zadanie indukcji CFG na podstawie etykietowanych zdań (przykładów) można zapisać w sposób następujący. Niech poszukiwaną gramatyką bezkontekstową będzie $G = (N, T, P, S)$. Skończony zbiór zdań uczących R jest zbiorem par (r, e), gdzie $r \in T^*$ jest zdaniem, a $e \in \{+, -\}$ jest etykietą zdania określającą, czy zdanie odpowiednio należy do języka $r \in L(G)$ czy też nie należy $r \notin L(G)$. Zbiór zdań uczących R składa się zatem z dwóch rozłącznych zbiorów R^+ oraz R^-, takich że:

$$R = R^+ \cup R^-, \tag{25}$$

$$R^+ = \{(r, +) \mid r \in T^*\}, \tag{26}$$

$$R^- = \{(r, -) \mid r \notin T^*\}, \tag{27}$$

$$R^+ \cap R^- = \emptyset. \tag{28}$$

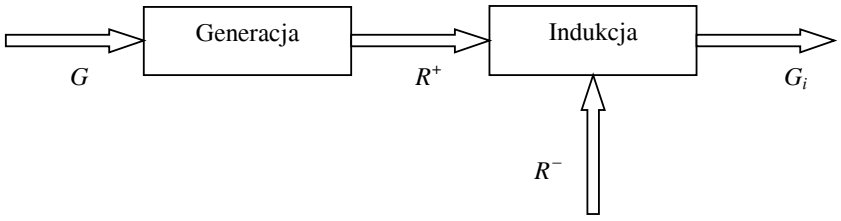
Indukcja gramatyki G jest procesem poszukiwania takiej gramatyki G_i (gramatyki indukowanej), dla której:

$$L(G_i) \cap R^+ = R^+, \tag{29}$$

$$L(G_i) \cap R^- = \emptyset. \tag{30}$$

Oczekuje się, że gramatyka G_i będzie w stanie generalizować przykłady zdań z R^+, definiując język $L(G_i) \supseteq R^+$. W idealnym przypadku $L(G_i) = L(G)$. W praktycznych zastosowaniach indukcji jest to najczęściej niemożliwe i dąży się do otrzymania gramatyki G_i akceptującej możliwie największy podzbiór $L(G)$ i jednocześnie możliwie

najmniejszy podzbiór $\{T^* - L(G)\}$. Zbiór R^+ powinien być strukturalnie kompletny (*structurally complete*), tzn. przy jego wyprowadzaniu (generacji) powinna być użyta każda produkcja gramatyki języka, którego jest podzbiorem. Od strukturalnej kompletności zbioru uczącego w dużej mierze zależy skuteczność inferencji gramatyki. Zbiór R^- jest najczęściej tworzony losowo. Schematycznie, proces indukcji gramatyki przedstawiono na rys. 17.



Rys. 17. Model indukcji gramatyki na podstawie etykietowanych przykładów
Fig. 17. Model of grammatical inference from labelled examples

Oczekiwanym efektem uczenia w modelu GCS jest wyewoluowanie gramatyki, która akceptuje zdania poprawne, odrzuca zdania niepoprawne oraz potrafi poprawnie sklasyfikować nieznane wcześniej zdania testowe, tzn. zaakceptować zdania należące do ewoluowanego języka, a odrzucić zdania spoza języka (tzw. testy generalizacji).

4.2. Werbalny opis modelu

Działanie modelu GCS można opisać, posługując się kategoriami uczącego się systemu klasyfikującego. W takim ujęciu zasada pracy GCS jest podobna do kanonicznego uczącego się systemu klasyfikującego pracującego w reżimie zadania wieloetapowego, ale różni się od niego istotnie:

- środowiskiem,
- reprezentacją populacji klasyfikatorów,
- metodą dopasowywania klasyfikatorów do stanu środowiskowego,
- metodami odkrywania nowych klasyfikatorów.

Populację klasyfikatorów modelu GCS tworzą produkcje gramatyki bezkontekstowej w PNC. Model ewoluuje jedną gramatykę zgodnie z podejściem Michigan. Przedmiotem operacji genetycznych są w tej metodzie indywidualne klasyfikatory – a w przypadku GCS pojedyncze produkcje gramatyki. Wszystkie klasyfikatory (produkcje) systemu tworzą populację osobników ewoluujących w czasie. W każdym kolejnym kroku pętli ewolucyjnej model GCS rozwiązuje wieloetapowe zadanie oceny

indukowanej gramatyki. Pojedynczy etap oceny związany jest z parsowaniem jednego zdania ze zbioru zdań uczących. Z kolei na każdy etap składa się wiele cykli, w czasie których algorytm parsujący dokonuje rozbioru zdania.

Parsowanie zestawu uczącego realizowane jest przez działający w czasie sześciennym algorytm CYK. Tablica parsera CYK pełni funkcje środowiska systemu klasyfikującego. W tablicy o rozmiarach $n \times n$, gdzie n jest długością analizowanego zdania wejściowego, zapisywana jest pełna historia rozbioru zdania uczącego bądź testowego. Obecność symbolu nieterminalnego w komórce $[i, j]$ oznacza, że symbol ten wyprowadza część zdania wejściowego o długości j, poczynając od pozycji i. Jeżeli rozbiór kończy się umieszczeniem w komórce $[1, n]$ symbolu startowego S, to oznacza, że gramatyka jest w stanie wyprowadzić całe zdanie. W każdym pojedynczym cyklu pracy parsera CYK następuje po kolei:

a) wygenerowanie prawych stron produkcji pasujących do aktualnego etapu parsowania,

b) dopasowywanie prawych stron wygenerowanych produkcji do istniejącej populacji produkcji,

c) umieszczenie w aktualnej komórce tablicy znalezionych lewych stron produkcji.

Umieszczenie w komórce tablicy więcej niż jednego symbolu nieterminalnego, oznaczającego znalezioną lewą stronę produkcji, sygnalizuje pojawienie się alternatywnych drzew rozbioru.

Każda produkcja, która bierze udział w rozbiorze, otrzymuje po przeanalizowaniu przez system pełnego zestawu uczącego miarę swojej użyteczności (przystosowania), która uwzględnia liczbę zastosowań produkcji podczas parsowania zdań poprawnych i niepoprawnych. Dodatkowo, funkcja oceny klasyfikatora może brać pod uwagę położenie produkcji w ciągu rozbioru zdania. Miara użyteczności produkcji wykorzystywana jest przez algorytm genetyczny w poszukiwaniu nowych klasyfikatorów.

Model GCS, podobnie jak większość systemów klasyfikujących, korzysta z dwóch mechanizmów, które pozwalają na odkrywanie nowych reguł. Pierwszym z nich jest algorytm genetyczny, drugim mechanizm pokrycia.

Algorytm genetyczny działa w modelu GCS na populacji klasyfikatorów, stosując różne typy selekcji, krzyżowanie, mutację oraz inwersję, a także metodę ścisku jako sposób zachowania różnorodności w populacji. Operatory genetyczne są uruchamiane z określonym prawdopodobieństwem dopiero po zakończeniu analizy pełnego zestawu uczącego.

Mechanizm pokrycia działa, po pierwsze, niezależnie od algorytmu genetycznego, po drugie, w trakcie parsowania zdania. Dodaje on do populacji klasyfikatorów takie produkcje, które w danej sytuacji środowiskowej umożliwiają dalszy rozbiór zdania.

Ocena przystosowania całej wyewoluowanej gramatyki dokonywana jest po przeanalizowaniu całego zestawu uczącego i bierze pod uwagę liczbę zaakceptowanych zdań poprawnych oraz odrzuconych zdań negatywnych.

4.3. Klasyfikator

Model GCS jest ewolucyjnym systemem uczącym się gramatyki bezkontekstowej, którego działanie można przedstawić, posługując się pojęciami uczącego się systemu klasyfikującego. Podstawowym elementem konstytuującym uczący się system klasyfikujący jest klasyfikator.

4.3.1. Definicja

Klasyfikator GCS jest układem $cl = (P_L, P_P, f, u_p, u_n, p, d)$, gdzie:

$P_P \mid P_L \rightarrow P_P, P_L \in N, P_P: a \vee P_P: XY, a \in T, X, Y \in N$

– prawa strona produkcji, czyli warunek klasyfikatora (symbol terminalny lub para symboli nieterminalnych),

P_L – lewa strona produkcji, czyli komunikat (akcja) klasyfikatora (symbol nieterminalny),

f – przystosowanie (*fitness*),

u_p – liczba zastosowań produkcji przy parsowaniu zdań poprawnych $r \in R^+$,

u_n – liczba zastosowań produkcji przy parsowaniu zdań niepoprawnych $r \in R^-$,

p – suma punktów zdobytych przez klasyfikator podczas parsowania zdań poprawnych $r \in R^+$ (*profit*),

d – suma punktów zdobytych przez klasyfikator podczas parsowania zdań niepoprawnych $r \in R^-$ (*debt*).

Klasyfikator cl w modelu GCS jest zatem produkcją gramatyki bezkontekstowej $P: P_L \rightarrow P_P$ z własnym przystosowaniem f oraz towarzyszącymi produkcji parametrami $Par_{cl} = \{u_p, u_n, p, d\}$. Symbolem $Par_{cl}(G)$ oznaczać będziemy zbiór parametrów wszystkich klasyfikatorów gramatyki G, czyli

$$Par_{cl}(G) = \bigcup_{cl \in G} Par_{cl}. \tag{31}$$

4.3.2. Płodność

O ile interpretacja parametrów u_p i u_n jest oczywista, o tyle parametery p i d wymagają wyjaśnienia. Każdy klasyfikator w modelu GCS podlega ewolucyjnej modyfikacji w czasie. Jednocześnie klasyfikator dopiero wspólnie z pozostałymi klasyfikatorami tworzącymi populację reprezentuje poszukiwane rozwiązanie zadania. Co więcej, klasyfikatory wchodzą pomiędzy sobą w interakcje, gdy jako produkcje gramatyki tworzą ciąg wyprowadzeń dla parsowanego zdania. Zjawisko to nosi nazwę

epistazy[91]. Stąd też nawet niewielka modyfikacja klasyfikatora może spowodować aktywację lub dezaktywację całego zbioru klasyfikatorów tworzących łańcuch wyprowadzeń, na czele którego stoi modyfikowany klasyfikator. Dodatkowym czynnikiem zwiększającym współzależność klasyfikatorów jest ich reprezentacja w PNC. Postać normalna Chomsky'ego wymusza stosowanie krótkich reguł, a zatem zapisanie nawet stosunkowo prostych reguł gramatycznych wymaga użycia kilku współdziałających ze sobą produkcji[92]. Jedną z metod łagodzenia negatywnych skutków wysokiej epistazy[93] w modelu GCS jest uwzględnianie w funkcji przystosowania klasyfikatora pozycji produkcji w ciągu wyprowadzeń, tzw. płodności (*fertility*). Produkcje stojące wyżej w ciągu wyprowadzeń, bardziej płodne, a jednocześnie biorące udział w rozbiorze zdań poprawnych, powinny być mniej narażone na modyfikację lub usunięcie. Każdy klasyfikator przepisujący symbole terminalne (typu $A \rightarrow a$) wnosi do systemu określoną kwotę bazową *ba* (*base amount*). Każdy klasyfikator, który znajdzie się wyżej w drzewie rozbioru, pobiera od klasyfikatorów bezpośrednio podrzędnych określoną przez współczynnik *raf* (*renounced amount factor*) część ich kwot. Im więcej rozwinięć danego symbolu, tym większą kwotę zbierze dany klasyfikator od swoich podrzędnych klasyfikatorów. Przykładowo, aby policzyć, jaką kwotę zebrał klasyfikator $A \rightarrow BC$, musimy znać kwoty wszystkich klasyfikatorów, które wyprowadzają symbol A oraz B. Uzbierana przez dany klasyfikator „kwota" jest zaliczana po stronie „ma" (parametr p), jeśli analizowane zdanie było przykładem pozytywnym lub po stronie „winien" (parametr d), jeśli zdanie było przykładem negatywnym. Uwzględnianiem płodności klasyfikatora w funkcji jego oceny można sterować poprzez odpowiednie współczynniki wzoru na przystosowanie klasyfikatora.

4.3.3. Przystosowanie

Każdy klasyfikator otrzymuje po przeanalizowaniu przez model pełnego zestawu uczącego (wszystkich zdań ze zbioru R) miarę swojej użyteczności (przystosowania) f liczoną według wzoru

---

[91] W sensie biologicznym jest to zjawisko polegające na oddziaływaniu *genu* (genu epistatycznego, supresora) na ujawnianie się innego genu (genu hipostatycznego) niebędącego jego *allelem*. Pojęcie epistazy jest blisko związane ze zjawiskiem plejotropowości i poligeniczności, które są modelowane w *K*-modelu Kwaśnickiej (1999).

[92] Wyard (1994) w drodze eksperymentalnej pokazał, że algorytm ewolucyjny przetwarzający gramatykę bezkontekstową w notacji BNF działa efektywniej niż z innymi reprezentacjami. Fakt ten tłumaczy się większą przestrzenią poszukiwań innych reprezentacji, jako wynik bardziej złożonego zapisu gramatyk. Należy jednak pamiętać, że to właśnie dla notacji PNC istnieją najbardziej efektywne algorytmy parsowania, a ograniczona prawa strona produkcji ułatwia projektowanie operatorów genetycznych.

[93] Badania empiryczne dowodzą, że algorytmy genetyczne są odpowiednie dla zadań o średniej epistazie. Co prawda konwencjonalne, nieewolucyjne metody rozwiązują problemy o małej epistazie dużo efektywniej od metod opartych na ewolucji, ale praktycznie nie radzą sobie z zadaniami o wysokiej epistazie (Hoffmann 1997).

$$f = \frac{w_c f_c + w_f f_f}{w_c + w_f}, \tag{32}$$

gdzie:

$$f_c = \begin{cases} \dfrac{w_p u_p}{w_n u_n + w_p u_p} & \text{dla} \quad u_n + u_p \neq 0 \\ f_0 & \text{dla} \quad u_n + u_p = 0 \end{cases}, \tag{33}$$

$$f_f = \frac{p - d - f_{f\min}}{f_{f\max} - f_{f\min}}, \tag{34}$$

$$f_{f\max} = \max_{cl \in G}(p - d), \tag{35}$$

$$f_{f\min} = \min_{cl \in G}(p - d), \tag{36}$$

f_0 – miara użyteczności klasyfikatora niebiorącego udziału w parsowaniu,
f_c – klasyczna funkcja przystosowania,
f_f – funkcja płodności klasyfikatora,
u_p – liczba zastosowań produkcji przy parsowaniu zdań poprawnych $r \in R^+$,
u_n – liczba zastosowań produkcji przy parsowaniu zdań niepoprawnych $r \in R^-$,
p – suma punktów zdobytych przez klasyfikator podczas parsowania zdań poprawnych $r \in R^+$,
d – suma punktów zdobytych przez klasyfikator podczas parsowania zdań niepoprawnych $r \in R^-$,
w_p – waga rozbioru zdania poprawnego $r \in R^+$,
w_n – waga rozbioru zdania niepoprawnego $r \in R^-$,
w_c – waga funkcji klasycznej,
w_f – waga funkcji płodności,
$f_{f\max}$ – maksymalna liczba zdobytych punktów przez różnicę $(p - d)$ w populacji klasyfikatorów,
$f_{f\min}$ – minimalna liczba zdobytych punktów przez różnicę $(p - d)$ w populacji klasyfikatorów.

Funkcja f_f jest znormalizowaną funkcją płodności klasyfikatora. Normalizacja f_f dokonywana jest w stosunku do maksymalnej i minimalnej różnicy $(p - d)$ w populacji. Funkcja f_c jest „klasyczną" funkcją przystosowania klasyfikatora, uwzględniającą liczbę zastosowań produkcji w rozbiorze zdań poprawnych i niepoprawnych. Wagi w_p oraz w_n pozwalają na sterowanie wpływem liczby rozbiorów odpowiednio zdań pozytywnych i negatywnych na przystosowanie klasyfikatora. Wreszcie funkcja przystosowania kla-

syfikatora *f* jest ważoną funkcją funkcji składowych, klasycznej i płodności. Odpowiednie ustawienie wag w_c oraz w_f pozwala na regulację istotności każdej ze składowych funkcji, odpowiednio klasycznej i płodności, na ostateczną wartość funkcji przystosowania.

Miara przystosowania produkcji *f* wykorzystywana jest przez algorytm genetyczny w ewolucyjnym poszukiwaniu najlepszych klasyfikatorów.

4.4. Gramatyka

W formalnym ujęciu klasyfikator *cl* modelu GCS jest produkcją gramatyki bezkontekstowej

$$G_{GCS} = (N, T, P^T, P^N, S, S_u), \tag{37}$$

gdzie:

N – zbiór symboli nieterminalnych,
T – zbiór symboli terminalnych,
P^T – produkcje terminalne, $P^T = \{A \rightarrow a \mid A \in N, a \in T\}$,
P^N – produkcje nieterminalne, $P^N = \{A \rightarrow BC \mid A, B, C \in N\}$,
S – wyróżniony symbol startowy $S \in N$,
S_u – wyróżniony symbol uniwersalny $S_u \in N$.

W stosunku do klasycznej definicji gramatyki bezkontekstowej (patrz wzór (10)) powyższa wprowadza dwa rozszerzenia, wynikające z natury omawianego modelu.

Po pierwsze, zbiór produkcji P został podzielny na dwa rozłączne zbiory P^T i P^N. Rozdział obu zbiorów produkcji jest próbą zapobieżenia szybkiego powstawania gramatyk bezużytecznych podczas ewolucji gramatyki. Indukowana gramatyka ma szanse na rozpoznanie wejściowego ciągu jedynie w sytuacji, gdy istnieje co najmniej jedna reguła postaci $A \rightarrow a$, dla każdego symbolu terminalnego. Efektem zastosowania operatorów genetycznych (krzyżowania, mutacji, inwersji) może być z dużym prawdopodobieństwem utrata niektórych reguł terminalnych, a w konsekwencji zniszczenie całej gramatyki. Owszem, można sobie wyobrazić takie operatory genetyczne, które by chroniły lub naprawiały produkcje terminalne, jednak działoby się to kosztem dodatkowego czasu obliczeniowego oraz zwiększaniem długości gramatyki lub – w przypadku ograniczania długości gramatyki – zwiększaniem liczby produkcji terminalnych kosztem nieterminalnych.

Drugim rozszerzeniem w definicji gramatyki bezkontekstowej modelu GCS jest wyróżnienie dodatkowego symbolu nieterminalnego S_u oznaczającego symbol uniwersalny, który przez analogię do symbolu # stosowanego w kanonicznym LCS, zastępuje (wyprowadza) dowolny symbol terminalny. Jeżeli w modelu GCS poprzez ustawienie zmiennej systemowej f_{Su} wyrażamy zgodę na stosowanie podczas parso-

wania symbolu uniwersalnego, to razem z produkcją terminalną typu $A \rightarrow a$ powstaje produkcja $S_u \rightarrow a$.

Ocena przystosowania całej ewoluowanej gramatyki jest obliczana według poniższego wzoru na zakończenie każdego kroku pętli ewolucyjnej

$$f_G = \frac{U_p + NU_n}{|R|}, \tag{38}$$

gdzie:

U_p – liczba sparsowanych zdań poprawnych,
NU_n – liczba niesparsowanych zdań niepoprawnych,
$|R|$ – liczba zdań w zestawie uczącym.

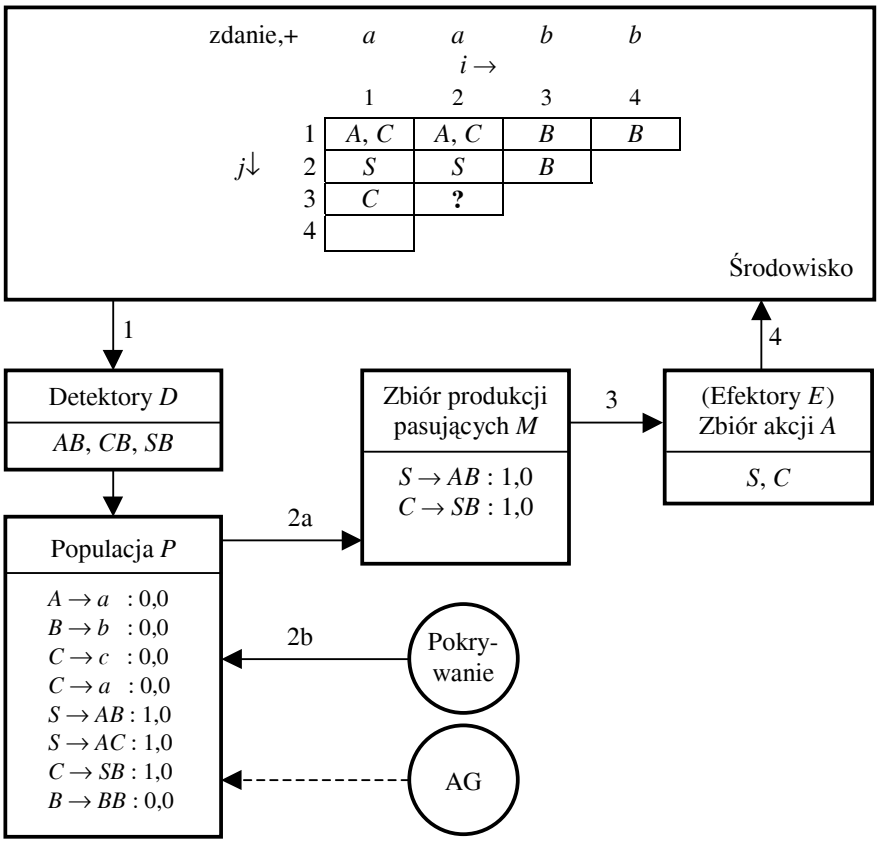
4.5. Architektura

W modelu GCS poza Populacją klasyfikatorów P, których definicja została podana w poprzednim podrozdziale, wyróżnić można Zbiór produkcji pasujących M, Zbiór akcji A (Efektorów E[94]) oraz zbiór Detektorów D, za pomocą których GCS kontaktuje się ze środowiskiem. Architekturę modelu przedstawiono na rys. 18.

Zadaniem uczącego się systemu klasyfikującego jest indukcyjne poszukiwanie populacji reguł, która będzie poprawnie reagować na przychodzące ze środowiska bodźce. W przypadku modelu GCS środowiskiem jest tablica parsera CYK, a poprzez poprawną reakcję będziemy rozumieli parsowanie zdania ze zbioru R^+, czyli umieszczenie symbolu startowego gramatyki S w komórce [1, n] tablicy CYK, gdzie n jest długością zdania wejściowego, a dla zdania ze zbioru R^- brak symbolu S w tejże komórce tablicy.

Jak już wspomniano wcześniej, uczenie gramatyki bezkontekstowej, reprezentowanej poprzez populację produkcji P, odbywa się w pętli ewolucyjnej, podczas której następuje wieloetapowa adaptacja gramatyki do uczącego zbioru R. Każdy etap adaptacji związany jest z rozbiorem jednego zdania ze zbioru uczącego, a każdy etap składa się z wielu cykli, podczas których algorytm parsujący wypełnia tablicę CYK. Na rysunku 18, poza architekturą modelu GCS, przedstawiono również cykl, podczas którego wypełniana jest komórka [2, 3] tablicy CYK dla $G = (\{A, B, C, S\}, \{a, b, c\}, \{S \rightarrow AB, S \rightarrow AC, C \rightarrow SB, C \rightarrow a, B \rightarrow BB, B \rightarrow b, A \rightarrow a\}, S)$ oraz łańcucha wejściowego $aabb \in R^+$. Dla uproszczenia rysunku, w populacji produkcji P pokazano przykładowe wartości jedynie dwóch parametrów u_p i u_n.

[94] Zbiór Efektorów w modelu GCS jest tożsamy ze Zbiorem akcji A, a zaznaczamy go w architekturze modelu GCS, aby zachować zgodność z prezentowanymi wcześniej modelami uczących się systemów klasyfikujących.



Rys. 18. System klasyfikujący GCS
Fig. 18. Learning classifier system GCS

Na rysunku 19 przedstawiono kolejne kroki wykonywane w modelu GCS podczas pojedynczego cyklu uczenia. Na początku każdego cyklu uczenia model otrzymuje poprzez Detektory *D* komunikat zewnętrzny reprezentujący aktualny stan środowiska (rys. 19 – krok 1). Zgodnie z działaniem algorytmu CYK (patrz rys. 1) na aktualny stan środowiska, przekazywany do zbioru Detektorów *D*, składa się zbiór „dwójek" symboli nieterminalnych $D = \{AB, CB, SB\}$. W kroku 2a następuje wypełnienie Zbioru produkcji pasujących *M* tymi klasyfikatorami ze zbioru *P*, dla których prawa strona $P_P \in D$. Jeżeli wynikiem dopasowania stanu otoczenia do populacji klasyfikatorów jest zbiór pusty, następuje wywołanie odpowiedniego operatora pokrycia (krok 2b). Operatory pokrycia omówione zostaną w następnym rozdziale.

W kroku 3 formułowany jest Zbiór akcji *A*, który składa się z niepowtarzających się lewych stron produkcji ze zbioru *M*. Jeżeli zbiór ten zawiera więcej niż jeden symbol nieterminalny, to oznacza, że istnieją alternatywne rozbiory analizowanego ciągu. W przypadku przedstawionym na rys. 18 dla komórki tablicy CYK [2, 3], zbiór

$A = \{S, C\}$. Oznacza to, że podciąg r_{23} analizowanego zdania, czyli ciąg *abb*, można wyprowadzić zarówno z symbolu S, jak i C[95]. W ostatnim kroku pojedynczego cyklu uczenia GCS (krok 4) następuje uaktywnienie symboli ze Zbioru akcji A pełniącego funkcję Efektorów E w środowisku, czyli wpis zawartości zbioru A do analizowanej komórki tablicy CYK.

GCS:

1. Utwórz komunikat zewnętrzny reprezentujący aktualny stan otoczenia
2.
   a. Utwórz Zbiór produkcji pasujących do komunikatu zewnętrznego
   b. Wywołaj operator pokrycia
3. Utwórz Zbiór akcji
4. Uaktywnij akcję w otoczeniu za pośrednictwem Efektorów

Rys. 19. Pojedynczy cykl pracy GCS
Fig. 19. Single learning cycle of GCS

W pierwszym kroku następnego cyklu Detektory D odbierają aktualny stan środowiska, czyli zgodnie z algorytmem działania algorytmu CYK wszystkie możliwe prawe strony produkcji, które wyprowadziłyby parsowany podciąg zdania (dla przykładu z rys. 18 kolejną wypełnianą komórką po [2, 3] jest komórka [1, 4], a zatem analizowane jest już pełne zdanie, gdyż ciąg $r_{41} = r$).

4.6. Pokrycie

Pokrycie (*covering*) jest jednym z dwóch (obok algorytmu genetycznego) mechanizmów odpowiedzialnych za odkrywanie nowych klasyfikatorów. W przeciwieństwie jednak do algorytmu genetycznego, którego działanie nie zawsze przynosi natychmiast oczekiwany efekt, pokrycie stosuje specjalizowane operatory, zadaniem których jest uzupełnienie populacji klasyfikatorów o produkcje umożliwiające w bieżącej sytuacji środowiskowej dalszą analizę zdania.

W modelu GCS zaprojektowano następujące metody pokrycia:

- Tworzenie klasyfikatorów przepisujących symbole terminalne. Tworzona jest produkcja typu $A \rightarrow a$ w sytuacji, gdy podczas rozbioru model napotyka nieznany wcześniej symbol terminalny (operator pokrycia terminalnego).
- Tworzenie klasyfikatora pełniącego funkcję symbolu uniwersalnego (*don't care*). Dla każdego symbolu terminalnego x tworzona jest dodatkowa produkcja typu $S_u \rightarrow x$, gdzie $x \in T$; przykładowo, jeżeli są dwa symbole terminalne a i b, to poza produkcjami terminalnymi $A \rightarrow a$ i $B \rightarrow b$ pojawią się dodatkowe dwie produkcje $S_u \rightarrow a$

[95] $S \rightarrow AB \rightarrow aB \rightarrow aBB \rightarrow abb$ lub $C \rightarrow SB \rightarrow Sb \rightarrow ABb \rightarrow abb$.

oraz $S_u \rightarrow b$, gdzie symbol S_u jest symbolem uniwersalnym (operator pokrycia uniwersalnego).

- Tworzenie symbolu startowego dla zdań o długości 1. Dla zdań pozytywnych $r \in R^+$ o długości 1 tworzona jest produkcja $S \rightarrow a$ (operator pokrycia startowego).
- Pełne pokrycie. Dla zdań pozytywnych tworzona jest produkcja typu $S \rightarrow AB$ w sytuacji, gdy w populacji istnieją już wyprowadzenia symboli A i B oraz analizowana jest komórka $[1, n]$ tablicy CYK (operator pokrycia pełnego).
- Pokrycie agresywne. Dla zdań pozytywnych tworzona jest produkcja typu $C \rightarrow AB$ w sytuacji, gdy w populacji istnieją już wyprowadzenia symboli A i B. Pokrycie agresywne uruchamiane jest z prawdopodobieństwem p_a (operator pokrycia agresywnego).

Algorytmiczną definicję operatorów pokrycia podaje rozdział następny.

4.7. Ścisk

Operatory pokrycia terminalnego i uniwersalnego dodają nową produkcję do populacji, zwiększając jednocześnie rozmiar populacji, operatory pokrycia startowego, pełnego i agresywnego przy dodawaniu produkcji stosują znany w obliczeniach ewolucyjnych schemat ze ściskiem (*crowding model*). Schemat ten, zaproponowany przez De Jonga (1975), a wzorowany na pracach dotyczących mechanizmu preselekcji Cavicchia (1970), wprowadza nowego osobnika do populacji w miejsce najbardziej podobnego. Schemat ze ściskiem zachowuje liczbę osobników w populacji. Podobieństwo osobnika wprowadzanego do osobnika zastępowanego mierzy się minimalną liczbą różnic na poszczególnych pozycjach chromosomu. W przypadku modelu GCS osobnikiem genetycznie modyfikowanym jest nieterminalna produkcja gramatyki bezkontekstowej w PNC, czyli postaci $A \rightarrow BC$, a zatem dwie produkcje mogą być maksymalnie różne na 3 pozycjach (dwie produkcje różne), minimalnie na żadnej (produkcje takie same po stronie warunku i akcji). Odległość mierząca podobieństwo pomiędzy produkcjami jest liczbą całkowitą z przedziału $[0, 3]$. W oryginalnym schemacie ze ściskiem osobnika do wymiany wybiera się spośród podzbioru *cf* (*crowding factor*, współczynnik ścisku) osobników wylosowanych z populacji[96]. W modelu GCS zastosowano mechanizm ścisku, w którym zastępuje się możliwie najsłabszego osobnika z populacji (Goldberg 1989). Z populacji wybieramy losowo podpopulację o wielkości *cs* (*crowding subpopulation*, podpopulacja ścisku) i zachowujemy z niej najsłabszego osobnika. Postępujemy tak *cf* razy. Otrzymujemy w ten sposób populację o rozmiarze *cf* osobników o niskim przystosowaniu, z których wybieramy do usunięcia najbardziej podobnego do nowej produkcji.

[96] Arabas pisze, że „prawdopodobieństwo zaliczenia osobnika do podpopulacji kandydatów do usunięcia jest wprost proporcjonalne do przystosowania osobników", (Arabas 2001, s. 127).

Warto zauważyć, że mechanizm ścisku jest, oprócz opisanej wcześniej płodności, kolejnym narzędziem, którego zadanie to zapobieganie negatywnym skutkom wysokiej epistazy populacji produkcji modelu GCS. Efektem zastosowania schematu ze ściskiem jest bowiem zachowanie różnorodności w populacji[97] i co może istotniejsze dla modelu GCS – ochrona pozycji produkcji w drzewie rozbioru przez zastępowanie klasyfikatora najbardziej podobnym.

4.8. Algorytm genetyczny

Model GCS, podobnie jak większość systemów klasyfikujących, korzysta z dwóch mechanizmów, które pozwalają na odkrywanie nowych reguł. Pierwszym z nich jest omówiony już wcześniej mechanizm pokrycia, drugim algorytm genetyczny.

4.8.1. Schemat działania

Algorytm genetyczny:

1. Dokonaj selekcji dwóch klasyfikatorów z populacji
2. Stwórz kopie wyselekcjonowanych klasyfikatorów
3.
   a. Zastosuj krzyżowanie na kopiach klasyfikatorów
   b. Zastosuj mutację na kopiach klasyfikatorów
   c. Zastosuj inwersję na kopiach klasyfikatorów
4. Dodaj ze ściskiem kopie klasyfikatorów do populacji, gwarantując przeżycie n_{elit} najlepszych osobników

Rys. 20. Schematyczny przebieg algorytmu genetycznego
Fig. 20. Schematic run of genetic algorithm

Algorytm genetyczny działa na populacji klasyfikatorów podobnie jak w innych systemach klasyfikujących. Istotną różnicą, wynikającą z odmiennej reprezentacji klasyfikatorów, jest operowanie jedynie na produkcjach ze zbioru P^N, czyli typu $A \rightarrow BC$. Na rysunku 20 przedstawiono schematyczny przebieg algorytmu genetycznego w modelu GCS. W pierwszym kroku następuje selekcja dwóch klasyfikatorów z populacji. W modelu GCS w zależności od ustawień konfiguracyjnych stosować można selekcję ruletkową, turniejową lub losową. W kroku drugim tworzone są kopie wskazanych

[97] Schemat ze ściskiem został zainspirowany zjawiskiem ekologicznym, w którym osobniki w naturalnej populacji, często na tym samym terenie, rywalizują ze sobą o ograniczone zasoby. Zróżnicowanie osobników prowadzi z czasem do zajęcia różnych nisz ekologicznych tak, że w zasadzie nie zachodzi rywalizacja. Ostatecznym rezultatem jest to, że w zrównoważonej populacji o stałej wielkości nowe osobniki z poszczególnych nisz zastępują podobne do nich osobniki starsze.

w kroku poprzednim klasyfikatorów[98], na których to kopiach stosuje się operatory genetyczne: krzyżowanie, mutacja i inwersja. Na koniec działania algorytmu genetycznego zmodyfikowane genetycznie kopie klasyfikatorów są dodawane do populacji metodą ścisku. Podczas dodawania osobników do populacji można zastosować sukcesję elitarną, zadaniem której jest zachowanie n_{elit} najlepszych produkcji w gramatyce.

4.8.2. Metody selekcji

Selekcja ruletkowa

W selekcji ruletkowej[99] dla każdego klasyfikatora liczone jest prawdopodobieństwo jego wyboru jako stosunek jego wartości przystosowania *f* do sumy wartości wszystkich klasyfikatorów w populacji. Następnie budowana jest ruletka, w której każdemu klasyfikatorowi przypisuje się wycinek koła o wielkości proporcjonalnej do prawdopodobieństwa jego wyboru. Selekcja klasyfikatora następuje po losowym obrocie koła ruletki.

Selekcja turniejowa

W selekcji turniejowej wybierana jest losowo podpopulacja klasyfikatorów o liczności *ts* (*tournament subpopulation*). Z wybranej podpopulacji wybierany jest klasyfikator o najwyższym przystosowaniu *f*.

Selekcja losowa

Selekcja losowa[100] wybiera z populacji losowo wskazany klasyfikator zgodnie z rozkładem normalnym.

Sukcesja elitarna

Zadaniem sukcesji[101] elitarnej jest zachowanie podpopulacji n_{elit} najlepszych produkcji w gramatyce w następnym kroku ewolucyjnym. W tym celu klasyfikatory z populacji rodzicielskiej są sortowane zgodnie z wartością funkcji przystosowania *f*. Pierwszych n_{elit} klasyfikatorów nie będzie podlegało mechanizmowi ścisku, czyli nie znajdzie się w podpopulacji o rozmiarze *cf*, z której wybierany jest osobnik do usunięcia.

---

[98] Pomysł genetycznej modyfikacji kopii wyselekcjonowanych osobników został zainspirowany jedną z implementacji systemu XCS (Butz i Wilson 2000).

[99] Ten typ selekcji nazywany jest selekcją z wykorzystaniem koła ruletki, selekcją ruletki, selekcją ruletkową, reprodukcją stochastyczną z powtórzeniami czy też reprodukcją proporcjonalną do przystosowania.

[100] Selekcja losowa, mimo że nie zapewnia odpowiedniego nacisku selektywnego (naporu selekcyjnego), jest stosowana w przetwarzaniu ewolucyjnym. Przykładem wykorzystania tego typu selekcji jest chociażby algorytm genetyczny ze zmienną liczebnością populacji (Michalewicz 1996), programowanie genetyczne stosowane w zadaniu poszukiwania algorytmów sortujących (Kinnear 1993), algorytm genetyczny rozwiązujący zadania komiwojażera (Freisleben i Merz 1996) czy też algorytm genetyczny ewoluujący zbiór klasyfikatorów o rzeczywistoliczbowych atrybutach (Corcoran i Sen 1994).

[101] Sukcesja polega na tworzeniu nowej populacji na podstawie osobników z populacji rodzicielskiej i potomnej (Arabas 2001).

4.8.3. Operatory genetyczne

Krzyżowanie

Operatorom genetycznym, w tym również krzyżowaniu, podlegają tylko produkcje typu $A \rightarrow BC$ (ze zbioru P^N). W modelu GCS krzyżowane dwie produkcje wymieniają swoje lewe strony oraz symbole z prawej strony produkcji na losowo wybranej pozycji. Przykładowo, wynikiem działania operatora na dwóch produkcjach:

$$A \rightarrow BC,$$

$$D \rightarrow EF$$

może być para produkcji

$$D \rightarrow EC,$$

$$A \rightarrow BF$$

przy założeniu, że zamianie (krzyżowaniu) podlegały symbole z pierwszej pozycji prawej strony produkcji. Operator krzyżowania uruchamiany jest z prawdopodobieństwem p_k.

Mutacja

Operator mutacji przechodzi przez kolejne pozycje produkcji typu $A \rightarrow BC$ i z prawdopodobieństwem p_a może zmienić symbol z danej pozycji produkcji na losowo wybrany ze zbioru nieterminali N. W krańcowym zatem wypadku, efektem zadziałania operatora może być produkcja z losowo zmienionymi symbolami zarówno po lewej stronie, jak i prawej stronie produkcji. Przykładowo, mutacja na dwóch pozycjach poniższej produkcji, tj. na lewej stronie produkcji oraz drugiej pozycji prawej strony,

$$A \rightarrow BC$$

może dać w efekcie produkcję postaci

$$G \rightarrow BD.$$

Inwersja

W ogólnym przypadku operator inwersji permutuje kod chromosomu. W przypadku produkcji nieterminalnej modelu GCS permutacji podlegają dwa symbole prawej strony produkcji. Efektem inwersji przykładowej produkcji

$$A \rightarrow BC$$

będzie produkcja postaci

$$A \rightarrow CB.$$

Operator inwersji stosowany jest z prawdopodobieństwem p_i.

Inwersja osobnika jest w tej chwili rzadziej stosowanym operatorem genetycznym, ze względu na możliwość stosowania np. krzyżowania równomiernego, które nie jest

tak wrażliwe na permutację kodu, jak operatory krzyżowania jedno- i dwupunktowego (Arabas 2001). Ze względu jednak na epistatyczne właściwości populacji klasyfikatorów w modelu GCS operator inwersji wydaje się być pożyteczny[102]. Warto bowiem zauważyć, że inwersja symboli po prawej stronie produkcji równoznaczna jest z zamianą kolejności poddrzew wyprowadzeń każdego z symboli, co z kolei nie powoduje zaburzenia związków epistatycznych pomiędzy klasyfikatorami.

Inwersja jest, oprócz wprowadzonych wcześniej płodności i ścisku, trzecim już mechanizmem, którego zadaniem jest utrzymanie wysokiego tempa zbieżności lub jej lepszym końcowym stopniem, pomimo istnienia wysokiej epistazy populacji produkcji modelu GCS.

4.9. Korekcja

Wynikiem działania operatorów genetycznych na produkcjach gramatyki, jak również możliwej losowej inicjacji populacji produkcji, może być pojawienie się bezużytecznych symboli nieterminalnych. *Symbol nieterminalny jest bezużyteczny*, jeżeli nie jest produktywny lub jest nieosiągalny. Z *symbolu produktywnego* można wyprowadzić jakiś łańcuch końcowy, składający się wyłącznie z symboli terminalnych, natomiast *symbol osiągalny* można wyprowadzić z symbolu startowego S (Hopcroft i Ullman 1979). *Produkcja jest produktywna*, jeżeli symbole nieterminalne prawej strony produkcji są produktywne. W przypadku reprezentacji gramatyki w PNC oznacza to, że obydwa symbole nieterminalne składające się na prawą stronę produkcji muszą być produktywne. *Produkcja jest osiągalna*, jeżeli symbol nieterminalny prawej strony produkcji jest osiągalny.

Co prawda algorytm CYK radzi sobie z produkcjami bezużytecznymi, jednak niewątpliwie takie reguły niepotrzebnie powiększają gramatykę, a co za tym idzie w sposób istotny zwiększają czas rozbioru analizowanego zdania. Zasadnym zatem się wydaje wprowadzenie możliwości upraszczania (korekcji) gramatyki. *Mechanizm korekcji* w pierwszej kolejności usuwa wszystkie produkcje redundantne w gramatyce, a następnie poszukuje i usuwa produkcje nieproduktywne i nieosiągalne.

Ze względu na epistatyczne własności produkcji gramatyki korekcja gramatyki wykonywana jest na kopii gramatyki, którą następnie przeprowadza się rozbiór zdania. Po rozbiorze zdań uczących obliczone wartości parametrów klasyfikatorów $Par_{cl}(G)$ zostają z powrotem skojarzone z klasyfikatorami gramatyki niekorygowanej. W dalszym procesie indukcji gramatyki bierze udział już tylko gramatyka niekorygowana, gdyż wiele reguł w danej chwili bezużytecznych może okazać się dobrym materiałem genetycznym w kolejnym kroku ewolucyjnym.

[102] O związkach inwersji i epistazy pisze Goldberg (1989).

4.10. Parametry

Proces uczenia modelu GCS sterowany jest przez zestaw parametrów, które można, idąc za (Pawlak 1999), podzielić na dwie grupy:

- parametry nieciągłe – zmienne decyzyjne,
- parametry ciągłe.

Zmienne decyzyjne

Do zmiennych decyzyjnych modelu GCS zaliczyć można:

f_{GA} – zmienna zezwalająca na uruchomienie algorytmu genetycznego (tak/nie),

f_{GA}^1 – rodzaj zastosowanej selekcji podczas wyboru pierwszej produkcji (ruletka, turniej, losowa),

f_{GA}^2 – rodzaj zastosowanej selekcji podczas wyboru drugiej produkcji (ruletka, turniej, losowa),

f_{kor} – zmienna zezwalająca na uruchomienie korekcji gramatyki (tak/nie),

f_{cs} – zmienna zezwalająca na uruchomienie operatora pokrycia startowego (tak/nie),

f_{cp} – zmienna zezwalająca na uruchomienie operatora pokrycia pełnego (tak/nie),

f_{cu} – zmienna zezwalająca na uruchomienie operatora pokrycia uniwersalnego (tak/nie).

Parametry ciągłe

Zbiór parametrów ciągłych modelu tworzą:

p_k – prawdopodobieństwo krzyżowania dla algorytmu genetycznego ($p_k \in [0, 1]$),

p_m – prawdopodobieństwo mutacji dla algorytmu genetycznego ($p_m \in [0, 1]$),

p_i – prawdopodobieństwo inwersji dla algorytmu genetycznego ($p_i \in [0, 1]$),

p_a – prawdopodobieństwo zastosowania operatora pokrycia agresywnego ($p_a \in [0, 1]$),

cf – współczynnik ścisku ($cf \in [1, 30]$),

cs – podpopulacja ścisku ($cs \in [1, 30]$),

ba – kwota bazowa ($ba \in [0, 15]$),

raf – współczynnik zmniejszania kwoty bazowej ($raf \in [0, 15]$),

ts – rozmiar podpopulacji turniejowej ($ts \in [1, 30]$),

n_P – rozmiar populacji,

n_{start} – liczba początkowych produkcji nieterminalnych ze zbioru P^N ($n_{\text{start}} \in [1, 30]$),

n_N – liczba symboli nieterminalnych ($n_N \in [1, 30]$),

n_T – liczba symboli terminalnych,

n_{elit} – wielkość elity ($n_{\text{elit}} \in [1, 30]$),

n_{run} – liczba iteracji ($n_{\text{run}} = 10, 50$),

$n_{\max}$ – maksymalna liczba kroków ewolucyjnych ($n_{\max} \in [1, 50\,000]$),

w_p – waga rozbioru zdania poprawnego $r \in R^+$ ($w_p \in [1, 20]$),

w_n – waga rozbioru zdania niepoprawnego $r \in R^-$ ($w_n \in [1, 20]$),

w_c – waga funkcji klasycznej klasyfikatora ($w_c \in [1, 20]$),
w_f – waga funkcji płodności klasyfikatora ($w_f \in [1, 15]$),
f_0 – miara użyteczności klasyfikatora niebiorącego udziału w parsowaniu ($f_0 \in [0, 10]$).

Po nazwie parametru, w nawiasie, umieszczono zakres jego badanej zmienności. Rozmiar populacji jest n_P jest wyliczany przez model jako suma liczby stworzonych produkcji terminalnych oraz liczby początkowych produkcji nieterminalnych.

4.11. Algorytmiczny opis modelu

W rozdziale tym przedstawiono algorytmiczny zapis modelu GCS. Zapis podany jest metodą zstępującą, czyli w pierwszej kolejności opisana jest zasadnicza, ewolucyjna pętla modelu (**procedure GCS** – rys. 21), a następnie pozostałe, wywoływane przez nią procedury.

```
procedure GCS
 1. begin
 2.   wczytaj zestaw zdań uczących R
 3.   ustaw parametry modelu
 4.   inicjuj gramatykę G_i = {N, T, P^T, P^N, S, S_u}
 5.   while (not kryterium stopu) do
 6.   begin
 7.     indukuj gramatykę G_i
 8.     oblicz f_G dla G_i
 9.     if f_GA then uruchom algorytm genetyczny na gramatyce G_i
10.   end
11. end
```

Rys. 21. Główna pętla modelu GCS
Fig. 21. Main loop of GCS model

4.11.1. Główna pętla programu

Działanie modelu rozpoczyna wczytanie zestawu uczącego R (rys. 21 – krok 2) oraz ustawienie parametrów programu: zmiennych decyzyjnych oraz wartości parametrów ciągłych (krok 3). W komputerowej implementacji modelu (patrz podrozdz. 4.12) można zastosować domyślne wartości parametrów.

W kroku 4 głównej pętli modelu wykonywana jest inicjacja indukowanej gramatyki G_i. Zadaniem inicjacji jest przede wszystkim utworzenie zbioru n_{start} początkowych produkcji nieterminalnych. Produkcje te mogą zostać do modelu wprowadzone ręcznie przez użytkownika lub – co ma najczęściej miejsce – zostać losowo wygenerowa-

ne. Podczas losowego tworzenia początkowego zestawu P^N, model korzysta ze zbioru symboli nieterminalnych N, o rozmiarze kontrolowanym przez parametr n_N. W kroku początkowym zbiór produkcji terminalnych P^T o mocy n_P może zostać pusty, gdyż za jego prawidłowe wypełnienie odpowiedzialny jest operator pokrycia terminalnego i operator pokrycia uniwersalnego w przypadku zezwolenia na jego działanie przez zmienną decyzjną f_{cu}. Kroki 6–10 to główna pętla modelu, w której następuje ewoluowanie gramatyki. Proces indukowania gramatyki trwa do momentu wypełnienia kryterium stopu, którym może być osiągnięcie zadanej liczby kroków pętli ewolucyjnej lub uzyskanie przez funkcję dostosowania gramatyki f_G wartości 100%. W każdym kroku pętli następuje procedura indukcji gramatyki (krok 7), ocena jej przystosowania f_G (krok 8 – patrz wzór (38)) i wreszcie genetyczna modyfikacja jej produkcji (w przypadku ustawienia zmiennej f_{GA} zezwalającej na uruchomienie algorytmu genetycznego).

4.11.2. Indukcja gramatyki

Algorytm indukcji gramatyki (**procedure Indukuj gramatykę** – rys. 22) opisuje wieloetapowy proces oceny poszczególnych produkcji tworzących gramatykę. Podczas każdego etapu uczenia wykonywany jest rozbiór kolejnego zdania uczącego ze zbioru R (krok 8), a następnie uaktualniane są wartości zbioru parametrów skojarzonych z każdą produkcją (krok 9). Końcowa ocena każdej produkcji, czyli obliczenie funkcji jej przystosowania f (krok 12), następuje dopiero po rozbiorze wszystkich zdań (kroki 6–10).

Procedure Indukuj gramatykę (*G*)

1. **begin**
2. inicjuj parametry produkcji $Par_{cl}(G)$
3. $G^* \leftarrow G$
4. $Par_{cl}(G^*) \leftarrow Par_{cl}(G)$
5. **if** f_{kor} **then** korekcja gramatyki G^*
6. **for each** $r \in R$ **do**
7. **begin**
8. parsuj zdanie r gramatyką G^*
9. aktualizuj parametry produkcji $Par_{cl}(G^*)$
10. **end**
11. $Par_{cl}(G) \leftarrow Par_{cl}(G^*)$
12. **for each** $cl \in G$ **do** oblicz f
13. **end**

Rys. 22. Algorytm indukcji gramatyki G
Fig. 22. Algorithm for induction of grammar G

Parsowanie zdań ze zbioru uczącego może odbywać się na skorygowanej kopii gramatyki (krok 5). Po zakończonym rozbiorze zdań obliczone wartości parametrów klasyfikatorów $Par_{cl}(G)$ zostają przypisane klasyfikatorom gramatyki niekorygowanej (krok 11). W procesie ewolucji (**procedure GCS** – krok 9) bierze udział gramatyka bez korekcji, gdyż wiele produkcji w danym kroku indukcyjnym bezużytecznych, może okazać się dobrym materiałem genetycznym w kolejnym kroku ewolucyjnym.

4.11.3. Parsowanie zdania

Algorytm parsowania zdania $r \in R$ gramatyką G jest w istocie rozszerzoną wersją algorytmu CYK (patrz podrozdz. 1.5.7), dostosowaną do modelu GCS. Po utworzeniu tabeli CYK (**procedure Parsuj zdanie**, rys. 23 – krok 3), stanowiącej środowisko dla modelu GCS (rys. 18), wypełniany jest pierwszy wiersz tabeli CYK (kroki 4–16).

procedure Parsuj zdanie (*r*, *G*)

1. **begin**
2. $n \leftarrow |r = a_1 a_2 ... a_n|$, a_i jest i-tym słowem zdania r o długości n
3. inicjuj tablicę CYK[n, n] reprezentującą otoczenie
4. **for** $i := 1$ **to** n **do**
5. **begin**
6. inicjuj: zbiory M, A, D
7. $D \leftarrow a_i \in D$, a_i jest komunikatem zewnętrznym reprezentującym aktualny stan otoczenia
8. $M \leftarrow \{ X \in N \mid X \rightarrow a_i, a_i \in D \}$
9. **if** $|M| = 0$ **then**
10. **begin**
11. zastosuj operator pokrycia terminalnego dla a_i
12. **if** f_{cs} **and** $n = 1$ **and** $r \in R^+$ **then** zastosuj operator pokrycia startowego dla a_i
13. **end**
14. $A \leftarrow M$
15. CYK[i, 1] $\leftarrow A$, czyli uaktywnij akcje w otoczeniu
16. **end**
17. **for** $j := 2$ **to** n **do**
18. **for** $i := 1$ **to** $n - j + 1$ **do**
19. **begin**
20. inicjuj: zbiory M, A, D
21. $r_{ij} \leftarrow a_i a_{i+1} ... a_{i+j-1}$, podciąg zdania r o długości j, rozpoczynający się od i-tej pozycji
22. $D \leftarrow \{BC \mid B$ wyprowadza pierwszą część r_{ij}, C wyprowadza drugą część r_{ij}, $B, C \in N \}$, czyli utwórz komunikat zewnętrzny reprezentujący aktualny stan otoczenia
23. $M \leftarrow \{ X \in N \mid X \rightarrow BC, BC \in D \}$
24. **if** $|M| = 0$ **and** $r \in R^+$ **then**
25. **begin**

```
26.             if liczba losowa z [0,1) < p_a then zastosuj operator pokrycia agresywnego dla
                D
27.             if f_cp and i = 1 and j = n then zastosuj operator pokrycia pełnego dla D
28.         end
29.         A ← M
30.         CYK[i, j] ← A, czyli uaktywnij akcje w otoczeniu
31.     end
32. end
```

Rys. 23. Algorytm parsowania zdania *r* gramatyką *G*
Fig. 23. Algorithm for parsing sentence *r* with grammar *G*

W pierwszym kroku pętli inicjowane są zbiory *M*, *A* oraz *D* (krok 6). Następnie (krok 7) do zbioru *D* wpisywane jest przetwarzane aktualnie słowo ze zdania *r*. W kroku 8 zbiór *M* wypełniany jest symbolami nieterminalnymi, z których można wyprowadzić dane słowo. Jeżeli nie ma takich symboli, to uruchamiany jest operator pokrycia terminalnego (**procedure Operator pokrycia terminalnego**), którego zadaniem jest dodanie brakującej produkcji terminalnej do gramatyki (krok 11). Jeżeli ustawienie parametru f_{cs} na to zezwala oraz analizowane jest zdanie pozytywne składające się z jednego słowa, to uruchamiany jest operator pokrycia startowego (**procedure Operator pokrycia startowego**), który dodaje do gramatyki produkcję wyprowadzającą analizowane słowo bezpośrednio z symbolu startowego gramatyki *S* (krok 12). Efektem działania powyższych operatorów pokrycia jest wypełnienie zbioru *M*. W kroku 15 procedury parsowania zdania wykonywany jest zapis do odpowiedniej komórki pierwszego wiersza tablicy CYK zawartości tablicy *M*, czyli symboli nieterminalnych, dla których istnieją w gramatyce produkcje terminalne wyprowadzające dane słowo. Zapis ten odbywa się za pośrednictwem zbioru *A* (krok 14).

Wypełnienie pierwszego wiersza tablicy CYK oznacza znalezienie w zbiorze produkcji lub też stworzenie produkcji terminalnych wyprowadzających wszystkie słowa zdania *r*. Innymi słowy, ten fragment algorytmu tworzy liście drzewa rozbioru zdania. Druga część algorytmu analizuje już dłuższe fragmenty zdania, począwszy od składających się z dwóch słów ($j = 2$), poszukując dla nich produkcji w gramatyce, które je wyprowadzają. Zagnieżdżone pętle **for** (kroki 17 i 18) powodują przeglądanie trójkątnej tablicy CYK. Wypełnianie pojedynczej komórki tablicy [*i*, *j*] odpowiada poszukiwaniu symboli nieterminalnych, z których można wyprowadzić fragment zdania o długości *j* słów i zaczynający się *i*-tym słowem zdania. W pierwszym kroku analizy komórki tablicy inicjalizowane są zbiory *M*, *A*, *E* (krok 20). W kroku kolejnym (krok 22) do zbioru *D* wpisywane są takie dwójki symboli nieterminalnych *BC*, z których *B* wyprowadza pierwszą część zdania, a symbol *C* część drugą zdania. Symbole *B* oraz *C* są poszukiwane w tych komórkach tablicy CYK, które reprezentują odpowiednio pierwsze i drugie części wyprowadzanego podciągu zdania. W kroku 23 wypełniany jest zbiór *M* lewymi stronami produkcji, których prawe strony należą do zbioru *D*. Jeżeli zbiór *M* jest pusty oraz analizowane jest zdanie pozytywne (krok 24),

to z prawdopodobieństwem p_a wywoływany jest operator pokrycia agresywnego (**procedure Operator pokrycia agresywnego**). Zadaniem operatora agresywnego jest dodanie ze ściskiem (**procedure Dodaj ze ściskiem**) do gramatyki produkcji, której prawa strona należy do zbioru *D*. Jeżeli dodatkowo rozpatrywana jest ostatnia komórka tablicy CYK [1, *n*] oraz zezwala na to ustawienie zmiennej f_{cp}, to uruchamiany jest operator pokrycia pełnego (**procedure Operator pokrycia pełnego**), dodający ze ściskiem do populacji produkcję, której lewą stronę tworzy symbol startowy *S*, a prawa należy do zbioru *D*.

Warto zauważyć, że wypełniona tablica CYK zawiera wszystkie możliwe rozbiory analizowanego zdania.

4.11.4. Ścisk

Operatory pokrycia startowego, agresywnego i pełnego, ale również algorytm genetyczny (**procedure Algorytm genetyczny**) dodają nową produkcję do gramatyki, stosując procedurę ścisku. Parametrami procedury są: nowa produkcja *nowy* oraz populacja produkcji *P*, do której będzie dodawana nowa produkcja (rys. 24).

Z populacji *P* wybierana jest losowo podpopulacja o wielkości *cs* (krok 5), z której zapamiętujemy produkcję o najniższej wartości funkcji przystosowania *f* (krok 6). Czynności te powtarzamy *cf* razy (kroki 3–8), otrzymując w rezultacie populację *K* o rozmiarze *cf* osobników o niskim przystosowaniu. Z populacji *K* wybierana jest produkcja *Y* najbardziej podobna do produkcji nowej (krok 9) pod względem zgodności symboli występujących w produkcji. W ostatnim kroku algorytmu nowa produkcja zastępuje miejsce produkcji *Y* w populacji (krok 11).

procedure Dodaj ze ściskiem (*nowy, P*)

1. **begin**
2. $K \leftarrow \emptyset$
3. **for** $i := 1$ **to** cf **do**
4. **begin**
5. $W^{cs} \subseteq P \leftarrow$ losowa podpopulacja z P o wielkości cs
6. $X \leftarrow$ najsłabszy osobnik z W^{cs}
7. $K \leftarrow K \cup X$
8. **end**
9. $Y \in K \leftarrow$ najbardziej podobny osobnik do *nowy*
10. $P \leftarrow P \setminus Y$
11. $P \leftarrow P \cup \{nowy\}$
12. **end**

Rys. 24. Algorytm dodawania ze ściskiem produkcji *nowy* do populacji *P*
Fig. 24. Crowding algorithm adding production *nowy* into the population *P*

4.11.5. Operatory pokrycia

Operator pokrycia terminalnego

Operator pokrycia terminalnego uruchamiany jest w procedurze parsowania zdania (rys. 23) w sytuacji, gdy algorytm CYK nie może znaleźć produkcji wyprowadzającej symbol terminalny (słowo zdania). W takiej sytuacji operator tworzy produkcję terminalną wyprowadzającą dany symbol terminalny (rys. 25 – krok 2), a następnie dodaje do zbioru produkcji terminalnych gramatyki (krok 3). Liczność zbioru produkcji n_P zostaje zwiększona o jeden. W kroku 4 zostaje wpisany do zbioru M symbol nieterminalny stojący po lewej stronie nowej produkcji. Operator pokrycia terminalnego uruchamia operator pokrycia uniwersalnego, jeżeli pozwala na to zmienna f_{cu}.

procedure Operator pokrycia terminalnego (a)
1. **begin**
2. utwórz $X \in N \rightarrow a$
3. $P^T = P^T \cup \{X \rightarrow a\}$
4. $M = \{X\}$
5. **if** f_{cu} **then** zastosuj operator pokrycia uniwersalnego dla a
6. **end**

Rys. 25. Algorytm operatora pokrycia terminalnego dla terminala a
Fig. 25. Terminal covering algorithm for terminal a

Operator pokrycia uniwersalnego

Zadaniem operatora pokrycia uniwersalnego jest dodanie do gramatyki produkcji terminalnej, wyprowadzającej zadany parametrem symbol terminalny (słowo zdania) z wyróżnionego symbolu uniwersalnego S_u. Wraz z pojawieniem się nowej produkcji zwiększającej rozmiar populacji n_P do zbioru M dodawany jest symbol uniwersalny S_u (rys. 26 – krok 4).

procedure Operator pokrycia uniwersalnego (a)
1. **begin**
2. utwórz $S_u \rightarrow a$
3. $P^N = P^N \cup \{S_u \rightarrow a\}$
4. $M = M \cup \{S_u\}$
5. **end**

Rys. 26. Algorytm operatora pokrycia uniwersalnego dla terminala a
Fig. 26. Don't care covering algorithm for terminal a

Ponieważ operator pokrycia uniwersalnego wywoływany jest przez operator pokrycia terminalnego, każdej produkcji terminalnej typu $A \rightarrow a$ towarzyszyć będzie produkcja $S_u \rightarrow a$. Oznacza to, że z symbolu uniwersalnego można będzie wyprowadzić wszystkie symbole terminalne gramatyki.

Operator pokrycia startowego

Działanie operatora pokrycia startowego uzależnione jest od ustawienia zmiennej f_{cs}. Dodatkowym warunkiem uruchomienia tego operatora jest rozbiór zdania pozytywnego (patrz rys. 23 – krok 12). Algorytm operatora rozpoczyna się od utworzenia produkcji terminalnej wyprowadzającej dany symbol terminalny z symbolu startowego S (rys. 27 – krok 2). W kroku następnym dodawana jest nowa produkcja do populacji produkcji za pomocą ścisku (krok 3). W kroku ostatnim algorytmu do zbioru M dodawany jest symbol startowy gramatyki.

procedure Operator pokrycia startowego (a)

1. **begin**
2. utwórz $S \rightarrow a$
3. dodaj ze ściskiem $S \rightarrow a$ do P^T
4. $M = M \cup \{S\}$
5. **end**

Rys. 27. Algorytm operatora pokrycia startowego dla terminala a
Fig. 27. Starting covering algorithm for terminal a

Operator pokrycia agresywnego

Operator pokrycia agresywnego uruchamiany jest z prawdopodobieństwem p_a w procedurze parsowania zdania (patrz rys. 23 – krok 26). W pierwszej kolejności tworzona jest nowa produkcja nieterminalna, której prawa strona należy do zbioru D (rys. 28 – krok 2). W kroku 3 procedury nowa produkcja dodawana jest ze ściskiem do zbioru produkcji nieterminalnych. W ostatnim kroku procedury zbiór M powiększany jest o lewą stronę nowej produkcji.

procedure Operator pokrycia agresywnego (D)

1. **begin**
2. utwórz $X \rightarrow BC \mid X \in N, BC \in D$
3. dodaj ze ściskiem $X \rightarrow BC$ do P^N
4. $M = M \cup \{X\}$
5. **end**

Rys. 28. Algorytm operatora pokrycia agresywnego dla zawartości Detektorów D
Fig. 28. Aggressive covering algorithm for detectors D

Operator pokrycia pełnego

Zadaniem operatora pokrycia pełnego jest, podobnie jak i operatora agresywnego, umożliwienie dalszej procedury rozbioru zdania pozytywnego, która bez pojawienia się nowej produkcji w gramatyce nie będzie możliwa. W odróżnieniu od operatora agresywnego, operator pokrycia pełnego zostaje uruchamiany tylko wtedy, gdy analizowana jest ostatnia komórka tablicy CYK, czyli w gramatyce są wszystkie produkcje poza produkcją wiążącą symbol startowy gramatyki z pozostałymi symbolami nieterminalnymi gramatyki. Dodatkowym warunkiem zadziałania operatora jest zezwolenie zmiennej f_{cp}.

Operator pokrycia pełnego tworzy nową produkcję nieterminalną, której prawa strona należy do zbioru D, a strona lewa jest symbolem startowym gramatyki (rys. 29 – krok 2). Następnie nowo utworzona produkcja dodawana jest ze ściskiem do populacji produkcji nieterminalnych (krok 3) oraz do zbioru M dodawany jest symbol startowy S.

procedure Operator pokrycia pełnego (D)

1. **begin**
2. utwórz $S \rightarrow BC \mid BC \in D$
3. dodaj ze ściskiem $S \rightarrow BC$ *do* P^N
4. $M = M \cup \{S\}$
5. **end**

Rys. 29. Algorytm operatora pokrycia pełnego dla zawartości Detektorów D
Fig. 29. Full covering algorithm for detectors D

4.11.6. Algorytm genetyczny

Algorytm genetyczny jest uruchamiany na zakończenie każdego kroku ewolucyjnego pod warunkiem zezwolenia ze strony zmiennej f_{GA} (patrz rys. 21 – krok 9). Celem działania algorytmu genetycznego na populacji produkcji jest ewolucyjne poszukiwanie gramatyki o lepszych własnościach niż wyindukowana w poprzednim kroku ewolucyjnym.

procedure Algorytm genetyczny (G)

1. **begin**
2. $P_1 \in P^N \leftarrow$ dokonaj selekcji produkcji z P^N
3. $P_2 \in P^N \leftarrow$ dokonaj selekcji produkcji z P^N
4. $P_1^* \leftarrow P_1$
5. $P_2^* \leftarrow P_2$
6. $P_{n_{\text{elit}}}^{\max} \in P^N \leftarrow$ wskaż n_{elit} najlepszych produkcji z P^N
7. **if** liczba losowa z $[0, 1) < p_k$ **then** zastosuj krzyżowanie na P_1^* i P_2^*
8. zastosuj mutację na P_1^*
9. zastosuj mutację na P_2^*
10. **if** liczba losowa z $[0, 1) < p_i$ **then** zastosuj inwersję na P_1^*
11. **if** liczba losowa z $[0, 1) < p_i$ **then** zastosuj inwersję na P_2^*
12. $P^N \leftarrow P^N \backslash P_{n_{\text{elit}}}^{\max}$
13. dodaj ze ściskiem P_1^* do P^N
14. dodaj ze ściskiem P_2^* do P^N
15. $P^N \leftarrow P^N \cup P_{n_{\text{elit}}}^{\max}$
16. **end**

Rys. 30. Algorytm genetyczny przetwarzający gramatykę G
Fig. 30. Genetic algorithm for grammar G

Pierwszym etapem algorytmu jest selekcja dwóch produkcji z populacji. Selekcja może się odbyć metodą ruletki, turniejową lub losową w zależności od ustawień zmiennych decyzyjnych f_{GA}^{1} oraz f_{GA}^{2} (rys. 30 – kroki 2–3). Wybrane produkcje są następnie kopiowane (krok 4–5). W kolejnym kroku wskazywanych jest n_{elit} najlepszych produkcji z populacji ze względu na wartość przystosowania. W kroku 7 algorytmu obie kopie wybranych produkcji są krzyżowane (**procedure Krzyżowanie**) z prawdopodobieństwem p_k. Następnie na każdej produkcji uzyskanej w wyniku krzyżowania działa operator mutacji (**procedure Mutacja**). Ostatnim operatorem przetwarzającym kopie wybranych produkcji jest inwersja (**procedure Inwersja**), która działa niezależnie na każdej produkcji z prawdopodobienstwem p_i (kroki 10–11). W krokach 12–15 algorytmu następuje dodanie przetworzonych ewolucyjnie produkcji do populacji z zachowaniem n_{elit} najlepszych produkcji.

Inwersja

Operator inwersji (rys. 31) uruchamiany jest w algorytmie genetycznym z prawdopodobieństwem p_i na każdej produkcji oddzielnie (rys. 30 – krok 10 i krok 11). Efektem działania operatora jest zamiana kolejności symboli po prawej stronie produkcji.

procedure Inwersja (*p*)
1. **begin**
2. utwórz $p^* := A \rightarrow BC \mid A, B, C \in N$
3. $p^* \leftarrow p$
4. $p^* \leftarrow (A \rightarrow CB)$
5. $p \leftarrow p^*$
6. **end**

Rys. 31. Algorytm operatora inwersji dla produkcji p
Fig. 31. Inversion algorithm for production p

Mutacja

Podobnie jak inwersja, mutacja działa niezależnie dla każdej genetycznie modyfikowanej produkcji (rys. 30 – krok 8 i krok 9). Z prawdopodobieństwem p_m może zostać zmieniona lewa strona produkcji (rys. 32 – krok 4), następnie pierwszy symbol prawej strony produkcji (krok 5) i wreszcie drugi symbol prawej strony produkcji (krok 6).

procedure Mutacja (*p*)
1. **begin**
2. utwórz $p^* := A \rightarrow BC \mid A, B, C \in N$
3. $p^* \leftarrow p$
4. **if** liczba losowa z $[0, 1) < p_m$ **then** $p^* \leftarrow (X \rightarrow BC) \mid X \in N$
5. **if** liczba losowa z $[0, 1) < p_m$ **then** $p^* \leftarrow (A \rightarrow XC) \mid X \in N$
6. **if** liczba losowa z $[0, 1) < p_m$ **then** $p^* \leftarrow (A \rightarrow BX) \mid X \in N$
7. $p \leftarrow p^*$
8. **end**

Rys. 32. Algorytm operatora mutacji dla produkcji p
Fig. 32. Mutation algorithm for production p

Krzyżowanie

Operator genetyczny krzyżowania dwóch produkcji uruchamiany jest przez procedurę algorytmu genetycznego z prawdopodobieństwem p_k (rys. 30 – krok 7). Krzyżowanie produkcji polega na zamianie miejscami bądź to symbolu pierwszego prawej strony produkcji (rys. 33 – kroki 7–10), bądź symbolu drugiego (kroki 12–15).

procedure Krzyżowanie (p_1, p_2)

1. **begin**
2. utwórz $p_1^* := A \rightarrow BC \mid A, B, C \in N$
3. utwórz $p_2^* := D \rightarrow EF \mid D, E, F \in N$
4. $p_1^* \leftarrow p_1$
5. $p_2^* \leftarrow p_2$
6. **if** liczba losowa z $\{1, 2\} = 1$ **then**
7. **begin**
8. $p_1^* \leftarrow (A \rightarrow EC)$
9. $p_2^* \leftarrow (D \rightarrow BF)$
10. **end**
11. **else**
12. **begin**
13. $p_1^* \leftarrow (A \rightarrow BF)$
14. $p_2^* \leftarrow (D \rightarrow EC)$
15. **end**
16. $p_1^* \leftarrow (D \rightarrow BC)$
17. $p_2^* \leftarrow (A \rightarrow EF)$
18. $p_1 \leftarrow p_1^*$
19. $p_2 \leftarrow p_2^*$
20. **end**

Rys. 33. Algorytm operatora krzyżowania dla produkcji p_1 i p_2
Fig. 33. Crossover algorithm for productions p_1 and p_2

Po zamianie któregoś z symboli prawej strony produkcji następuje zamiana miejscami symboli znajdujących się po lewej stronie krzyżowanych produkcji (kroki 16–17).

4.11.7. Korekcja

Algorytm korekcji gramatyki składa się z trzech kroków: usunięcia produkcji redundantnych z gramatyki (rys. 34 – krok 2), usunięcia produkcji nieproduktywnych (krok 3) oraz usunięcia produkcji nieosiągalnych (krok 4). Korekcja jest uwarunkowana wartością zmiennej f_{kor} i jest wywoływana przez procedurę indukcji gramatyki (rys. 22).

procedure Korekcja (*G*)

1. **begin**
2. usuń produkcje redundantne z G
3. usuń produkcje nieproduktywne z G
4. usuń produkcje nieosiągalne z G
5. **end**

Rys. 34. Algorytm korekcji gramatyki G
Fig. 34. Correction algorithm for grammar G

Zadanie znalezienia i usunięcia produkcji redundantnych z gramatyki jest stosunkowo proste i polega na pełnym przeglądzie listy produkcji i eliminacji wszystkich produkcji, które na tej liście występują więcej niż jeden raz.

procedure Usuń produkcje nieproduktywne (*G*)

1. **begin**
2. $N^* \leftarrow \{X \mid X \rightarrow a \in P^T,\ a \in T\}$
3. **repeat**
4. **for each** $X \rightarrow BC \in P^N$ **do**
5. **if** $\{B, C\} \subseteq N^*$ **then** $N^* \leftarrow N^* \cup \{X\}$
6. **until** N^* się nie zmienia
7. $P^N \leftarrow \{X \rightarrow BC \in P^N \mid B, C \in N^*\}$
8. **end**

Rys. 35. Algorytm procedury usuwania produkcji nieproduktywnych z gramatyki G
Fig. 35. Algorithm for removing non-productive productions from grammar G

Drugim krokiem korekcji jest usunięcie z populacji wszystkich produkcji nieproduktywnych, czyli takich, z których nie można wyprowadzić symboli terminalnych. W pierwszym kroku procedury do tymczasowego zbioru symboli nieterminalnych N^* przepisywane są wszystkie te symbole, z których wyprowadza się symbole końcowe gramatyki (rys. 35 – krok 2). Następnie w pętli (krok 3–6) wyszukuje się wszystkie symbole nieterminalne produktywne, poczynając od symboli wyprowadzających symbole terminalne (krok 5). Pętla wykonywana jest tak długo, aż tymczasowy zbiór symboli nieterminalnych N^* nie ulega już zmianie (krok 6). W ostatnim kroku procedury tworzony jest na nowo zbiór produkcji nieterminalnych gramatyki P^N. Do tego zbioru przepisywane są tylko te produkcje, których prawe strony należą do zbioru tymczasowgo N^* (krok 7).

Ostatnim etapem korekcji gramatyki jest usunięcie produkcji nieosiągalnych, czyli takich, które można wywieść z istniejących w populacji produkcji (rys. 36). Algorytm zaczyna się od stworzenia tymczasowego zbioru symboli nieterminalnych N^*, do którego wstawiany jest symbol końcowy gramatyki S (krok 2). Tworzony jest również

tymczasowy zbiór produkcji P^* (krok 3). Następnie wykonywana jest pętla tak długo, aż operacje wykonywane w środku pętli nie będą już zmieniły zawartości zbioru N^* i P^* (krok 4–10). W środku pętli iteracyjnie przeszukiwana jest lista produkcji nieterminalnych gramatyki pod kątem produkcji, których prawa strona należy do zbioru N^* (krok 5). Dla każdej takiej produkcji do zbioru N^* dodawane są symbole z prawej strony produkcji (krok 7), a zbiór P^* powiększa się o znalezioną produkcję (krok 8). Po wyjściu z pętli zbiór N^* nadpisuje zbiór symboli nieterminalnych gramatyki N (krok 11), a zbiór P^* nadpisuje zbiór produkcji nieterminalnych gramatyki P^N (krok 12).

procedure Usuń produkcje nieosiągalne (G)

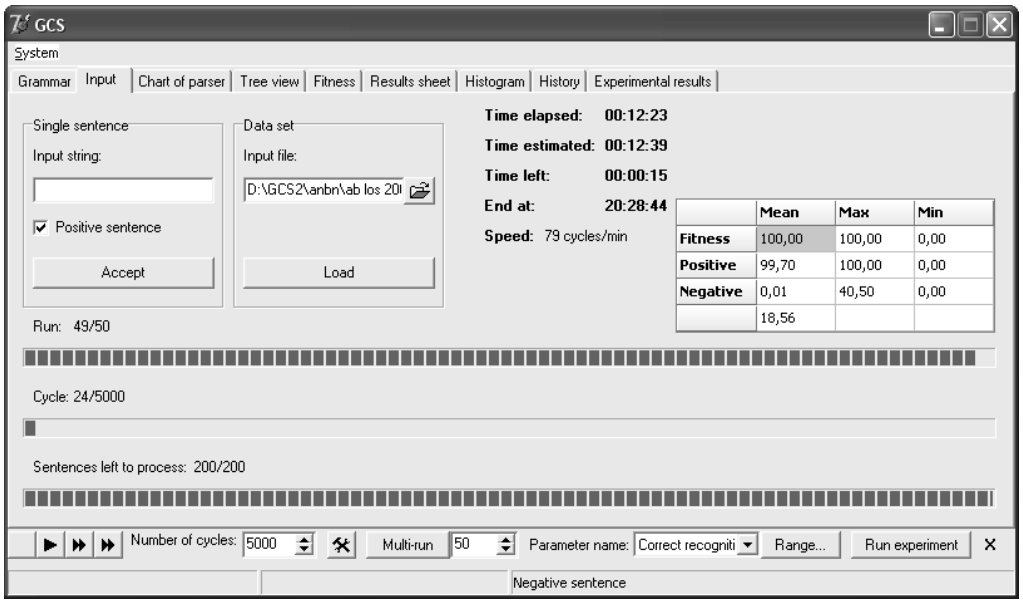
1. **begin**
2. $N^* \leftarrow \{S\}$
3. $P^* \leftarrow \emptyset$
4. **repeat**
5. **for each** $\{X \rightarrow BC \in P^N \mid X \in N^*\}$ **do**
6. **begin**
7. $N^* \leftarrow N^* \cup \{B, C\}$
8. $P^* \leftarrow P^* \cup \{ X \rightarrow BC\}$
9. **end**
10. **until** N^* i P^* się nie zmieniają
11. $N \leftarrow N^*$
12. $P^N \leftarrow P^*$
13. **end**

Rys. 36. Algorytm procedury usuwania produkcji nieosiągalnych z gramatyki G
Fig. 36. Algorithm for removing unreachable productions from grammar G

4.12. Opis programu gcs

Implementacja komputerowa modelu GCS została opracowana przez Cieleckiego w ramach pracy dyplomowej (Cielecki 2004). Program komputerowy **gcs** został przygotowany w środowisku programistycznym Borland Delphi 7 Personal dla systemu MS Windows.

Funkcjonalność programu została podzielona pomiędzy dziewięć zakładek: **Grammar**, **Input**, **Chart of parser**, **Tree view**, **Fitness**, **Results sheet**, **Histogram**, **History** oraz **Experimental results**. Dwie pierwsze zakładki Grammar i Input służą do definiowania zadania indukcji gramatyki, pozostałe zakładki prezentują wyniki uczenia.



Rys. 37. Zakładka Input programu gcs
Fig. 37. Input tab sheet of gcs application

Zakładka Input

Po uruchomieniu programu pojawia się zakładka Input (rys. 37), która pozwala na:

- wskazanie zbioru uczącego (obszar Data set),
- lub wpisanie analizowanego pojedynczego zdania (obszar Single sentence),
- określenie maksymalnej liczby kroków ewolucyjnych (Number of cycles, n_{max}),
- podanie liczby iteracji (Multi-run, n_{run}).

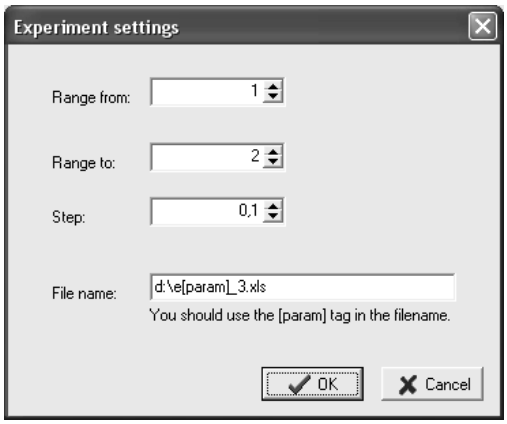
Zbiór uczący jest zapisywany w pliku tekstowym w tzw. formacie *abbadingo*[103] (Lang i in. 1998). W formacie tym zdania uczące umieszczane są w kolejnych liniach pliku. Symbole zdania rozdzielane są tzw. białym znakiem (spacją lub tabulatorem). Każde zdanie poprzedzone jest znacznikiem wskazującym, czy zdanie jest pozytywne (wartość 1) czy też negatywne (wartość 0). Po znaczniku występuje liczba określająca długość zdania. Pierwszy wiersz pliku zawiera liczbę zdań w zestawie oraz liczbę symboli terminalnych występujących w zbiorze uczącym. Na rysunku 38 przestawiono przykładową zawartość pliku (reprezentującą zbiór uczący dla tzw. języka Tomity (*ab*)*).

[103] Abbadingo to nazwa międzynarodowego konkursu z 1998 r. na najefektywniejszy program indukcji gramatycznej deterministycznych automatów skończonych (http://abbadingo.cs.unm.edu/).

```
15 2
1 2 a b
1 4 a b a b
1 6 a b a b a b
1 8 a b a b a b a b
1 14 a b a b a b a b a b a b a b
0 1 a
0 1 b
0 2 a a
0 2 b b
0 2 b a
0 3 a b a
0 3 a b b
0 7 a b b a b a b
0 5 a b a a b
0 9 a a b a b a b a b
```

Rys. 38. Przykładowy zbiór uczący programu gcs
Fig. 38. Exemplary learning set for gcs application

Program gcs umożliwia również pracę w trybie wsadowym (Run experiment), w którym automatycznie zmienia wartość analizowanego parametru o zadaną wartość. Uruchomienie trybu wsadowego poprzedza wybranie analizowanego parametru (Parametr name) oraz wskazanie zakresu zmiany parametru (Range from Range to), kroku zmiany (Step) i parametryzowanej nazwy pliku, do którego będą zapisywane wyniki indukcji dla pojedynczej wartości badanego parametru (rys. 39).

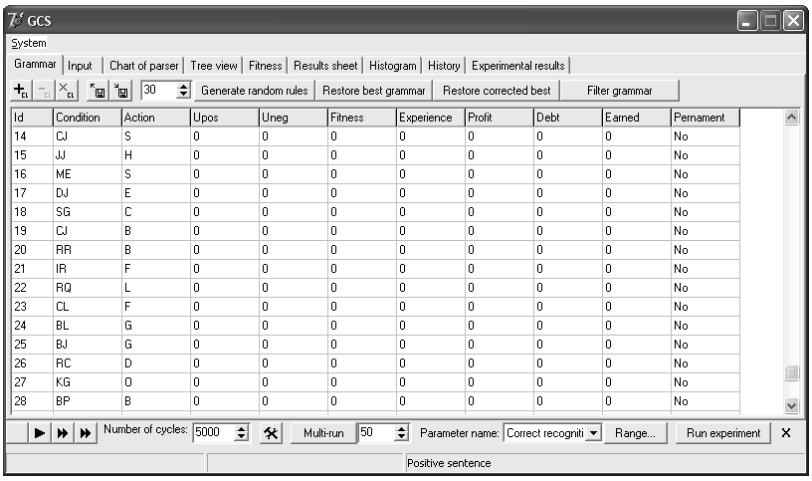


Rys. 39. Okno zmiany zakresu parametru programu gcs
Fig. 39. Parameter range setting window of gcs application

Na zakładce Input podawane są również orientacyjne czasy trwania eksperymentu (Time elapsed, Time estimated, Time left, End at), prędkość przetwarzania mierzona

w liczbie kroków ewolucyjnych na minutę (Speed), a także aktualnie wykonywany krok ewolucyjny danego przebiegu. Dodatkowo, zakładka na bieżąco pozwala śledzić średnie, maksymalne i minimalne (Mean, Max, Min) wartości dopasowania gramatyki, kompetencji pozytywnej oraz negatywnej (Fitness, Positive, Negative), uśrednione po wszystkich krokach wszystkich iteracji (przebiegów) eksperymentu. Pozytywna kompetencja gramatyki określa procent sparsowanych zdań poprawnych, a kompetencja negatywna wyraża procent sparsowanych zdań niepoprawnych.

Uczenie można prowadzić w trybie krokowym, analizując pojedyncze zdania z zestawu uczącego – wypełnienie tablicy CYK (▶) lub uruchamiając pojedynczy cykl pracy modelu – wypełnienie jednej komórki tablicy CYK (▶▶).



Rys. 40. Zakładka Grammar programu gcs
Fig. 40. Grammar tab sheet of gcs application

Zakładka Grammar

Zakładka Grammar służy do generacji populacji początkowej produkcji indukowanej gramatyki (rys. 40). Populację początkową gramatyki można wygenerować losowo, podając liczbę produkcji nieterminalnych oraz uruchamiając przycisk Generate random *rules*. Funkcja tworzy losowo n_{start} produkcji nieterminalnych typu $X \rightarrow BC$, których symbole należą do zbioru N o liczności n_N. Produkcje można również wpisywać ręcznie, korzystając z okna dodawania produkcji (rys. 41), wywoływanego przyciskiem **+**. Zarówno w przypadku generacji losowej początkowego zestawu produkcji, jak i ręcznego ich dodawania można zabezpieczyć daną produkcję przed jakąkolwiek modyfikacją w trakcie indukcji (własność Pernament classifier).

Aktualnie wyświetlaną listę produkcji można zapisać na dysk lub też z dysku odczytać wcześniej zapisany zestaw produkcji (przyciski ↖🖫 oraz ↘🖫).

Na listwie narzędziowej zakładki dostępne są jeszcze funkcje:

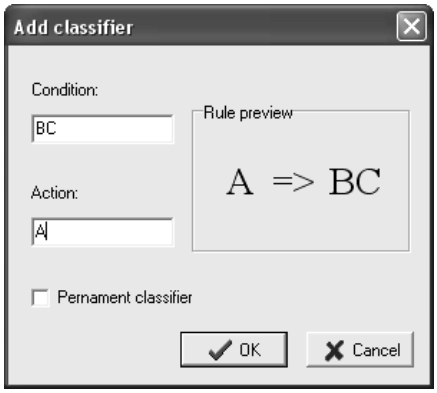
Restore best grammar – funkcja wyświetla najlepszą pod względem funkcji dostosowania gramatykę po zakończonym procesie indukcji,

Restore corrected best – funkcja wyświetla najlepszą pod względem funkcji dostosowania gramatykę po zakończonym procesie indukcji, dodatkowo uruchamiając na niej procedurę korekcji,

Filter grammar – funkcja usuwa z wyświetlanego zestawu produkcji wszystkie produkcje, które nie były użyte w procesie indukcji (ich parametry u_p i u_n wynoszą zero).

Wyświetlana na zakładce Grammar lista produkcji złożona jest z następujących kolumn:

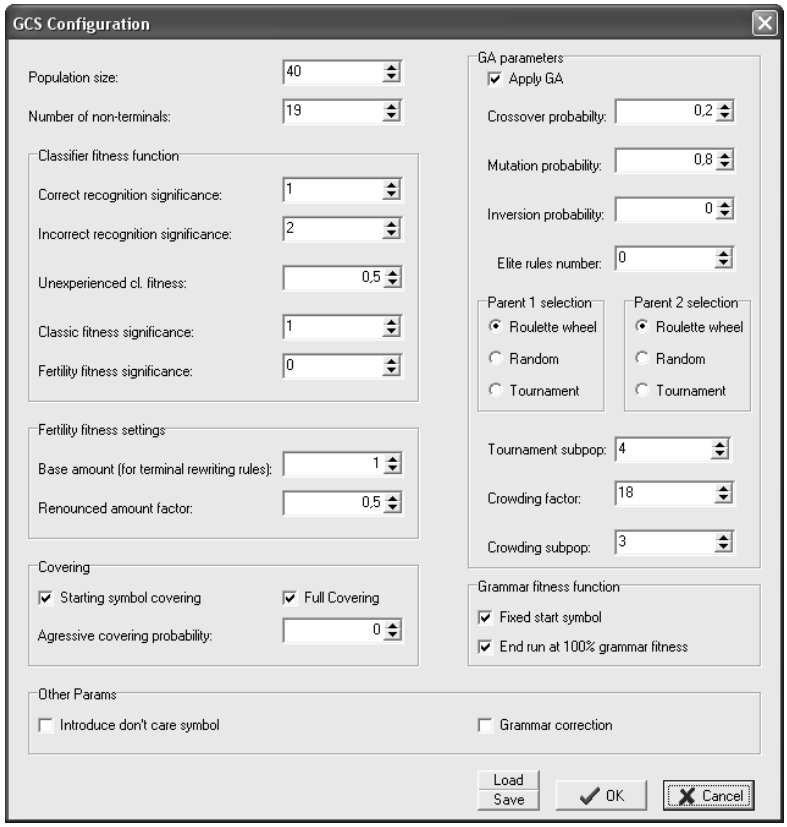
Id – identyfikator produkcji,
Condition – prawa strona produkcji (symbol terminalny lub dwójka symboli nieterminalnych),
Action – lewa strona produkcji (symbol nieterminalny),
Upos – liczba zastosowań produkcji przy parsowaniu zdań poprawnych ze zbioru uczącego (parametr u_p),
Uneg – liczba zastosowań produkcji przy parsowaniu zdań niepoprawnych ze zbioru uczącego (parametr u_n),
Fitness – wartości funkcji dopasowania produkcji f,
Experience – liczba zastosowań produkcji,
Profit – suma punktów zdobytych przez klasyfikator podczas parsowania zdań poprawnych (parametr p),
Debt – suma punktów zdobytych przez klasyfikator podczas parsowania zdań niepoprawnych (parametr d),
Earned – suma punktów zebranych za płodność produkcji (różnica $p - d$),
Pernament – wartość zmiennej logicznej określającej, czy dana produkcja jest niezmienna w trakcie uczenia.



Rys. 41. Okno dodawania produkcji programu gcs
Fig. 41. Add new production window of gcs application

Okno Parametrów

Poza określeniem zbioru uczącego i inicjalizacji populacji produkcji należy podać wartości zmiennych decyzyjnych oraz parametrów ciągłych modelu GCS. Okno parametrów programu (rys. 42) uruchamiane jest przyciskiem 🛠 z dolnej listy narzędziowej programu.



Rys. 42. Okno parametrów programu gcs
Fig. 42. Parameters configuration window of gcs application

W oknie parametrów definiuje się:

Population size – rozmiar populacji (n_P),

Number of non-terminals – liczba symboli nieterminalnych (n_N), w programie przyjęto, że ostatni symbol nieterminalny pełni funkcję symbolu startowego gramatyki, a przedostatni symbolu uniwersalnego, o ile jest zezwolenie na używanie tego symbolu podczas indukcji,

Correct recognition significance – waga rozbioru zdania poprawnego (w_p),

Incorrect recognition significance – waga rozbioru zdania niepoprawnego (w_n),

Unexperienced cl. fitness – miara użyteczności klasyfikatora niebiorącego udziału w parsowaniu (f_0),

Classic fitness significance – waga funkcji klasycznej klasyfikatora (w_c),

Fertility fitness significance – waga funkcji płodności klasyfikatora (w_p),

Base amount – kwota bazowa (ba),

Renounced amount factor – współczynnik zmniejszania kwoty bazowej (raf),

Start covering – zmienna zezwalająca na uruchomienie operatora pokrycia startowego (f_{cs}),

Full covering – zmienna zezwalająca na uruchomienie operatora pokrycia pełnego (f_{cp}),

Agressive covering – prawdopodobieństwo zastosowania operatora pokrycia agresywnego (p_a),

Introduce don't care symbol – zmienna zezwalająca na na uruchomienie operatora pokrycia uniwersalnego (f_{cu}),

Grammar correction – zmienna zezwalająca na uruchomienie korekcji gramatyki (f_{kor}),

Apply GA – zmienna zezwalająca na uruchomienie algorytmu genetycznego (f_{GA}),

Crossover probability – prawdopodobieństwo krzyżowania (p_k),

Mutation probability – prawdopodobieństwo mutacji (p_m),

Inversion probability – prawdopodobieństwo inwersji (p_i),

Elite rules number – wielkość elity (n_{elit}),

Parent 1 selection – rodzaj zastosowanej selekcji podczas wyboru pierwszej produkcji: ruletka, turniej, losowa (f_{GA}^1),

Parent 2 selection – rodzaj zastosowanej selekcji podczas wyboru drugiej produkcji: ruletka, turniej, losowa (f_{GA}^2),

Tournament subpop – rozmiar podpopulacji turniejowej (ts),

Crowding factor – współczynnik ścisku (cf),

Crowding subpop – podpopulacja ścisku (cs),

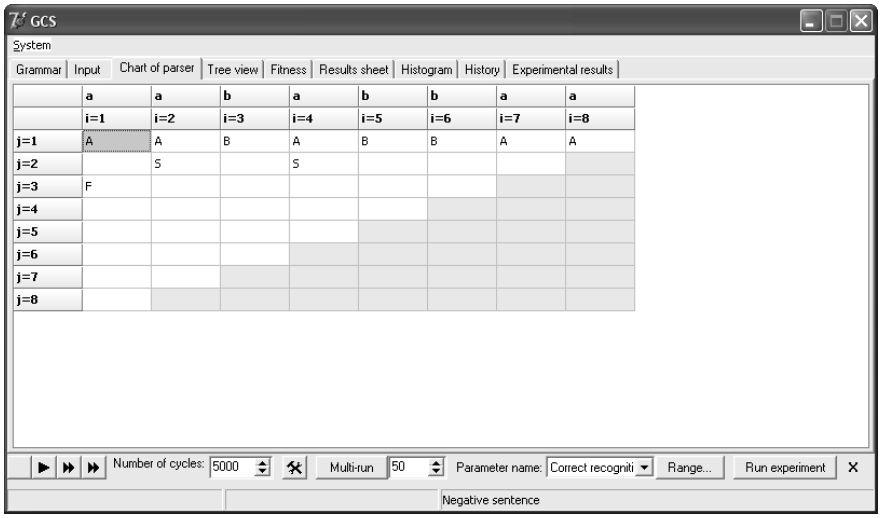
Fixed start symbol – jeżeli zmienna jest ustawiona na „tak", to rozbiór zdania uznaje się za osiągnięty – w ostatniej komórce tablicy CYK znajdzie się symbol startowy gramatyki, w przeciwnym wypadku rozbiór jest uznawany za osiągnięty, wówczas, gdy w ostatniej komórce tablicy CYK znajdzie się dowolny symbol nieterminalny,

End run at 100% grammar fitness – zmienna zezwalająca na zakończenie danego przebiegu eksperymentu w momencie osiągnięcia przez gramatykę 100% dostosowania.

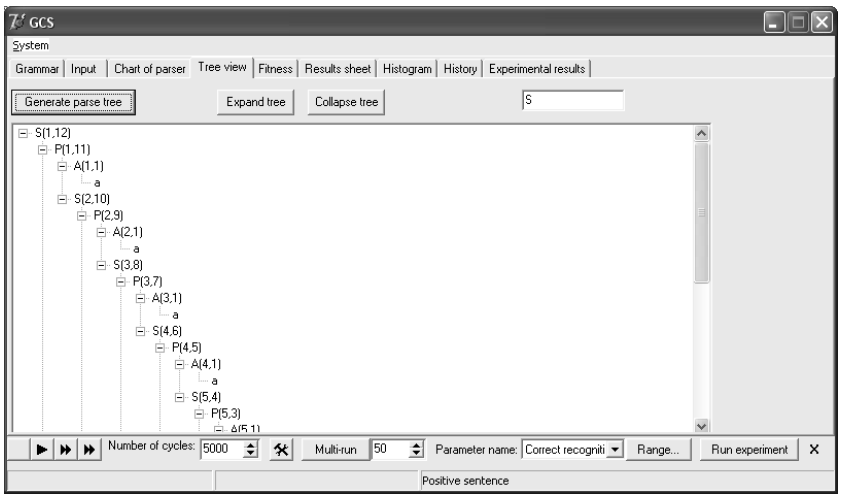
Parametry można zapisać do zbioru dyskowego (opcja Save) lub odczytać z wcześniej zapisanego zbioru (Load).

Zakładka Chart of parser

Zakładka Chart of parser prezentuje bieżącą zawartość tablicy CYK (rys. 43).



Rys. 43. Zakładka Chart of parser programu gcs
Fig. 43. Chart of parser tab sheet of gcs application



Rys. 44. Zakładka Tree view programu gcs
Fig. 44. Tree view tab sheet of gcs application

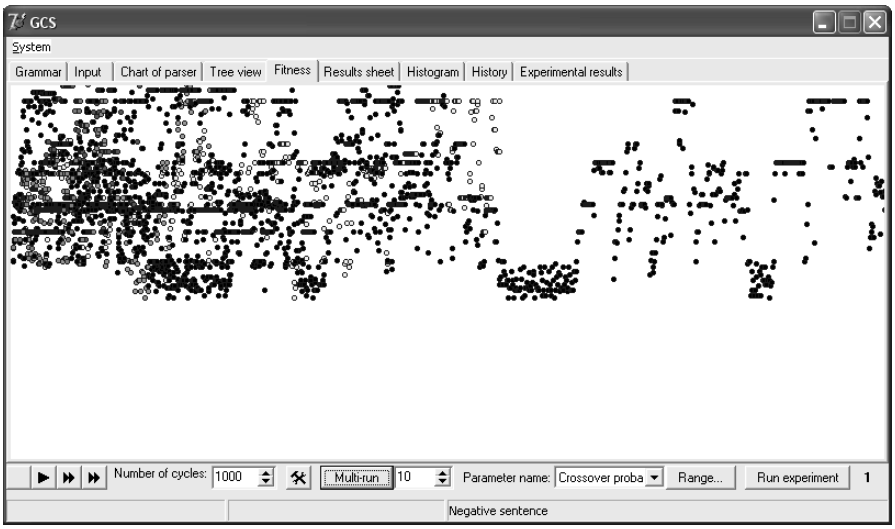
Zakładka Tree view

Zakładka Tree view prezentuje drzewo aktualnego rozbioru, a zatem przydatna jest w trybie krokowym pracy programu (rys. 44). Przed wyborem tej zakładki należy zakończyć rozbiór zdania. Po przejściu do zakładki Tree view należy wpisać do okna edycyjnego symbol nieterminalny, którego rozwinięcie chcemy analizować. Podanie

symbolu startowego gramatyki pozwala na obserwację całego drzewa rozbioru (pierwszego drzewa z możliwego lasu drzew rozbioru). Drzewo rozbioru generowane jest po wybraniu przycisku Generate parse tree. Przy każdym symbolu nieterminalnym pojawiają się w nawiasie dwie liczby oznaczające współrzędne tego symbolu w tablicy CYK, czyli pozycję, od której dany symbol rozwija zdanie oraz długość tego rozwinięcia. Przycisk Expand tree rozwija drzewo rozbioru od podanego symbolu, przycisk Collapse tree zwija drzewo do pojedynczego symbolu.

Zakładka Fitness

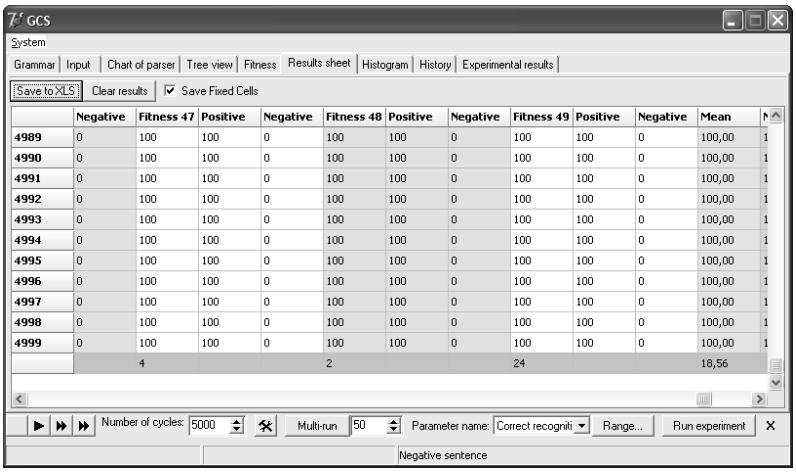
Wstępną, graficzną ocenę indukcji gramatyki umożliwia wykres punktowy dopasowania gramatyki w kolejnym kroku ewolucyjnym, który prezentowany jest na zakładce Fitness (rys. 45).



Rys. 45. Zakładka Fitness programu gcs
Fig. 45. Fitness tab sheet of gcs application

Zakładka Results sheet

Zakładka Results sheet prezentuje wyniki indukcji gramatyki (rys. 46). Kolejne wiersze tabeli zawierają wartość funkcji dopasowania gramatyki w kolejnych krokach ewolucyjnych uczenia. W poszczególnych kolumnach umieszczone są natomiast kolejne iteracje indukcji. Każda iteracja reprezentowana jest przez trzy kolumny: Fitness, Positive oraz Negative, które odpowiednio podają dopasowanie gramatyki, procent sparsowanych zdań pozytywnych, procent sparsowanych zdań negatywnych. Ostatnich dziewięć kolumn tabeli pokazuje wartości średnie uzyskane w danym kroku ewolucyjnym ze wszystkich przebiegów następujących parametrów: średnie Fitness, Positive, Negative; maksymalne Fitness, Positive, Negative i wreszcie minimalne Fitness, Positive, Negative. Tabelę wyników można zapisać do arkusza kalkulacyjnego (Save to XLS).

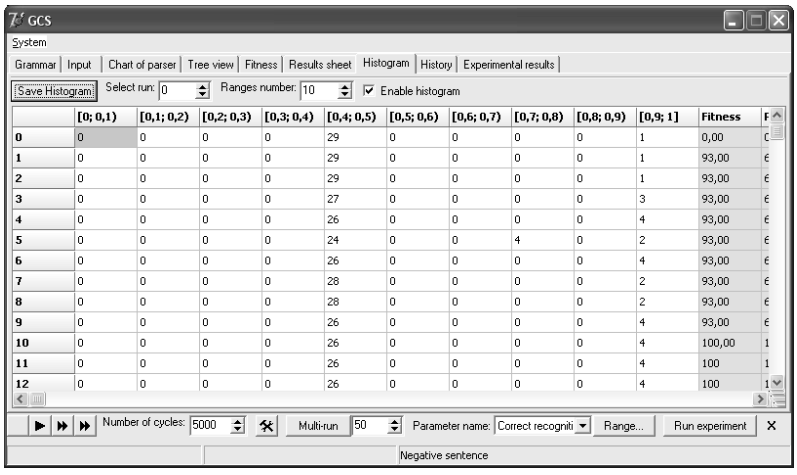


	Negative	Fitness 47	Positive	Negative	Fitness 48	Positive	Negative	Fitness 49	Positive	Negative	Mean
4989	0	100	100	0	100	100	0	100	100	0	100,00
4990	0	100	100	0	100	100	0	100	100	0	100,00
4991	0	100	100	0	100	100	0	100	100	0	100,00
4992	0	100	100	0	100	100	0	100	100	0	100,00
4993	0	100	100	0	100	100	0	100	100	0	100,00
4994	0	100	100	0	100	100	0	100	100	0	100,00
4995	0	100	100	0	100	100	0	100	100	0	100,00
4996	0	100	100	0	100	100	0	100	100	0	100,00
4997	0	100	100	0	100	100	0	100	100	0	100,00
4998	0	100	100	0	100	100	0	100	100	0	100,00
4999	0	100	100	0	100	100	0	100	100	0	100,00
		4			2			24			18,56

Rys. 46. Zakładka Results sheet programu gcs
Fig. 46. Results tab sheet of gcs application

Zakładka Histogram

Szczegółową analizę uzyskanych wyników umożliwia zakładka Histogram (rys. 47). Dla wskazanego przebiegu indukcji (Select run) oraz zakresu zmienności (Ranges number) zakładka pokazuje histogram wartości przystosowania produkcji w poszczególnych krokach ewolucyjnych. Każda komórka tabeli zawiera liczbę produkcji, które w danym kroku ewolucyjnym (wierszu) osiągnęły przystosowanie o wartości wskazanej w nagłówku kolumny. Ostatnie cztery kolumny tabeli zawierają wartości Fitness, Positive, Negative oraz populację produkcji dla danego kroku ewolucyjnego i przebiegu. Histogram można zapisać do arkusza kalkulacyjnego (Save Histogram).

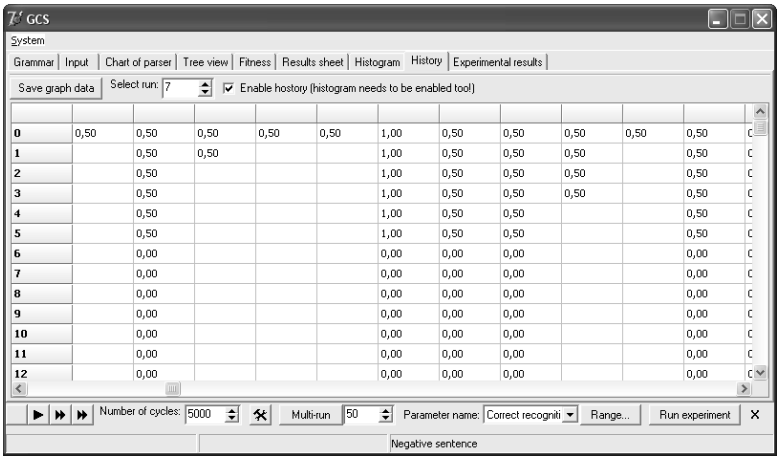


	[0; 0,1)	[0,1; 0,2)	[0,2; 0,3)	[0,3; 0,4)	[0,4; 0,5)	[0,5; 0,6)	[0,6; 0,7)	[0,7; 0,8)	[0,8; 0,9)	[0,9; 1]	Fitness
0	0	0	0	0	29	0	0	0	0	1	0,00
1	0	0	0	0	29	0	0	0	0	1	93,00
2	0	0	0	0	29	0	0	0	0	1	93,00
3	0	0	0	0	27	0	0	0	0	3	93,00
4	0	0	0	0	26	0	0	0	0	4	93,00
5	0	0	0	0	24	0	0	4	0	2	93,00
6	0	0	0	0	26	0	0	0	0	4	93,00
7	0	0	0	0	28	0	0	0	0	2	93,00
8	0	0	0	0	28	0	0	0	0	2	93,00
9	0	0	0	0	26	0	0	0	0	4	93,00
10	0	0	0	0	26	0	0	0	0	4	100,00
11	0	0	0	0	26	0	0	0	0	4	100
12	0	0	0	0	26	0	0	0	0	4	100

Rys. 47. Zakładka Histogram programu gcs
Fig. 47. Histogram tab sheet of gcs application

Zakładka History

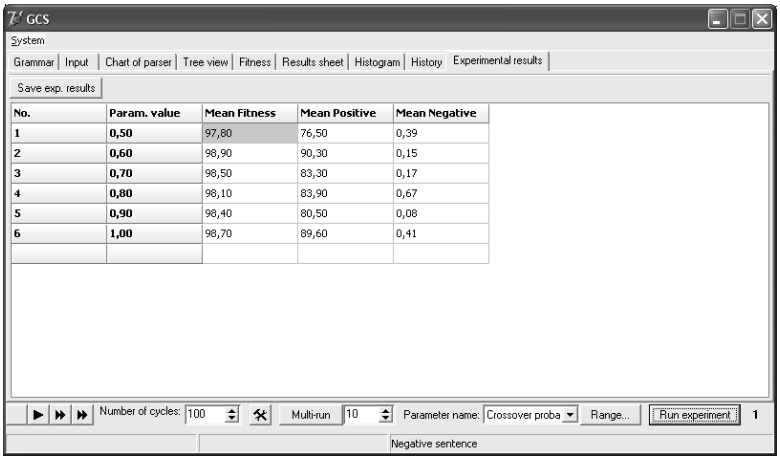
Zakładka History bardzo umożliwia analizę życia produkcji w trakcie procesu uczenia (rys. 48). Pole Select run pozwala na wybranie przebiegu indukcji. Każdy wiersz tablicy odpowiada jednemu krokowi ewolucyjnemu indukcji, każda produkcja ma w tablicy zarezerwowaną jedną kolumnę, w której pokazywana jest jej wartość przystosowania w danym kroku. Tablicę można zapisać do arkusza kalkulacyjnego (Save graph data).



Rys. 48. Zakładka History programu gcs
Fig. 48. History tab sheet of gcs application

Zakładka Experimental results

Ostatnia zakładka programu GCS prezentuje wybrane rezultaty uzyskane podczas pracy programu w trybie wsadowym (rys. 49). W kolejnych wierszach tablicy poka-



Rys. 49. Zakładka Experimental results programu gcs
Fig. 49. Experimental results tab sheet of gcs application

zywane są średnie wartości parametrów Fitness, Positive, Negative uzyskanych dla wskazanej w pierwszej kolumnie wartości badanego parametru. Dane można zapisać do arkusza kalkulacyjnego (Save exp. results).

5. Badania symulacyjne modelu GCS

Dziedzina algorytmów ewolucyjnych (AE), pomimo że sięgająca początku lat sześćdziesiątych ubiegłego wieku, wciąż dynamicznie rozwija się, szukając nowych obszarów zastosowań. Oznacza to między innymi, że koncentruje się bardziej na praktycznych, empirycznych aspektach algorytmów niż pogłębionych studiach teoretycznych. Obecnie stosowane w badaniach teoretycznych AE metody można zakwalifikować do jednej z następujących grup:

- teoria schematów (*schema theory*) (Holland 1992),
- teoria łańcuchów Markowa (*Markov chains theory*) (Rudolph 1998),
- analiza wymiarowa (*dimensional analysis*) (Thierens 1996),
- statystyki pozycyjne (*order statistics*) (Beyer 1995),
- genetyka klasyczna (*quantitative genetics*) (Mühlenbein i Schlierkamp-Voosen 1993),
- ortogonalna analiza funkcyjna (*orthogonal functions analysis*) (Bethke 1980),
- kwadratowe układy dynamiczne (*quadratical dynamical systems*, QDS) (Vose i Liepins 1991),
- fizyka statystyczna (*statistical physics*) (Asselmeyer i Ebeling 1997).

Jednak pomimo stosowania wyrafinowanych i złożonych modeli matematycznych nie uzyskano dotychczas rozwiązań, które nie miałyby wąskiego obszaru zastosowań (jak np. teoria schematów czy też łańcuchy Markowa) bądź nie byłyby przedmiotem ożywionych dyskusji i polemik. Wielokrotnie też przyjmowane w analizie teoretycznej założenia upraszczające na tyle zniekształcają analizowane algorytmy, że dużym znakiem zapytania należy opatrzyć rzeczywisty związek pomiędzy otrzymanymi wynikami a badanymi algorytmami.

Dominującą metodą oceny nowych modeli ewolucyjnych, zwłaszcza pod kątem porównywania z istniejącymi już rozwiązaniami, są zatem wciąż badania empiryczne (Kwaśnicka 1999). Również model GCS został poddany badaniom empirycznym, których wyniki, w zestawieniu z osiągniętymi w literaturze przedmiotu rezultatami, pozwolą na określenie ogólnych własności modelu. Symulacyjnie zostanie także zbadany wpływ wybranych parametrów modelu na indukcję gramatyk. Oceniane będą przede wszystkim dokładność uzyskiwanych rozwiązań oraz koszt symulacji (Arabas 2001).

5.1. Estymatory symulacji

5.1.1. Dokładność indukcji

W przypadku modelu GCS główną miarą oceny dokładności rozwiązania będzie średnia wartość funkcji dopasowania gramatyki f_G (patrz wzór (38)) w danym kroku ewolucyjnym, oznaczana w dalszej części pracy jako estymator *fitness*. Estymator *fitness* liczony jest jako średnia *j*-tego kroku ewolucyjnego po wszystkich iteracjach procesu uczenia gramatyki i normalizowany do 100%:

$$fitness(j) = \frac{\sum_{i=1}^{n_{\text{run}}} f_{G(i)}}{n_{\text{run}}} 100, \tag{39}$$

gdzie:

j – krok ewolucyjny, $j = 1, \ldots, n_{\max}$,

$n_{\max}$ – maksymalna liczba kroków ewolucyjnych,

n_{run} – liczba iteracji indukcji gramatyki,

$f_{G(i)}$ – wartość funkcji dopasowania gramatyki w *i*-tym kroku ewolucyjnym.

Estymator oblicza średni procent prawidłowo sparsowanych zdań ze zbioru uczącego, tj. sparsowanych zdań pozytywnych i niesparsowanych zdań negatywnych, i jest nazywany *kompetencją ogólną* gramatyki. Dodatkowymi miarami oceniającymi jakość indukowanej gramatyki są *kompetencja pozytywna* (*positive*) oraz *kompetencja negatywna* (*negative*). Estymator *positive* określa średni procent sparsowanych zdań poprawnych, a estymator *negative* średni procent sparsowanych zdań niepoprawnych, oba estymatory liczone w danym kroku ewolucyjnym ze wszystkich iteracji indukcji:

$$positive(j) = \frac{\sum_{i=1}^{n_{\text{run}}} (U_{p(i)} / |R^+|)}{n_{\text{run}}} 100, \tag{40}$$

$$negative(j) = \frac{\sum_{i=1}^{n_{run}} (U_{n(i)} / |R^-|)}{n_{\text{run}}} 100, \tag{41}$$

gdzie:

$U_{p(i)}$ – liczba sparsowanych zdań poprawnych w *i*-tej iteracji,

$U_{n(i)}$ – liczba sparsowanych zdań niepoprawnych w *i*-tej iteracji,

$|R^+|$ – liczba zdań pozytywnych w zestawie uczącym,

$|R^-|$ – liczba zdań negatywnych w zestawie uczącym.

Zakres zmienności kompetencji gramatyki w danym kroku ewolucyjnym indukcji obliczany jest przez następujące miary:

$$fitness(j)_{\max} = \max_{j=1,\ldots,n_{\text{run}}} (fitness(j)), \tag{42}$$

$$fitness(j)_{\min} = \min_{j=1,\ldots,n_{\text{run}}} (fitness(j)), \tag{43}$$

$$positive(j)_{\max} = \max_{j=1,\ldots,n_{\text{run}}} (positive(j)), \tag{44}$$

$$positive(j)_{\min} = \min_{j=1,\ldots,n_{\text{run}}} (positive(j)), \tag{45}$$

$$negative(j)_{\max} = \max_{j=1,\ldots,n_{\text{run}}} (negative(j)), \tag{46}$$

$$negative(j)_{\min} = \min_{j=1,\ldots,n_{\text{run}}} (negative(j)). \tag{47}$$

Aby móc oceniać jakość całego procesu indukcji gramatyki jedną miarą, wprowadzono syntetyczny estymator $fitness_{\text{avg}}$, który jest uśrednioną po liczbie kroków ewolucyjnych kompetencją ogólną gramatyki

$$fitness_{\text{avg}} = \frac{\sum_{j=1}^{n_{\max}} fitness(j)}{n_{\max}}. \tag{48}$$

Podobne uśrednione miary można wprowadzić dla kompetencji pozytywnej

$$positive_{\text{avg}} = \frac{\sum_{j=1}^{n_{\max}} positive(j)}{n_{\max}} \tag{49}$$

i kompetencji negatywnej

$$negative_{\text{avg}} = \frac{\sum_{j=1}^{n_{\max}} negative(j)}{n_{\max}}. \tag{50}$$

Zestaw estymatorów modelu GCS zamykają wartości maksymalne i minimalne poszczególnych kompetencji po zakończonym procesie indukcji gramatyki:

$$fitness_{\max} = \max_{j=1,\ldots,n_{\max}} (fitness(j)_{\max}), \tag{51}$$

$$fitness_{\min} = \min_{j=1,\ldots,n_{\max}} (fitness(j)_{\min}), \tag{52}$$

$$positive_{\max} = \max_{j=1,\ldots,n_{\max}} (positive(j)_{\max}), \tag{53}$$

$$positive_{\min} = \min_{j=1,\ldots,n_{\max}} (positive(j)_{\min}), \tag{54}$$

$$negative_{\max} = \max_{j=1,\ldots,n_{\max}} (negative(j)_{\max}), \tag{55}$$

$$negative_{\min} = \min_{j=1,\ldots,n_{\max}} (negative(j)_{\min}). \tag{56}$$

Dodatkowym kryterium oceny wyindukowanej gramatyki może być również liczba nieterminali $|N|$ oraz liczba produkcji $|P|$, za pomocą których gramatyka została wyrażona.

5.1.2. Koszt indukcji

Jako koszt indukcji przyjmowany będzie nakład obliczeń potrzebnych do wyuczenia przez model GCS zadanej gramatyki. Wygodnym parametrem – ze względu na łatwość wyznaczania – jest liczba iteracji algorytmu. Za wyborem tego kryterium przemawia dodatkowo fakt, że jest powszechnie używany podczas prezentowania wyników wnioskowania gramatycznego. Pod uwagę nie będzie brany czas pracy procesora ze względu na brak danych porównawczych w literaturze.

Estymator *nSuccess* oblicza procentową liczbę iteracji indukcji gramatyki, które zakończyły się sukcesem, tj. wyindukowana podczas przebiegu uczenia gramatyka osiągnęła stuprocentowe przystosowanie ($f_G = 100\%$). Wartość estymatora może być też podawana jako iloraz liczby iteracji zakończonych sukcesem do liczby wszystkich iteracji.

Drugim estymatorem związanym z oceną kosztu indukcji jest *nEvals*[104], obliczający średnią liczbę kroków ewolucyjnych (uśrednianych po liczbie iteracji n_{run}) potrzebnych do osiągnięcia 100% przystosowania gramatyki.

Dodatkowymi estymatorami są *minEvals* i *maxEvals* – odpowiednio minimalna i maksymalna liczba kroków ewolucyjnych spośród wszystkich iteracji, po której system znalazł gramatykę zgodną ze zbiorem uczącym.

5.1.3. Dokładność generalizacji

Testy generalizacji, w których sprawdzana jest dokładność wyindukowanej gramatyki na nieznanym wcześniej zbiorze testowym, oceniane są przez procentowo wyrażoną liczbę poprawnie sklasyfikowanych przykładów ze zbioru testowego. Ponieważ test generalizacji polega na jednokrotnym sparsowaniu przez gramatykę wszystkich zdań ze zbioru testowego, dokładność generalizacji *nGen* może być wyrażona wzorem na wartość funkcji dopasowania gramatyki f_G (patrz wzór (38))

[104] Oznaczenia *nSuccess* oraz *nEvals* zostały zaczerpnięte z pracy (Lucas i Reynolds 2005).

$$nGen = \frac{U_p + NU_n}{|R|}, \tag{57}$$

gdzie:

U_p – liczba sparsowanych zdań poprawnych,

NU_n – liczba niesparsowanych zdań niepoprawnych,

$|R|$ – liczba zdań w zestawie uczącym.

Dodatkową informacją, którą można uzyskać podczas testu generalizacji, jest średni procent sparsowanych zdań poprawnych ($nGen_{\text{pos}}$) i średni procent sparsowanych zdań niepoprawnych ($nGen_{\text{neg}}$)[105] opisanych odpowiednio wzorami:

$$nGen_{\text{pos}} = \frac{U_p}{|R^+|}, \tag{58}$$

$$nGen_{\text{neg}} = \frac{U_n}{|R^-|}, \tag{59}$$

gdzie U_n – liczba sparsowanych zdań niepoprawnych.

5.2. Zbiory uczące

Przegląd literatury dotyczącej indukcji gramatyk lub równoważnych im automatów z zastosowaniem algorytmów ewolucyjnych (patrz podrozdz. 1.5.6) wskazuje, że nie ma powszechnie uznanych, przyjętych i stosowanych zbiorów uczących i testowych. Autorzy nowych metod indukcji gramatycznej, o ugruntowanej pozycji naukowej, jak przykładowo Yasubumi Sakakibara, publikujący w uznanych pismach, jak *Pattern Recogniton* (Sakakibara 2005), testują częstokroć nowe algorytmy na wybranych arbitralnie dwóch, trzech językach (w przypadku cytowanej pracy są to $a^m b^m c^n$ oraz $ac^m \cup bc^m$). Z drugiej strony, badacze zajmujący się uczącymi się systemami klasyfikującymi, do których można zaliczyć ewolucyjny model GCS, testują nowe modele w powszechnie akceptowanym środowisku *Maze* (Gerard i in. 2002), *Woods* (McMahon i in. 2005), nie wspominając już klasycznego *problemu multipleksera* (Huang i Sun 2004). Niestety, w przypadku modelu GCS, jego unikalnej architektury oraz przeznaczenia, wspomniane wyżej środowiska testowe nie są do przyjęcia.

[105] Warto zauważyć, że wprowadzone estymatory *fitness*, *postive*, *negative* oraz *nGen*, $nGen_{\text{pos}}$, $nGen_{\text{neg}}$ pozwalają również w prosty sposób na wyrażenie efektywności procesu uczenia oraz generalizacji za pomocą miar stosowanych głównie w klasyfikacji przypadków medycznych: *true positive*, *true negative*, *false positive*, *false negative*. Przykład zastosowania tych miar zawiera podrozdz. 6.2.2.

W związku z powyższym przed modelem GCS postawiono zadanie indukcji jak najpełniejszego zbioru gramatyk niewychodzących poza klasę gramatyk języków bezkontekstowych, które jednocześnie spotyka się najczęściej w literaturze przedmiotu. Indukcją objęto gramatyki formalne, naturalne oraz biosekwencje. W ramach gramatyk formalnych uczeniem objęto wyrażenia regularne, należące do tzw. zbioru Tomity, wybrane języki bezkontekstowe oraz gramatykę dziecięcą (*toy-grammar*).

5.2.1. Wyrażenia regularne

Powszechnie uznanym zbiorem testowym w indukcji automatów DFA jest zestaw siedmiu języków zdefiniowanych po raz pierwszy w pracy (Tomita 1982):

L1: a^*,

L2: $(ab)^*$,

L3: $(b|aa)^*(a^*|(abb(bb|a)^*))$
dowolne zdanie nad $\{a, b\}$ bez nieparzystej liczby symboli b po nieparzystej liczbie a[106],

L4: $a^*((b|bb)aa^*)^*(b|bb|a^*)$
dowolne zdanie nad $\{a, b\}$ niezawierające podciągu trzech lub więcej symboli b,

L5: $((aa|bb)^*((ba|ab)(bb|aa)^*(ba|ab)(bb|aa)^*)^*(aa|bb)^*$
dowolne zdanie nad $\{a, b\}$ zawierające parzystą liczbę symboli a i parzystą liczbę symboli b[107],

L6: $((b(ba)^*(a|bb))|(a(ab)^*(b|aa)))^*$
dowolne zdanie nad $\{a, b\}$ takie, że różnica liczby symboli a i liczby symboli b jest wielokrotnością liczby 3,

L7: $b^*a^*b^*a^*$.

Metody wnioskowania gramatycznego, których celem jest indukcja automatów skończonych DFA lub równoważnych im wyrażeń regularnych, można podzielić na klasę metod uczenia aktywnego lub pasywnego (Bongard i Lipson 2005). Uczenie aktywne zakłada, że algorytm uczący ma czynny wpływ na zestaw danych uczących, który dobiera w zależności od aktualnego stanu konstruowanego modelu. Uczenie aktywne polega najczęściej na iteracyjnym zadawaniu pytań o przynależność, a zatem ilość danych uczący wzrasta wraz z upływem czasu uczenia. W uczeniu pasywnym

[106] Zbiór uczący języka L3 podany w (Luke i in. 1999) zawiera dwa nieprawidłowe negatywne zdania uczące *baaabbaaba* oraz *aabaaabbaab*.

[107] W literaturze spotyka się również opisową definicję języka L5 jako zbiór zdań o parzystej liczbie liczby par symboli (*ab*) i (*ba*). W (Delgado i Pegalajar 2003) podane jest wyrażenie regularne tak definiowanego języka Tomity w formie $((ab|ba)(ab|ba))^*$. Na marginesie warto wspomnieć, że w dosyć często cytowanej pracy (Luke i in. 1999) podane jest niepoprawne wyrażenie:

$$(((a|b)(a|b))^*(a|b))|((aa|bb)^*((ba|ab)(bb|aa)^*(ba|ab)(bb|aa)^*)^*(aa|bb)^*),$$

które dopuszcza zdania składające się z nieparzystej liczby symboli a i b.

– a do takiej klasy metod należy model GCS – dane uczące są już zadane przed rozpoczęciem procesu indukcji. W pracach (Pitt 1989, Porat i Feldman 1991, Dupont 1996, Lang i in. 1998) zbiór danych uczących był wybierany losowo, w (Pao i Carr 1978, Parekh i Honovar 1996) zestaw uczący był strukturalnie kompletny, w (Oncina i García 1992) zastosowano zbiór charakterystyczny, a w (Angluin 1981) kompletny „żywy" zbiór[108] (*live complete set*). Luke i in. (1999) oraz Lucas i Reynolds (2005) przyjęli do inferencji języków Tomity zbiory przykładów zbilansowanych, tj. zawierające równą liczbę przykładów pozytywnych i negatywnych. Zbiory te stosowane były również przez innych badaczy, m.in. (Tomita 1982, Angeline 1997), a z niewielkimi zmianami również (Angeline 1994) i (Waltrous i Kuhn 1992).

Na szczególną uwagę zasługują metody zaproponowane w (Luke i in. 1999) i (Lucas i Reynolds 2005). Po pierwsze, obie prace prezentują ewolucyjne algorytmy indukcji, a zatem klasę metod, do której zalicza się również model GCS, po drugie – opublikowane wyniki należą do najlepszych ze znanych w literaturze i to nie tylko wśród metod stosujących przetwarzanie ewolucyjne. Z tego względu, dla celów porównawczych, model GCS będzie indukował języki Tomity na identycznych zbiorach uczących, jakie przyjęto we wspominanych pracach. Ta sama zasada dotyczyć będzie testów generalizacji.

W załączniku A zamieszczono zbiory uczące języków Tomity. Zbiory zostały podane w stosowanym przez program gcs formacie *abbadingo*. Ze względu na gramatykę indukowaną w PNC założono, że języki nie generują elementu pustego ε. Mała moc zbiorów uczących (największy z nich zawiera w sumie zaledwie 24 przykłady) powoduje, że mamy do czynienia z językami „rzadkimi" (*sparse*), w których liczba generowanych przykładów pozytywnych jest znacznie mniejsza od możliwej liczby przykładów negatywnych. Stanowi to dodatkową trudność dla algorytmu indukującego, zwłaszcza podczas generalizacji. Testy generalizacji zostały przeprowadzone na pełnej populacji ciągów symboli a i b, o długości nieprzekraczającej 15 znaków (65 534 zdań testowych).

5.2.2. Języki bezkontekstowe

Wśród formalnych języków bezkontekstowych najczęściej indukowany jest następujący zestaw:

AB: dowolne zdanie nad $\{a, b\}$ zawierające taką samą liczbę symboli a i b,
AnBn: $a^n b^n$,
BRA1: język zbilansowanych nawiasów,
BRA3: język zbilansowanych nawiasów trzech typów,
PAL2: palindromy nad $\{a, b\}$ o parzystej długości.

[108] Kompletny „żywy" zbiór zawiera ciąg dla każdego „żywego" stanu automatu. Stan automatu jest „żywy", jeżeli istnieje przejście z tego stanu do stanu końcowego (Angluin 1981).

5.2.3. Gramatyka dziecięca

Gramatyka dziecięca (*toy-grammar*) jest przykładem języka bezkontekstowego opisującego uproszczoną strukturę języka naturalnego[109]. Okrojona gramatyka języka naturalnego jest wyrażona przez poniższy zestaw produkcji nieterminalnych:

$S \rightarrow np\ vp$,
$np \rightarrow det\ n \mid np\ pp$,
$pp \rightarrow prep\ n$,
$vp \rightarrow v\ np \mid vp\ pp$,

gdzie: *np* oznacza grupę rzeczownikową (*nominal phrase*), *vp* – grupę czasownikową (*verbal phrase*), *det* – rodzajnik (*determiner*), *n* – rzeczownik (*noun*), *pp* – grupę przyimkową (*prepositional phrase*), *prep* – przyimek (*preposition*), *v* – czasownik (*verb*).

5.2.4. Korpusy językowe

Zadaniem, które rozwiązuje model GCS jest indukcja CFG na podstawie etykietowanych zdań uczących. Możliwe jest zatem zastosowanie modelu w indukcji gramatyki na podstawie naturalnego korpusu językowego[110] pod następującymi warunkami:

- proces uczenia zasilany jest zbiorem uczącym składającym się ze zdań poprawnych i niepoprawnych;
- korpus językowy wymaga oznakowania morfosyntaktycznego (*part-of-speech tags*, POS)[111], czyli przejścia z tekstu języka naturalnego na zapis symboliczny, składający się z sekwencji tagów.

Uczenie gramatyki na podstawie oznakowanego jedynie morfosyntaktycznie korpusu językowego kwalifikuje proces indukcji za pomocą modelu GCS do uczenia bez nadzoru[112] (patrz przypis [8]).

---

[109] Lankhorst (1994) używa pojęcia *micro – NL language*.

[110] Język naturalny jest umieszczany zwykle w hierarchii Chomsky'ego na wysokości języków bezkontekstowych, niekiedy kontekstowych (Partee i in. 1993). Powszechne modelowanie struktury języka naturalnego środkami gramatyk bezkontekstowych (Pullum i Gazdar 1982, Gazdar i Pullum 1985, Shieber 1985) jest uzasadnione ograniczonymi kompetencjami językowymi człowieka, chociażby na praktycznie akceptowalną długość zdania (patrz również przypis [44]).

[111] Przejście z tekstu języka naturalnego na ciąg tagów morfosyntaktycznych, choć powszechnie stosowane w literaturze przedmiotu (Clark 2001a, Klein i Manning 2003), niesie ze sobą niebezpieczeństwo utraty istotnych informacji, niezbędnych w indukcji syntaktyki. Wynika to z faktu, że wiele konstrukcji językowych silnie zależy od specyficznych własności poszczególnych słów języka. Niemniej jednak, takie przetworzenie korpusu jest konieczne ze względu na efektywność analizy oraz jest wystarczające dla potrzeb indukcji struktur syntaktycznych.

[112] W tym miejscu potrzebny może być komentarz, ze względu na mogące pojawić się terminologiczne zamieszanie. Otóż z punktu widzenia uczenia maszynowego, uczenie w modelu GCS jest typo-

Zdecydowana większość opublikowanych metod uczenia bez nadzoru stosuje statystyczne metody indukcji (Carroll i Charniak 1992, Pereira i Schabes 1992, Brill 1993, Stolcke i Omohundro 1994, Klein i Manning 2003, Klein i Manning 2005, Solan i in. 2005). Inne metody opierają się na analizie dystrybucyjnej (*distributional analysis*), jak (Adriaans 1999, Klein i Manning 2001, van Zaanen 2002), czy też kompresji (Chen 1995, Clark 2001b, Keller i Lutz 1997, Wolff 2003). Wszystkie wyżej wymienione grupy metod uczenia bez nadzoru indukują gramatykę jedynie na podstawie zdań z korpusu. Metody statystyczne posiłkują się twierdzeniem, które mówi, że probabilistyczna CFG (PCFG) może być wyuczona w granicy już na podstawie zdań poprawnych[113] (Horning 1969), metody dystrybucyjne bazują na modelu uczenia PAC (lub PACS), a kompresja na modelu MDL (patrz podrozdz. 1.4).

W indukcji bez nadzoru stosuje się stosunkowo często (płatne) repozytoria językowe (*treebanks*), które oprócz zbioru zdań języka naturalnego zawierają tzw. metadane, jak: oznaczenia końców zdań, akapitów, oznaczenia morfosyntaktyczne słów, informacje o strukturze syntaktycznej zdań, informacje semantyczne (np. podział korpusu na części tematyczne). Do najbardziej znanych korpusów należy Penn treebank (Marcus i in. 1993) oraz Wall Street Journal (WSJ) i ATIS (Hemphill i in. 1990), które są fragmentami tego pierwszego, a także British National Corpus (BNC) (Burnard 1995), OVIS (Bonnema i in. 1997) i Brown (Francis i Kuera 1982).

Model indukcji gramatyki bezkontekstowej GCS jest istotnie różny od wzmiankowanych wyżej metod uczenia bez nadzoru. Rezultatem działania modelu nie jest PCFG lub jakiś rodzaj gramatyki kategorialnej, lecz nieprobabilistyczna gramatyka bezkontekstowa. Proces uczenia skupia się na rozwiązywaniu problemu przynależności zdania do języka, a nie budowania jego struktury syntaktycznej (chociaż w procesie indukcji znajdowane są wszystkie możliwe drzewa rozbioru). Wreszcie wnioskowanie wymaga etykietowanego zbioru uczącego. Jedną z nielicznych w literaturze przedmiotu prac spełniających postawione założenia jest (Aycinena i in. 2003). Podobnie jak ma to miejsce w modelu GCS, indukowana gramatyka jest reprezentowana przez postać normalną Chomsky'ego, a rozbiór dokonywany jest przez parser tablicowy CYK. Doktoranci ze Stanford indukowali gramatykę, stosując algorytm genetyczny na podstawie dziewięciu korpusów językowych.

Korpus children

Korpus tworzą wybrane teksty z literatury dziecięcej, dostępnej pod adresem http://www.magickeys.com/books.

---

wym przykładem uczenia z nadzorem, zwanym również inaczej *uczeniem z nauczycielem* (Cichosz 2001). Etykietowany zbiór uczący jest przykładem informacji instruktażowej, zawierającej pożądane odpowiedzi systemu. Jednak rozpatrując proces uczenia modelu w kontekście gramatycznej indukcji z korpusu językowego, uczenie to stanowi klasyczny przykład uczenia bez nadzoru, gdyż algorytm nie korzysta z żadnych dodatkowych informacji o strukturze korpusu.

[113] Twierdzenie nie wskazuje jednak na żaden praktyczny algorytm.

Korpus wizard

Na korpus składają się obszerne fragmenty z książki *Czarownik z krainy Oz* (*The Wizard of Oz*) L. Franka Bauma, dostępne pod adresem

http://www.ucalgary.ca/dkbrown/storclas.html.

Korpus alice

W skład korpusu wchodzą obszerne fragmenty z książki *Alicja w krainie czarów* (*Alice in Wonderland*) L. Carrolla, dostępne pod adresem

http://www.ucalgary.ca/dkbrown/storclas.html.

Korpus tom

Korpus składa się z obszernych fragmentów książki *Tomek Sawyer* (*Tom Sawyer*) M. Twaina, dostępnej pod adresem

http://www.infomotions.com/alex/authors.html.

Korpus brown

Korpus tworzy pięć nieoznakowanych fragmentów z repozytorium Browna[114], oznaczanych brown_a do brown_e

(afs/ir.stanford.edu/data/linguistic-data/Brown/ICAME-Brown1).

Wzrastający pod względem stopnia trudności tekstu poziom korpusów, począwszy od literatury dziecięcej, przez młodzieżową, aż do literatury dla dorosłych, ma na celu zbadanie efektywności zastosowanej metody indukcji.

Aby przygotować odpowiedni zestaw uczący, słowa w korpusie zostały najpierw oznaczone symbolami morfosyntaktycznymi przy użyciu znanego *taggera* autorstwa Brilla (1993). Następnie usunięte zostały słowa języka angielskiego, a pozostałe ciągi tagów zredukowano do 7 nieterminali (w nawiasach podano zastępowane zbiory tagów):

a: rzeczowniki, zaimki (NN, NNP, NNPS, NNS, PRP, WP);

b: czasowniki, czasowniki posiłkowe (MD, VB, VBD, VBG, VBN, VBP, VBZ);

c: przymiotniki, liczebniki, zaimki dzierżawcze (CD, JJ, JJR, JJS, PRP$, WP$);

d: przysłówki (RB, RBR, RBS, WRB);

e: przyimki, partykuły (IN, RP, TO);

f: spójniki, rodzajniki (CC, DT, EX, PDT, WDT);

g: pozostałe – słowa obce, symbole, wykrzykniki (FW, SYM, UH).

W ostatniej fazie tworzenia zbioru uczącego, zredukowany ciąg tagów formatowany był do używanego przez program gcs formatu *abbadingo*. Zbiór uczący został uzupełniony o przykłady negatywne, reprezentowne przez losowo wygenerowane ciągi zredukowanych tagów o długości od 5 do 15 słów zgodnie z rozkładem normalnym. W załączniku B zamieszczono przykładową transformację źródłowego korpusu językowego do postaci formatu akceptowanego przez program.

[114] Na korpus Browna składa się ponad milion słów amerykańskiej prozy wydanej w Stanach Zjednoczonych w 1961 r.

Tabela 1. Statystyki zbiorów uczących i testowych dla korpusów językowych

Korpus	$\|U\|$	$\|U^+\|$	$\|U^-\|$	$\|T\|$	$\|T^+\|$	$\|T^-\|$
children	1972	986	986	986	493	493
wizard	3080	1540	1540	1542	771	771
alice	2024	1012	1012	1014	507	507
tom	7202	3601	3601	3602	1801	1801
brown_a	5578	2789	2789	2790	1395	1395
brown_b	3560	1780	1780	1782	891	891
brown_c	2198	1099	1099	1100	550	550
brown_d	2124	1062	1062	1064	532	532
brown_e	5022	2511	2511	2512	1256	1256

W tabeli 1 zebrano wybrane dane statystyczne opisujące przetwarzane korpusy językowe U oraz korpusy testowe T. W poszczególnych kolumnach tabeli umieszczono: moc zbioru uczącego $|U|$, moc zbioru zdań uczących pozytywnych $|U^+|$, moc zbioru zdań uczących negatywnych $|U^-|$, moc zbioru testowego $|T|$, moc zbioru zdań testowych pozytywnych $|T^+|$ oraz moc zbioru zdań testowych negatywnych $|T^-|$. Korpusy językowe są zbilansowane i stosunkowo obszerne. Testy generalizacji zostały wykonane na wyjętych z korpusów i nieprezentowanych wcześniej zdaniach, uzupełnionych o losowo wygenerowane ciągi nieterminali nienależące do indukowanego języka.

5.3. Indukcja wybranych języków regularnych

Indukcję języków formalnych za pomocą modelu GCS rozpoczęły eksperymenty na siedmiu językach Tomity reprezentujących problem indukcji automatów skończonych lub równoważnych im wyrażeń regularnych. Podczas wszystkich doświadczeń stosowano poniższy zestaw parametrów modelu:

Zmienne decyzyjne:

f_{GA} = tak – zmienna zezwalająca na uruchomienie algorytmu genetycznego,

f_{GA}^1 = ruletka – rodzaj zastosowanej selekcji podczas wyboru pierwszej produkcji,

f_{GA}^2 = ruletka – rodzaj zastosowanej selekcji podczas wyboru drugiej produkcji,

f_{kor} = nie – zmienna zezwalająca na uruchomienie korekcji gramatyki,

f_{cs} = tak – zmienna zezwalająca na uruchomienie operatora pokrycia startowego,

f_{cp} = tak – zmienna zezwalająca na uruchomienie operatora pokrycia pełnego,

f_{cu} = nie – zmienna zezwalająca na na uruchomienie operatora pokrycia uniwersalnego.

Parametry ciągłe

$n_{\max}$ = 5000 – maksymalna liczba kroków ewolucyjnych,

n_{run} = 50 – liczba iteracji,

$n_P = 40$ – rozmiar populacji,
$n_{\text{start}} = 30$ – liczba początkowych produkcji nieterminalnych,
$n_N = 19$ – liczba symboli nieterminalnych,
$n_T = 2$ – liczba symboli terminalnych,
$p_k = 0{,}2$ – prawdopodobieństwo krzyżowania dla algorytmu genetycznego,
$p_m = 0{,}8$ – prawdopodobieństwo mutacji dla algorytmu genetycznego,
$p_i = 0$ – prawdopodobieństwo inwersji dla algorytmu genetycznego,
$p_a = 0$ – prawdopodobieństwo zastosowania operatora pokrycia agresywnego,
$cf = 18$ – współczynnik ścisku,
$cs = 3$ – podpopulacja ścisku,
$ba = 1$ – kwota bazowa,
$raf = 0{,}5$ – współczynnik zmniejszania kwoty bazowej,
$n_{\text{elit}} = 0$ – wielkość elity,
$w_p = 1$ – waga rozbioru zdania poprawnego,
$w_n = 2$ – waga rozbioru zdania niepoprawnego,
$w_c = 1$ – waga funkcji klasycznej klasyfikatora,
$w_f = 0$ – waga funkcji płodności klasyfikatora,
$f_0 = 0{,}5$ – miara użyteczności klasyfikatora niebiorącego udziału w parsowaniu.

Zbiór ten nie był dobierany specjalnie do postawionego zadania, zawierał domyślne ustawienia parametrów stosowane podczas wstępnych eksperymentów, a wyłączał m.in. takie własności modelu, jak: inwersja, pokrycie agresywne, sukcesja elitarna, czy płodność klasyfikatorów. Liczba dziewiętnastu symboli terminalnych była wybrana ze względu na właściwość programu gcs, który ostatni dopuszczalny symbol nieterminalny przyjmuje za symbol startowy gramatyki (symbolem dziewiętnastym w alfabecie łacińskim jest litera *S*). Obydwie produkcje wybierane były selekcją ruletkową. Dominująca rola mutacji (wartość 0,8 wobec wartości 0,2 dla krzyżowania) wynika z udowodnionego w literaturze efektywniejszego wpływu właśnie operatora mutacji w populacjach o małych rozmiarach (Mühlenbein i Schlierkamp-Voosen 1995), do których należy również populacja produkcji modelu GCS.

W tabeli 2 zebrano podstawowe statystyki zbiorów uczących U dla poszczególnych języków oraz zbiorów testowych T. W poszczególnych kolumnach tabeli znajdują się: moc zbioru uczącego $|U|$, moc zbioru zdań uczących pozytywnych $|U^+|$, moc zbioru zdań uczących negatywnych $|U^-|$, moc zbioru testowego $|T|$, moc zbioru zdań testowych pozytywnych $|T^+|$ oraz moc zbioru zdań testowych negatywnych $|T^-|$. Zbiory uczące dla języków Tomity są zbilansowane, ale mało liczne. Testy generalizacji zostały wykonywane na pełnej populacji ciągów symboli *a* i *b*, o długości nieprzekraczającej 15 znaków – analogicznie jak w pracach (Luke i in. 1999) oraz (Lucas i Reynolds 2005). Zbiory testowe są obszerne (65 534 zdań każdy) i niezbilansowane, gdyż każdy z nich zawiera znacznie większą liczbę zdań nienależących do języka, niż należących.

Tabela 2. Statystyki zbiorów uczących i testowych dla języków Tomity

Język	$\|U\|$	$\|U^+\|$	$\|U^-\|$	$\|T\|$	$\|T^+\|$	$\|T^-\|$
L1	16	8	8	65 534	15	65 519
L2	15	5	10	65 534	7	65 527
L3	24	12	12	65 534	9447	56 087
L4	19	10	9	65 534	23 247	42 287
L5	21	9	12	65 534	10 922	54 612
L6	21	9	12	65 534	21 844	43 690
L7	20	12	8	65 534	2515	63 019

W tabeli 3 przedstawiono uzyskane przez model gcs wyniki podczas uczenia języków Tomity. Wyniki zostały uśrednione po 50 niezależnych uruchomieniach programu dla każdego indukowanego języka ($n_{\text{run}} = 50$). Cztery pierwsze kolumny tablicy podają wartości estymatorów kosztu indukcji danego języka: *nSuccess* to iloraz liczby iteracji zakończonych sukcesem do liczby wszystkich iteracji, *nEvals* to średnia liczba kroków ewolucyjnych, podczas których algorytm znajduje 100% przystosowaną gramatykę, *s* to odchylenie standardowe *nEvals*, *minEvals* to najmniejsza ze wszystkich iteracji liczba kroków ewolucyjnych potrzebna do znalezienia rozwiązania. W ostatniej kolumnie umieszczono wartość estymatora dokładności generalizacji *nGen*.

Tabela 3. Wyniki indukcji języków Tomity uzyskane przez model GCS

Język	*nSuccess*	*nEvals*	*s*	*minEvals*	*NGen*
L1	50/50	2,02	0,14	2	100
L2	50/50	2,28	0,45	2	100
L3	1/50	666	–	666	100
L4	24/50	2455,17	1626,25	52	100
L5	50/50	200,58	236,52	11	92,40
L6	49/50	1471,39	1074,31	130	96,90
L7	11/50	2902,18	1277,30	729	92

Model GCS znalazł dla każdego języka gramatykę w mniej niż 5000 krokach ewolucyjnych ($n_{\max} = 5000$). W przypadku języków L1, L2 oraz L5 odpowiednia gramatyka była znajdowana podczas każdej iteracji eksperymentu, dla języka L6 tylko podczas jednej iteracji gramatyka nie została znaleziona. Niejako na drugim biegunie kosztów indukcji są eksperymenty przeprowadzone dla języka L3, podczas których tylko jedna z pięćdziesięciu iteracji zakończyła się sukcesem. Indukcja języków L1 oraz L2 okazała się zadaniem trywialnym dla modelu, gdyż średnio kończyła się w drugim kroku ewolucyjnym. Poza oczywistymi przypadkami indukcji języków L1, L2 i L3 widać wyraźną różnicę pomiędzy średnią (wraz ze standardowym odchyleniem) a minimalną liczbą kroków potrzebnych do znalezienia rozwiązania. Pełny obraz indukcji daje dopiero zestawienie kosztów indukcji *nSuccess* oraz *nEvals* z dokładnością generalizacji *nGen*. Dla czterech języków model GCS znalazł gramatykę, która bezbłędnie klasyfikuje zbiór

65 534 zdań zbioru testowego. Dla pozostałych trzech języków L5–L7 dokładność generalizacji jest większa bądź równa 92%. Zwrócić należy uwagę, że analiza jedynie pojedynczego estymatora jest niepełna. Porównując przykładowo wyniki indukcji języków L3 i L4, można stwierdzić – ponieważ średnia liczba kroków niezbędnych do znalezienia odpowiedniej gramatyki jest niższa dla języka L3, zatem ten proces uczenia przebiega szybciej. Jednak dopiero zestawiając wartości estymatorów *nEvals* z *nSuccess* dla poszczególnych języków, można wyprowadzić prawidłowe wnioski. Do indukcji języka L3 średnio potrzeba 50 iteracji po 665 kroków każda (33 250 kroków), natomiast poprawna gramatyka języka L4 znajdowana jest średnio już po 2 iteracjach, każda po 2454 kroki ewolucyjne (4908 kroków).

Poniżej zestawiono zbiór produkcji gramatyki bezkontekstowej w PNC ręcznie stworzonej dla języka L4: $a^*((b|bb)aa^*)^*(b|bb|a^*)$, który nie dopuszcza zdań zawierających podciągi trzech lub więcej symboli *b*:

1. $S \rightarrow a$
2. $S \rightarrow b$
3. $S \rightarrow BB$
4. $S \rightarrow AD$
5. $S \rightarrow DI$
6. $S \rightarrow DH$
7. $S \rightarrow GH$
8. $S \rightarrow AS$
9. $S \rightarrow EG$
10. $S \rightarrow CA$
11. $S \rightarrow CF$
12. $I \rightarrow GH$
13. $H \rightarrow a$
14. $H \rightarrow AD$
15. $H \rightarrow b$
16. $H \rightarrow BB$
17. $G \rightarrow EG$
18. $G \rightarrow CA$
19. $G \rightarrow CF$
20. $F \rightarrow AD$
21. $E \rightarrow CA$
22. $E \rightarrow CF$
23. $D \rightarrow AD$
24. $D \rightarrow a$
25. $C \rightarrow b$
26. $C \rightarrow BB$
27. $B \rightarrow b$
28. $A \rightarrow a$.

Zbiór produkcji został stworzony zgodnie z regułami przekształcenia wyrażenia regularnego na zbiór produkcji gramatyki bezkontekstowej w postaci normalnej Chomsky'ego (patrz załącznik C). Zestawmy ręcznie utworzony zbiór produkcji, ze zbiorem produkcji wyuczonym przez model:

1. $S \rightarrow a$
2. $S \rightarrow b$
3. $S \rightarrow HS$
4. $S \rightarrow BB$
5. $S \rightarrow SA$
6. $S \rightarrow AS$
7. $H \rightarrow SA$
8. $B \rightarrow b$
9. $A \rightarrow a$.

Wyindukowany zbiór produkcji składa się z ponad trzykrotnie mniejszej liczby produkcji i precyzyjnie opisuje nie tyle wyrażenie regularne L4 (chociaż oczywiście jest z nim zgodne), co raczej opis werbalny języka L4. Jest to oczywisty wynik poszukiwania reguł opisujących zaprezentowany algorytmowi uczącemu zestaw zdań uczących (patrz załącznik A). Wyuczona reguła $S \rightarrow BB$ jest jedyną w zbiorze produkcji, która może w połączeniu z innymi regułami zaakaceptować (lub też wygenerować) maksymalny podciąg dwóch symboli *b*.

Przypatrzmy się jeszcze zbiorowi produkcji wyuczonemu dla języka L7, który opisuje się wyrażeniem $b^*a^*b^*a^*$. Uczenie języka L7 zakończyło się sukcesem w 11 z 50 iteracji eksperymentu, ale dokładność generalizacji wyindukowanej gramatyki jest na poziomie 92,00% (patrz tabela 3). Poniżej zamieszczono wyuczony przez model zbiór dziewięciu produkcji:

1. $S \rightarrow b$
2. $S \rightarrow a$
3. $S \rightarrow SA$
4. $S \rightarrow BC$
5. $S \rightarrow AB$
6. $S \rightarrow BS$
7. $C \rightarrow SB$
8. $B \rightarrow b$
9. $B \rightarrow a$.

Powyższy zbiór produkcji w sposób prawidłowy klasyfikuje zdania ze zbioru uczącego, ale jednocześnie akceptuje (generuje) zdania spoza języka (np. $S \rightarrow BC \rightarrow bSb \rightarrow baCb \rightarrow baSab \rightarrow babab$), chociaż nie jest w stanie zaakceptować wszystkich zdań należących do języka, jak np. *abb*.

Zarówno uczenie, jak i testy generalizacji przebiegały na identycznych zbiorach przykładów, jak te zastosowane w (Luke i in. 1999) oraz (Lucas i Reynolds 2005). Obie wymienione prace prezentują – podobnie jak model GCS – ewolucyjne podejście do indukcji automatów, a opublikowane w nich wyniki należą do najlepszych ze znanych w literaturze i to nie tylko wśród metod ewolucyjnych.

W metodzie zaproponowanej przez Luke'a i in. (1999), zwanej dalej metodą GP („*Genetic*" *Programming*), indukowany automat skończony jest reprezentowany przez genom składający się z nieograniczonej liczby genów. Każdy gen reprezentuje stan automatu. Pozostałe atrybuty każdego genu, zwane przez autorów artykułu „chemicznym wzorcem" (*chemical template*), są używane do sterowania przejściami pomiędzy stanami. Do każdego genu przypisana jest zmienna logiczna wskazująca, czy skojarzony z genem stan jest stanem akceptującym. Zastosowany model regulacji międzygenowych został zainspirowany mechanizmami genetycznymi występującymi u muszki owocowej (*Drosophila melanogaster*).

Zupełnie odmienną koncepcję ewolucji automatu zastosowano w (Lucas i Reynolds 2005). Ewolucji podlega jedynie macierz przejść automatu, a „sprytny" (*smart*) algorytm etykietowania stanów indukowanego automatu uzupełnia jego opis. Zastosowanym mechanizmem ewolucyjnym jest prosta strategia ewolucyjna (1 + 1). Lucas i Reynolds zbadali trzy różne mutacje własnej metody: metodę, w której ewolucji podlegała zarówno macierz przejść automatu, jak i wektor etykiet stanów (tzw. metoda Plain), metodę, w której ewoluowano jedynie macierz przejść automatu, a liczba stanów automatu była stała i równa 10 (Smart) i wreszcie metodę, w której liczba stanów automatu była równa minimalnej liczbie stanów automatu reprezentującego badany język (nSmart).

Wyniki uzyskane w indukcji języków Tomity przez model GCS zostały porównane w pierwszej kolejności z metodą GP. Zestawiając bowiem podejście Luke'a i in. z modelem GCS oraz metodami Lucasa i Reynoldsa, należy stwierdzić, że dwa pierw-

sze mechanizmy reprezentują algorytmy o swobodnej, nieograniczonej reprezentacji rozwiązań (*variable size methods*), natomiast metody Plain, Smart i nSmart to algorytmy o ustalonym odgórnie rozmiarze danych (*fixed-size structure*). Generalnie metody o narzuconych odgórnie ograniczeniach na rozmiar danych, czy tym bardziej rozmiarze dopasowanym do rozwiązywanego zadania (jak ma to miejsce w metodzie nSmart), uzyskują porównywalnie lepsze wyniki.

W tabeli 4 zawarto porównanie rezultatów osiągniętych podczas indukcji języków regularnych Tomity w pracy (Luke i in. 1999) z wynikami uzyskanymi przez model GCS. W kolejnych parach kolumn zestawiono ze sobą wartości estymatorów *nSuccess*, *nEvals* oraz *nGen* odpowiednio dla metody GP i GCS. Jak nietrudno zauważyć, model GCS dla każdego języka uzyskał lepsze lub równe wartości poszczególnych badanych parametrów.

Tabela 4. Porównanie rezultatów uzyskanych przez model GCS i (Luke i in. 1999) dla języków Tomity

Język	*nSuccess*		*nEvals*		*nGen*	
	GP	GCS	GP	GCS	GP	GCS
L1	31/50	50/50	30	2	88,4	100
L2	7/50	50/50	1010	2	84	100
L3	1/50	1/50	12 450	666	66,3	100
L4	3/50	24/50	7870	2455	65,3	100
L5	0/50	50/50	13 670	201	68,7	92,4
L6	47/50	49/50	2580	1471	95,9	96,9
L7	1/50	11/50	11 320	2902	67,7	92

Dla obydwu modeli ewolucyjnych podobną trudność stanowiła indukcja języka L3 – tak w jednym, jak i w drugim algorytmie tylko jedna na pięćdziesiąt iteracji zakończyła się wyuczeniem prawidłowej gramatyki. Jednak o ile metodzie GP zajęło to 12 450 kroków ewolucyjnych – model GCS znalazł rozwiązanie w 666 krokach. Analizując dalej język L3, należy też zwrócić uwagę na poziom generalizacji, jaki obie metody osiągnęły. Wyindukowana przez model GCS gramatyka bezbłędnie klasyfikuje w zdecydowanej większości nieobserwowany wcześniej zbiór testowy składający się z 65 534 przykładów. Gramatyka wyuczona przez model GP dokonuje poprawnej klasyfikacji jedynie w niewiele ponad 66%. Zresztą ani jedna indukowana w eksperymentach Luke'a i in. gramatyka nie osiągnęła 100% generalizacji. Model GCS wyindukował dla 4 języków zestaw produkcji poprawnie klasyfikujących zbiór testowy. Dla języka L6 oba modele osiągnęły podobny stopień generalizacji na poziomie 96–97%. Język L6 jest dosyć charakterystyczny, gdyż obie metody osiągnęły porównywalne wyniki nie tylko w dokładności indukcji, ale również w jej kosztach. Dużą dysproporcję widać w wynikach indukcji języka L5. O ile metoda GP nie znalazła gramatyki w 100% zgodnej ze zbiorem uczącym, o tyle model GCS w każdej z pięćdziesięciu iteracji wyuczył

się poprawnie klasyfikującej zbiór uczący gramatyki. Co więcej, model GCS odpowiedniej gramatyki uczył się średnio w zaledwie 201 krokach ewolucyjnych. Dwa pierwsze języki nie sprawiły żadnej trudności dla modelu GCS – 100% poprawnie generalizującą gramatykę model znajdował średnio w drugim kroku ewolucyjnym podczas każdej iteracji eksperymentu. Metodzie GP znajdowanie gramatyki zgodnej ze zbiorem uczącym zajmowało zdecydowanie więcej kroków ewolucyjnych (30 kroków dla L1 i 1010 dla L2), nie każda iteracja zakończyła się sukcesem i żadna ze znalezionych gramatyk nie była 100% zgodna ze zbiorem testowym.

Zestawione obecnie zostaną rezultaty uzyskane przez metody GP i GCS oraz metody o stałym rozmiarze danych Plain, Smart i nSmart. W tabeli 5 pokazano koszty indukcji *nEvals* wymienionych metod. Analiza zawartości tabeli wskazuje, że dla języków L1 oraz L2 model GCS okazał się bezkonkurencyjny ze średnim rezultatem dwóch kroków ewolucyjnych – najlepszy wynik z pozostałych metod osiągnął nSmart, któremu indukcja automatu równoważnego językowi L1 zabrała 15 kroków ewolucyjnych. Dla języka L5 model GCS osiągnął porównywalny wynik do najlepszego rezultatu, który osiągnęła metoda Smart, a zdecydowanie lepszy od wyników pozostałych algorytmów ewolucyjnych (201 kroków GCS do 195 w metodzie Smart). Również w przypadku języka L3 wynik uzyskany przez model GCS był tylko nieznacznie gorszy od najlepszego, osiągniętego metodą nSmart (666 do 237). W tym wypadku należy jednak wziąć pod uwagę również wartość estymatora *nSuccess*. Jak podają autorzy artykułu (Lucas i Reynolds 2005), metoda nSmart była dwudziestokrotnie iterowana i za każdym razem nSmart znajdował gramatykę zgodną ze zbiorem uczącym[115]. W tym konkretnym wypadku zatem, dla języka L3 średnią osiągniętą przez model GCS należy dodatkowo zestawić z niską wartością estymatora *nSuccess*, wynoszącą 1/50 (patrz tab. 4). Dla pozostałych trzech języków, tj. L4, L6 i L7, wyniki uzyskane przez metody o stałej reprezentacji były zdecydowanie lepsze od wyników modelów GCS i GP.

Tabela 5. Porównanie kosztów indukcji *nEvals* języków Tomity dla różnych metod

Język	Plain	Smart	nSmart	GP	GCS
L1	107	25	15	30	2
L2	186	37	40	1010	2
L3	1809	237	833	12 450	666
L4	1453	177	654	7870	2455
L5	1059	195	734	13 670	201
L6	734	93	82	2580	1471
L7	1243	188	1377	11 320	2902

W tabeli 6 podsumowano dokładność generalizacji, jaką osiągnęły metody GCS, GP, Smart, nSmart oraz EDSM. Metoda EDSM (*Evidence Driven State Merging*), do której

[115] Lucas i i Reynolds (2005) piszą wprost jedynie o metodzie nSmart, która podczas każdej iteracji uzyskała gramatykę odpowiadającą zbiorowi trenującemu.

porównuje się również Lucas i Reynolds (2005), powstała na potrzeby międzynarodowego konkursu Abbadingo, rozegranego w 1997 r. – jego celem było współzawodnictwo w indukcji automatów skończonych (Lang i in. 1998). EDSM jest obecnie jedną z najlepszych metod indukcji DFA. Metoda jest przykładem heurystycznego algorytmu, który iteracyjnie kompresuje początkowo duży automat, za każdym razem dbając o to, by zachować prawidłową klasyfikację zbioru uczącego przed i po kompresji. Metoda EDSM była uruchomiona jednokrotnie, gdyż dla podanego zestawu uczącego jest deterministyczna. Model GCS dla czterech z siedmiu indukowanych języków (L1–L3 oraz L7) uzyskał najlepsze wyniki generalizacji ze wszystkich metod, a dla języka L4 taką samą bezbłędną generalizację jak metody nSmart oraz EDSM. Indukcja języków L5 oraz L6 dała natomiast drugi rezultat wśród porównywanych metod – istotnie wyższy od 90%.

Tabela 6. Porównanie dokładności generalizacji *Gen* języków Tomity dla różnych metod

Język	Smart	nSmart	EDSM	GP	GCS
L1	81,8	100	52,4	88,4	100
L2	88,8	95,5	91,8	84	100
L3	71,8	90,8	86,1	66,3	100
L4	61,1	100	100	65,3	100
L5	65,9	100	100	68,7	92,4
L6	61,9	100	100	95,9	96,9
L7	62,6	82,9	71,9	67,7	92

5.4. Indukcja wybranych języków bezkontekstowych

W przeciwieństwie do literatury dotyczącej indukcji języków regularnych i odpowiadających im automatów skończonych, literatura zajmująca się uczeniem języków bezkontekstowych nie dopracowała się standardowego zbioru języków testowych. Wynika to zapewne z faktu, iż dotychczas nie znaleziono efektywnych algorytmów wnioskowania dla CFG, a badania teoretyczne wskazują, że gramatyki bezkontekstowe oraz równoważne im automaty ze stosem nie są wielomianowo identyfikowane w czasie i danych (Higuera 1997). W chwili obecnej najwięcej prac z zakresu indukcji gramatyk bezkontekstowych prowadzonych jest z zastosowaniem metod stochastycznych, które w uczeniu dopuszczają jedynie stosowanie przykładów pozytywnych (Horning 1969). Zdecydowanie mniej opublikowanych metod indukuje niestochastyczną CFG lub równoważny gramatyce bezkontekstowej automat ze stosem. Wśród nich należy wymienić prace (Wyard 1991, Sen i Janakiraman 1992, Lucas 1993, Huijsen 1993, Wyard 1994, Lucas 1994, Lankhorst 1994, 1995, Smith i Witten 1995, Zomorodian 1995, Bianchi 1996, Smith i Witten 1996, Korkmaz i Ucoluk 2001, Cyre 2002, Mernik i in. 2003, Javed i in. 2004, Tsoulos i Laga-

ris 2005, Sakakibara 2005). Do tej grupy zaliczają się również prace autora z zakresu GI. W większości wymienionych publikacji trudno doszukać się wspólnego zestawu indukowanych języków (patrz podrozdz. 1.5.6), brakuje precyzyjnie opisanego procesu przygotowywania danych uczących, zbiory trenujące dobierane są intuicyjnie i do konkretnego języka, rzadko kiedy też uzyskane wyniki podlegają testom generalizacji. Wszystkie te czynniki razem utrudniają porównywanie zastosowanych metod i uzyskanych rezultatów.

Aby zobiektywizować osiągane wyniki, założono, że wybrane języki AB, AnBn, BRA1, BRA3, PAL2, PAL3 oraz gramatyka dziecięca (TOY) będą trenowane za pomocą losowo wygenerowanych zbiorów uczących, składających się z 200 przykładów nieprzekraczających długości 30 symboli i porównywalnej liczbie zdań pozytywnych i negatywnych. Przyjęcie przykładów uczących o długości do 30 symboli wynika po pierwsze z raportowanych w literaturze eksperymentów właśnie na zdaniach o takiej długości (Lankhorst 1995) oraz z faktu, że dla języka AnBn zwiększa to znacząco moc zbioru $|U^+|$. Testy generalizacji zostaną przeprowadzone, podobnie jak miało to miejsce w przypadków testów języków regularnych, na pełnej populacji zdań o długości nieprzekraczającej 15 znaków[116]. Wyjątkiem są języki BRA3 oraz TOY, które zawierają więcej niż 2 symbole terminalne[117]. Dla tych języków testy zostały przeprowadzone na losowo wygenerowanych zbiorach testowych zawierających 65 534 przykładów. W tabeli 7 zestawiono statystyki zbiorów uczących i testowych zastosowanych w eksperymentach z modelem GCS, gwiazdką zaznaczono losowe zbiory testowe dla języków BRA3 i TOY. Podczas wszystkich doświadczeń stosowano ten sam, domyślny zestaw parametrów modelu, który konfigurował model podczas indukcji języków regularnych, a zatem m.in. każdy eksperyment był iterowany 50 razy, a pojedyncza iteracja nie mogła przekraczyć 5000 kroków ewolucyjnych.

Tabela 7. Statystyki zbiorów uczących i testowych dla języków bezkontekstowych

Język	$\|U\|$	$\|U^+\|$	$\|U^-\|$	$\|T\|$	$\|T^+\|$	$\|T^-\|$
AB	200	101	99	65 534	4706	60 828
AnBn	200	15	185	65 534	7	65 527
BRA1	200	101	99	65 534	625	64 909
BRA3	200	100	100	65 534*	22274	43 260
PAL2	200	101	99	65 534	224	65 310
TOY	200	99	101	65 534*	105	65 429

[116] W żadnej ze znanych autorowi publikacji dotyczących indukcji niestochastycznych gramatyk bezkontekstowych nie stosowano tak obszernego zbioru testowego. Co prawda, Lucas (1994) indukował wybrane języki na kompletnym zbiorze zdań o długości 4 symboli, ale testy generalizacji przebiegały na kompletnym zbiorze o długości 7 symboli.

[117] Kompletny zbiór uczący dla języka BRA3 o rozmiarze do 10 symboli liczy 72 559 410 przykładów, a plik tekstowy zawierający dane uczące ma ponad 1,8 GB!

W tabeli 8 zebrano wyniki indukcji języków bezkontekstowych. Wszystkie przeprowadzone eksperymenty zakończyły się pełnym powodzeniem, tj. model GCS znalazł dla każdego języka prawidłową gramatykę zgodną nie tylko ze zbiorem uczącym, ale również z obszernym zbiorem testowym. Co więcej, indukcja gramatyki poprawnie klasyfikującej wszystkie przykłady uczące miała miejsce w każdej z pięćdziesięciu iteracji kolejnego eksperymentu. Analizując koszt indukcji, wyraźnie widać, że największą trudność w uczeniu sprawiła modelowi gramatyka palindromiczna. Świadczy o tym zarówno najwyższa średnia liczba kroków koniecznych do wyewoluowania poprawnej gramatyki (ponad 417 kroków ewolucyjnych z odchyleniem standardowym $s = 410{,}79$), jak i największa wartość estymatora *minEvals* (16).

Tabela 8. Wyniki indukcji języków bezkontekstowych uzyskane przez model GCS

Język	*nSuccess*	*nEvals*	*s*	*minEvals*	*nGen*	$\|P\|/\|P_i\|$
AB	50/50	287,84	385,25	3	100	9/13
AnBn	50/50	37,42	41,34	3	100	5/5
BRA1	50/50	39,72	45,57	2	100	5/6
BRA3	50/50	47,00	34,22	8	100	13/14
PAL2	50/50	418,38	410,79	16	100	8/8
TOY	50/50	210,38	226,04	13	100	11/13

Poziom złożoności indukcji języków nawiasowych BRA1 i BRA3 oraz języka AnBn jest porównywalny, choć intuicyjnie zrozumiałe, że najwyższe wartości *nEvals* i *minEvals* ma spośród tych trzech języków język z trzema typami nawiasów BRA3. Dosyć zaskakującym wnioskiem jest porównywalna liczba średnich iteracji poszukujących optymalne zbiory produkcji dla języków AB oraz TOY (odpowiednio 288 i 210). Na pierwszy rzut oka wydawać by się mogło, że znalezienie gramatyki dla prostego w opisie języka AB powinno być znacznie szybsze niż dla, okrojonej co prawda, ale opisującej złożone już jednak struktury języka naturalnego, gramatyki TOY. Wyjaśnienia można szukać jednak nie tyle w intuicyjnym porównywaniu złożoności opisów obydwu języków, ile raczej w rzeczywistej liczbie reguł opisujących języki. W ostatniej kolumnie tab. 8 podano liczbę produkcji gramatyki źródłowej $|P|$ oraz liczbę produkcji gramatyki wyuczonej $|P_i|$. Język AB opisywany jest przez 9 produkcji gramatyki źródłowej, a język TOY przez 11 produkcji. Model GCS znalazł gramatykę dla obydwu języków składającą się, tak w jednym, jak i drugim przypadku, ze zbioru 13 produkcji. Z tego punktu widzenia zatem, złożoność obydwu gramatyk jest porównywalna i może tłumaczyć porównywalny również ich koszt indukcji. Nasuwa się pytanie, dlaczego więc indukcja gramatyki języka BRA3, składającej się aż z 13 źródłowych reguł jest istotnie szybsza od porównywalnych liczbą produkcji języków AB i TOY. Odpowiedź tkwi już w analizie samych produkcji tworzących gramatykę BRA3. Należy bowiem pamiętać, że język ten wymaga aż 6 reguł terminalnych, tworzonych automatycznie przez model podczas pierwszego czytania zdań uczących, a dodatkowe 3 reguły są trywialne i opisują

najprostsze zdania złożone z kolejnych par nawiasów (szczegółowa gramatyka przedstawiona jest w dalszej części rozdziału). Z tego też powodu złożoność poszukiwanego zbioru produkcji języka BRA3 jest podobna do zbioru produkcji języków AB oraz TOY.

Poniżej zamieszczono przykładowy zbiór produkcji wyuczony podczas indukcji języka AB:

1. $S \rightarrow SS$
2. $S \rightarrow OO$
3. $S \rightarrow AJ$
4. $S \rightarrow BA$
5. $S \rightarrow AN$
6. $S \rightarrow JA$
7. $S \rightarrow AB$
8. $S \rightarrow NA$
9. $O \rightarrow AB$
10. $N \rightarrow BS$
11. $J \rightarrow SB$
12. $B \rightarrow b$
13. $A \rightarrow a$.

Ten sam zestaw produkcji w formule nieznormalizowanej ma postać:
$S \rightarrow SS \mid AB \mid BA \mid BSA \mid ASB \mid ABAB \mid ABS \mid SBA$
$B \rightarrow b$
$A \rightarrow a$.

Zestawmy wyindukowany zbiór produkcji ze wzorcowym zbiorem produkcji dla tego języka w PNC:

1. $S \rightarrow SS$
2. $S \rightarrow BF$
3. $S \rightarrow BA$
4. $S \rightarrow AB$
5. $S \rightarrow AD$
6. $F \rightarrow SA$
7. $D \rightarrow SB$
8. $B \rightarrow b$
9. $A \rightarrow a$

oraz postaci nieznormalizowanej:
$S \rightarrow SS \mid AB \mid BA \mid BSA \mid ASB$
$B \rightarrow b$
$A \rightarrow a$.

Porównując obydwie listy produkcji, można zauważyć, że model wyuczył się dodatkowych trzech reguł $S \rightarrow ABAB$, $S \rightarrow ABS$, $S \rightarrow SBA$, lecz owe redundantne reguły w żaden sposób nie zmieniają dokładności gramatyki.

W przypadku języka AnBn model GCS wyindukował zestaw identyczny z zestawem produkcji gramatyki źródłowej, tj.:

1. $S \rightarrow OB$
2. $S \rightarrow AB$
3. $O \rightarrow AS$
4. $B \rightarrow b$
5. $A \rightarrow a$

lub w postaci nieznormalizowanej:
$S \rightarrow ASB \mid AB$

$B \rightarrow b$
$A \rightarrow a$.

Zestaw produkcji dla źródłowej gramatyki języka zbilansowanych nawiasów BRA1 jest następujący (gdzie symbol *a* oznacza lewy nawias, a symbol *b* nawias prawy):

1. $S \rightarrow SS$
2. $S \rightarrow AB$
3. $A \rightarrow AS$
4. $A \rightarrow a$
5. $B \rightarrow b$.

Model GCS wyuczył się natomiast poniższego zbioru produkcji:

1. $S \rightarrow SS$
2. $S \rightarrow AB$
3. $S \rightarrow CB$
4. $C \rightarrow AS$
5. $B \rightarrow b$
6. $A \rightarrow a$.

Można zauważyć, że chociaż lista produkcji wyindukowanych jest o jedną produkcję dłuższa od listy wzorcowej, jednak nie zawiera reguł nadmiarowych. Wynika to z równoważnego zapisu reguły źródłowej $A \rightarrow AS$ poprzez regułę $S \rightarrow ASB$, która jest nieznormalizowaną postacią dwóch reguł $S \rightarrow CB$ oraz $C \rightarrow AS$.

Gramatykę źródłową dla języka BRA3 tworzy poniższy zestaw produkcji:

1. $S \rightarrow AD$
2. $S \rightarrow BE$
3. $S \rightarrow CF$
4. $S \rightarrow SS$
5. $S \rightarrow AS$
6. $C \rightarrow CS$
7. $B \rightarrow BS$
8. $A \rightarrow a$
9. $B \rightarrow b$
10. $C \rightarrow c$
11. $D \rightarrow d$
12. $E \rightarrow e$
13. $F \rightarrow f$,

gdzie trzy typy nawiasów tworzą pary symboli *ad*, *be* i *cf*.

W drodze eksperymentów model GCS wyindukował poniższy zbiór produkcji:

1. $S \rightarrow AD$
2. $S \rightarrow BE$
3. $S \rightarrow CF$
4. $S \rightarrow SS$
5. $S \rightarrow BK$
6. $K \rightarrow SE$
7. $F \rightarrow SF$
8. $D \rightarrow SD$
9. $A \rightarrow a$
10. $B \rightarrow b$
11. $C \rightarrow c$
12. $D \rightarrow d$
13. $E \rightarrow e$
14. $F \rightarrow f$.

Mamy tutaj sytuację niemal identyczną jak w przypadku języka z jednym typem nawiasów. Pojawiły się bowiem również dwie produkcje $S \rightarrow BK$ i $K \rightarrow SE$, które są

równoważne regule źródłowej $B \rightarrow BS$. Model zastąpił także iteracyjne wyprowadzanie lewych nawiasów (reguły źródłowe $A \rightarrow AS$ oraz $C \rightarrow CS$) równoważnym, iteracyjnym wyprowadzaniem prawych nawiasów ($F \rightarrow SF$ oraz $D \rightarrow SD$).

Indukcja gramatyki palindromicznej PAL2 zakończyła się wyuczeniem źródłowego zestawu produkcji, czyli:

1. $S \rightarrow AA$
2. $S \rightarrow BB$
3. $S \rightarrow OA$
4. $S \rightarrow BH$
5. $O \rightarrow \mathrm{AS}$
6. $H \rightarrow SB$
7. $B \rightarrow b$
8. $A \rightarrow a$

lub w postaci nieznormalizowanej

$S \rightarrow AA \mid BB \mid ASA \mid BSB$

$B \rightarrow b$

$A \rightarrow a$.

Na rysunku 50 przedstawiono uśredniony po 50 iteracjach przebieg procesu indukcji palindromu, który to proces zgodnie z wynikami przedstawionymi w tab. 8 miał najwyższy koszt indukcji. Charakterystyczny dla uczenia jest gwałtowny wzrost w pierwszej fazie uczenia produkcji akceptujących zarówno zdania poprawne (linia *positive*), jak i niepoprawne (linia *negative*). Ewolucyjną eksplorację przestrzeni rozwiązań zastępuje z czasem eksploatacja zbioru produkcji, które odrzucają coraz większą liczbę zdań negatywnych.

Ostatnią indukowaną gramatyką była tzw. gramatyka dziecięca, która w wersji źródłowej ma następujący zestaw produkcji:

$S \rightarrow np\ vp$

$np \rightarrow det\ n \mid np\ pp$

$pp \rightarrow prep\ n$

$vp \rightarrow v\ np \mid vp\ pp$.

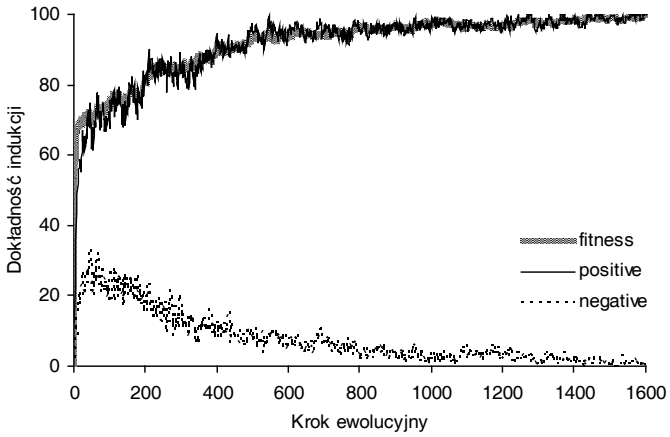
Efektem uczenia modelu był poniższy zestaw produkcji:

1. $S \rightarrow MB$
2. $S \rightarrow ML$
3. $S \rightarrow EB$
4. $S \rightarrow PS$
5. $S \rightarrow AS$
6. $P \rightarrow BC$
7. $M \rightarrow BD$
8. $L \rightarrow AB$
9. $E \rightarrow SC$
10. $A \rightarrow a$
11. $B \rightarrow b$
12. $C \rightarrow c$
13. $D \rightarrow d$,

w którym symbol a oznacza rodzajnik *det*, symbol b rzeczownik n, symbol c przyimek *prep*, a symbol d czasownik v.

Trudno doszukać się analogii pomiędzy zestawem produkcji gramatyki źródłowej a wyindukowanej. Jednak testy generalizacji świadczą o tym, że model znalazł równoważną gramatykę dla tego samego zbioru uczącego. Analizując otrzymaną gramatykę,

można wyciągnąć interesujące wnioski dotyczące samej składni zbioru uczącego. Model pogrupował na przykład ze sobą rzeczownik i przyimek (reguła $P \rightarrow BC$), ale w kolejności odwrotnej do tej, która jest w gramatyce źródłowej ($pp \rightarrow prep\ n$). Rozdzielił niejako wzorcową grupę przyimkową, łącząc w parę rzeczownik z pierwszym słowem następnej frazy. Grupa P rozpoczynać może każde zdanie (reguła $S \rightarrow PS$), podobnie zresztą jak rodzajnik (reguła $S \rightarrow AS$). Nieoczekiwane grupowanie reprezentuje też reguła $M \rightarrow BD$, łącząca bezpośrednio rzeczownik z czasownikiem. W gramatyce wzorcowej rzeczownik i czasownik są składowymi odrębnych grup składniowych. Spodziewaną produkcją jest natomiast $L \rightarrow AB$, która wyprowadza po rodzajniku rzeczownik.



Rys. 50. Indukcja palindromu PAL2
Fig. 50. Induction of palindrome PAL2

Uzyskane wyniki w indukcji języków bezkontekstowych zostaną porównane przede wszystkim rezultatami osiągniętymi w (Bianchi 1996). Jest to uzasadnione, ponieważ ewolucyjny model Bianchiego jest w kilku miejscach podobny do modelu GCS. System Bianchiego oparty został o klasyczną architekturę systemu klasyfikującego z uproszczoną wersją algorytmu kubełkowego, a rozbiór zdań wykonywany był – podobnie jak w modelu GCS – z zastosowaniem algorytmu CYK. Bianchi indukował języki BRA1, BRA3 oraz TOY. Zbiór uczący składał się z 50 przykładów pozytywnych i 50 negatywnych, wyniki każdego eksperymentu uśredniano po 10 niezależnych uruchomieniach

Tabela 9. Porównanie kosztów indukcji *nEvals* języków bezkontekstowych w modelu GCS i (Bianchi 1996)

Język	LCS	GA	GCS
BRA1	53	500	40
BRA3	319	9500	47
TOY	3000	–	210

programu. Poza własnym modelem, opartym o uczący się system klasyfikujący (LCS), Bianchi zbadał również efektywność uczenia klasycznego algorytmu genetycznego (GA), w którym pojedynczy chromosom kodował całą gramatykę. Porównanie rezultatów modelu GCS oraz LCS i GA zawarto w tab. 9. W pracy (Bianchi 1996) nie indukowano gramatyki dziecięcej z użyciem algorytmu genetycznego.

Model GCS, podobnie jak model LCS Bianchiego, okazał się efektywniejszy od implementacji prostego algorytmu genetycznego. Porównując ze sobą obydwa modele oparte o uczące się systemy klasyfikujące, można stwierdzić, że w przypadku języka BRA1 koszt indukcji obydwu modeli jest porównywalny, choć lepszą wartość osiąga GCS (odpowiednio 53 kroki dla LCS i 40 dla GCS). Zdecydowanie wyższą efektywnością wykazuje się już model GCS dla kolejnego języka zbilansowanych nawiasów BRA3, natomiast w przypadku języka TOY koszt indukcji jest aż 14-krotnie niższy (3000 kroków dla LCS i 210 kroków dla GCS).

W tabeli 10 zawarto porównanie modelu GCS z ewolucyjnymi modelami Lankhorsta. Lankhorst (1995) indukował niedeterministyczny automat ze stosem, używając dwóch metod reprezentacji zadania: binarnej (Lan1) oraz całkowitoliczbowej (Lan2). Funkcja oceny uwzględniała poprawność klasyfikacji zdań uczących, częściowo poprawne analizowane podciągi oraz stopień wypełnienia stosu. Lankhorst zastosował losowo generowane zbiory uczące składające się ze 100 zdań pozytywnych oraz 100 negatywnych, o maksymalnej długości 30 symboli. Również stuelementowe losowe zbiory zdań pozytywnych

Tabela 10. Porównanie rezultatów uzyskanych przez model GCS i (Lankhorst 1995) dla języków bezkontekstowych

Język	*nEvals*			*nGen*		
	Lan1	Lan2	GCS	Lan1	Lan2	GCS
AB	398	629	288	96,50	96,50	100
BRA1	18	19	40	97	90,50	100
PAL2	727	704	418	79	82	100
TOY	328	235	210	98,50	98,50	100

i negatywnych służyły do testów generalizacji. Każdy eksperyment był 10-krotnie powtarzany. Porównanie modelu GCS z metodami Lankhorsta można rozpocząć od analizy dokładności generalizacji algorytmów. Widoczny jest brak 100% generalizacji obydwu metod Lankhorsta, najniższy poziom obydwie metody osiągnęły podczas indukcji palindromu, co potwierdza dużą złożoność procesu uczenia tego języka. Model GCS dla każdego języka indukował gramatykę bezbłędnie klasyfikującą wszystkie zdania z obszernych zbiorów testowych. Również porównanie kosztów indukcji wypada korzystniej dla modelu GCS. W 3 przypadkach na 4 model GCS indukował gramatykę w mniejszej średnio liczbie kroków niż automaty Lankhorsta, choć należy zauważyć, że całkowitoliczbowa reprezentacja automatu osiągnęła zbliżony wynik podczas uczenia gramatyki TOY (metoda Lan2 – 235 kroków, model GCS – 210 kro-

ków). Obydwie metody opisane w (Lankhorst 1995) osiągnęły lepszy wynik podczas indukcji prostego języka nawiasowego BRA1, znajdując rozwiązanie średnio w 18–19 krokach w porównaniu do 40 kroków modelu GCS. Należy jednak podkreślić, że żaden z indukowanych automatów nie potrafił poprawnie sklasyfikować wszystkich zdań ze zbioru testowego.

Wspomniano wcześniej, że w chwili obecnej w literaturze przedmiotu przeważa stosowanie gramatyk stochastycznych w indukcji języków bezkontekstowych. Najpoważniejszym argumentem przemawiającym za stosowaniem metod probabilistycznych jest możliwość uczenia jedynie na podstawie przykładów pozytywnych, które w realnych zastosowaniach są bardziej dostępne od przykładów negatywnych. Keller i Lutz badali możliwości zastosowania algorytmów genetycznych w indukcji stochastycznych gramatyk bezkontekstowych reprezentowanych w postaci normalnej Chomsky'ego (Keller i Lutz 1997, 2005). W ich podejściu ewolucji podlegała nie tyle cała gramatyka, co jedynie zbiór parametrów opisujących poszczególne produkcje, w tym prawdopodobieństwa produkcji. Proces uczenia zaczynał się od stworzenia gramatyki pokrywającej zbiór uczący, ale zawierającej również produkcje generujące zdania spoza tego zbioru. Następnie w drodze ewolucji poszukiwano optymalnego zbioru parametrów wejściowej gramatyki. Przyjęto, że zbiór optymalny pozwala wygenerować gramatykę zgodną z danymi uczącymi z zadanym, wysokim prawdopodobieństwem (tj. powyżej 93%). Oznacza to, że wynikowa gramatyka może mieć nieprawidłowe produkcje z prawdopodobieństwem większym od zera, ale na tyle małym, że można je pominąć (tzw. *near-miss grammar*). W tabeli 11 zestawiono ze sobą wartość estymatora *nSuccess* uzyskanego podczas indukcji tych samych języków bezkontekstowych przez model GCS, podejście Kellera i Lutza (KL) oraz podejście, w którym zbiór parametrów gramatyki stochastycznej jest optymalizowany za pomocą algorytmu *inside-outside* (IO). Z porównania metod wynika, że zastosowanie algorytmu genetycznego w optymalizacji zbioru produkcji gramatyki stochastycznej daje lepsze wyniki od powszechnie stosowanego algorytmu IO. Dla języków AnBn oraz BRA1 w każdej iteracji eksperymentu algorytm KL uzyskał maksymalną wartość estymatora. Model GCS w indukcji wszystkich języków osiągnął 100% zgodność ze zbiorem uczącym (ale również i zbiorem testowym – patrz tab. 8). Należy jednak pamiętać, że takie porównania są utrudnione, ze względu na różnice w definicji samej gramatyki oraz w konstrukcji zbiorów uczących.

Tabela 11. Porównanie wartości estymatora *nSuccess* dla języków bezkontekstowych indukowanych przez model GCS oraz modele stochastyczne

Język	IO	KL	GCS
AB	82	95	100
AnBn	94	100	100
BRA1	92	100	100
PAL2	42	85	100

5.5. Indukcja języka naturalnego

Eksperymenty indukcji gramatyki języka naturalnego zostały przeprowadzone na zbiorze 9 korpusów językowych zastosowanych w (Aycinena i in. 2003). Ze względu na czasochłonność obliczeń, związaną przede wszystkim z rozmiarem zbiorów uczących, uczenie gramatyki każdego korpusu przebiegało w 10 niezależnych iteracjach programu ($n_{run} = 10$), z których każda nie przekraczała 1000 kroków ewolucyjnych ($n_{max} = 1000$)[118]. Pozostałe parametry eksperymentów, a zatem zmienne decyzyjne oraz parametry ciagłe, były identyczne z parametrami zastosowanymi podczas indukcji języków formalnych.

Zgodnie z oczekiwaniami, w żadnym eksperymencie model nie wyuczył się gramatyki zdolnej w 100% do prawidłowej klasyfikacji wszystkich zdań ze zbioru uczącego. W tabeli 12 podano wyniki indukcji dla poszczególnych korpusów językowych.

Pierwsze trzy kolumny zawierają syntetyczne estymatory określające proces uczenia danego korpusu – f_{avg} oznacza $fitness_{avg}$, p_{avg} $positive_{avg}$, n_{avg} $negative_{avg}$. W kolejnych czterech kolumnach zapisano kompetencję gramatyki: ogólną (f_{max}), pozytywną (p), negatywną (n), dla której uzyskano najlepszą wartość $fitness_{max}$ oraz liczbę kroków ewolucyjnych (*Evals*), potrzebnych by model wyuczył się tej gramatyki. Ostatnie trzy kolumny tabeli opisują wartości estymatorów uzyskane przez model podczas testów generalizacji: *nGen* oznaczone symbolem *nG*, $nGen_{pos}$ (nG_{pos}) i $nGen_{neg}$ (nG_{neg}). Estymator $fitness_{avg}$ przyjmuje wartości pomiędzy 64,8% dla korpusu brown_c a 83,5% dla korpusu children.

Tabela 12. Wyniki indukcji korpusów językowych uzyskane przez model GCS

Korpus	Zbiór uczący							Zbiór testowy		
	f_{avg}	p_{avg}	n_{avg}	Iteracja, w której uzyskano $fitness_{max}$						
				f_{max}	p	n	*Evals*	*nG*	nG_{pos}	nG_{neg}
children	83,5	92,1	25,1	93,2	98,8	12,5	9	92,2	97,2	12,8
wizard	80,2	69,7	9,3	94,6	99,3	10,2	32	94,2	99,5	11,0
alice	75,1	63,8	13,6	89,5	96,8	17,9	81	89,5	97,2	18,1
tom	77,0	90,6	35,3	86,3	98,4	25,9	3	86,1	98,3	26,0
brown_a	82,4	81,6	16,8	93,8	98,3	11,6	45	93,8	98,1	10,5
brown_b	82,0	75,6	11,6	94,6	99,3	10,2	506	94,2	99,0	10,7
brown_c	64,8	34,5	4,8	92,5	96,7	11,7	592	91,8	95,8	12,2
brown_d	69,9	49,0	9,1	91,6	97,1	13,8	18	91,0	95,1	13,2
brown_e	70,7	54,1	12,5	89,5	93,4	14,5	38	90,0	94,6	14.6

Można zadać pytanie, czy wydłużenie okresu uczenia nie przyniosłoby wyższych wartości estymatora. Na rysunku 51 zilustrowano krzywą zbieżności estymatora *fit-*

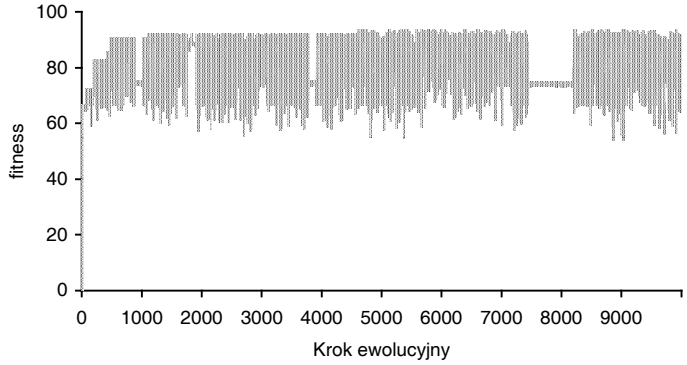
[118] Jeden eksperyment indukcji korpusu językowego, składający się z 10 iteracji, trwał średnio około 24 h (Pentium 2,81 GHz, 2,00 GB RAM).

ness podczas eksperymentu tylko z jedną iteracją ($n_{\text{run}} = 1$), ale z wydłużonym dziesięciokrotnie okresem uczenia ($n_{\max} = 10\,000$). Eksperyment nie przyniósł żadych istotnych zmian wartości badanych estymatorów ($fitness_{\text{avg}} = 82{,}0\%$, $positive_{\text{avg}} = 92{,}8\%$, $negative_{\text{avg}} = 28{,}8\%$, $fitness_{\max} = 93{,}3\%$), krzywa estymatora *fitness* oscyluje wokół wartości ok. 74% z amplitudą ok. 15%.

Poniżej podano wyewoluowaną gramatykę:

1. $S \rightarrow SE$
2. $S \rightarrow SS$
3. $S \rightarrow CA$
4. $S \rightarrow DS$
5. $S \rightarrow BS$
6. $S \rightarrow SK$
7. $S \rightarrow MF$
8. $S \rightarrow SB$
9. $S \rightarrow ES$
10. $S \rightarrow FS$
11. $S \rightarrow GR$
12. $R \rightarrow SM$
13. $M \rightarrow AB$
14. $K \rightarrow AD$
15. $G \rightarrow AF$
16. $E \rightarrow MM$
17. $C \rightarrow BD$
18. $C \rightarrow BF$
19. $S \rightarrow a$
20. $S \rightarrow c$
21. $G \rightarrow g$
22. $F \rightarrow f$
23. $E \rightarrow e$
24. $D \rightarrow d$
25. $C \rightarrow c$
26. $B \rightarrow b$
27. $A \rightarrow a$.

Dosyć oczywistą grupę *przymiotnik rzeczownik* tworzy reguła 3 czy też reguła 13 *rzeczownik czasownik*. Model odnalazł w zbiorze zdań uczących również często występujące w języku angielskim bigramy, takie jak: *rzeczownik przysłówek* (reguła 14), *rzeczownik spójnik* (reguła 15), *czasownik przysłówek* (reguła 17) czy *czasownik spójnik* (reguła 18). Zdanie może rozpoczynać się od rodzajnika (reguła 10) albo też dodanie rodzajnika na początek każdego zdania zachowuje jego poprawność. Zdecydowana większość reguł bezkontekstowych rozpoczyna się od symbolu startowego, co sugeruje dużą ogólność tych reguł. Z jednej strony pozwala to na oszczędne zapisanie całej gramatyki, z drugiej jednak – taka uniwersalność umożliwia parsowanie również zdań nienależących do języka.



Rys. 51. Indukcja korpusu children ($n_{\mathbf{run}} = 1$, $n_{\max} = 10\,000$)
Fig. 51. Induction of children corpus ($n_{\mathbf{run}} = 1$, $n_{\max} = 10\,000$)

Maksymalne wartości kompetencji ogólnej $fitness_{\max}$ uzyskane podczas indukcji korpusów są oczywiście dużo wyższe od wartości estymatora $fitness_{\text{avg}}$ i mieszczą się

w przedziale od 86,3% dla korpusu tom do 94,6% dla korpusów wizard oraz brown_b. Zwracają uwagę wysokie wartości kompetencji pozytywnej gramatyk, które uzyskały najwyższe wartości kompetencji ogólnej i stosunkowo wysokie kompetencji negatywnej tych samych gramatyk. Najwyższą wartość *negative* zanotował korpus tom. Prawie 26% przykładów negatywnych zostało sparsowanych przez gramatykę. Wyniki testów generalizacji nie odbiegają znacząco od wartości estymatorów *fitness*, *postive* oraz *negative* uzyskanych przez najlepsze gramatyki poszczególnych korpusów. Świadczy to nie tyle o własnościach gramatyk, co raczej jednorodnym pochodzeniu zbiorów uczących i testowych. Charakterystyczną cechą indukcji korpusów jest w większości wypadków niewielka liczba kroków ewolucyjnych, w których model wyindukował najlepsze gramatyki. Dla 7 korpusów liczba kroków nie przekracza 100, a tylko w dwóch przypadkach (korpusy brown_b oraz brown_c) nieznacznie przekracza liczbę 500 kroków.

Interesujące jest porównanie wyników indukcji modelu GCS z podejściem zaproponowanym przez doktorantów ze Stanford (Aycinena i in. 2003), oznaczonym dalej skrótem AKM od pierwszych liter nazwisk. W tabeli 13 zebrano parametry obydwu podejść, które można ze sobą zestawić. Dwie pierwsze kolumny zawierają wartości najlepszej kompetencji ogólnej gramatyki, jaką udało się podczas uczenia uzyskać dla poszczególnych korpusów, odpowiednio dla modelu GCS i podejścia AKM[119]. Następne kolumny w parach model GCS – podejście AKM zestawiają ze sobą wartości estymatorów *positive*, *negative* oraz liczbę kroków potrzebnych do wyuczenia się gramatyki o najlepszej wartości kompetencji ogólnej (*Evals*).

Tabela 13. Porównanie rezultatów uzyskanych przez model GCS i (Aycinena i in. 2003) dla korpusów językowych

Korpus	$fitness_{max}$		*positive*		*negative*		*Evals*	
	GCS	AKM	GCS	AKM	GCS	AKM	GCS	AKM
children	93,2	93,1	98,8	91,8	12,5	5,7	9	200 000
wizard	94,6	90,2	99,3	89,5	10,2	9,2	32	200 000
alice	89,5	92,1	96,8	92,5	17,9	8,4	81	200 000
tom	86,3	92,1	98,4	92,7	25,9	8,6	3	200 000
brown_a	93,8	94,0	98,3	94,1	11,6	6,1	45	48 500
brown_b	94,6	94,0	99,3	94,7	10,2	6,7	506	200 000
brown_c	92,5	87,9	96,7	80,5	11,7	4,7	592	15 500
brown_d	91,6	91,3	97,1	88,2	13,8	5,6	18	45 000
brown_e	89,5	94	93,4	93,9	14,5	5,9	38	122 000

W przypadku 5 korpusów model GCS wyindukował gramatykę o wyższej wartości $fitness_{max}$, dla korpusu brown_a wartość ta jest tylko nieznacznie niższa (93,8% dla modelu GCS, 94% dla AKM), a w pozostałych 3 przypadkach wartość estymatora jest

[119] W pracy (Aycinena i in. 2003) nie podano tej wartości, ale można ją wyliczyć, znając liczebność zbiorów przykładów pozytywnych i negatywnych oraz wartości estymatorów *positive* i *negative*.

niższa, lecz różnica nie przekracza 5%. Wartości kompetencji pozytywnej są w 8 przypadkach zdecydowanie wyższe dla modelu GCS (różnice wahają się w przedziale od 4,2% do 16,2%), a dla korpusu brown_e podejście AKM uzyskało lepszy wynik o 0,5%. Zdecydowanie najgorzej dla modelu GCS wypada porównanie wartości kompetencji negatywnej – model dla każdego korpusu uzyskał wyższe wartości tego estymatora, a różnice są w przedziale od 1% (korpus *wizard*) do 17,3% (korpus tom). Oznacza to, że podczas indukcji gramatyki model GCS stworzył w kilku wypadkach (dla 5 korpusów różnice nie przekraczają 7%) zbyt uniwersalne produkcje w porównaniu do podejścia AKM, które parsują również część zdań negatywnych. Ostatnim parametrem dającym się porównać jest liczba kroków ewolucyjnych, w których obydwa podejścia znalazły najlepsze rozwiązania. Model GCS aż w 6 wypadkach nie przekroczył liczby 50 kroków, w kolejnym liczby 100 kroków, a dwie najdłuższe indukcje trwały tylko nieco ponad 500 kroków (czyli nieco ponad 1 h). Podejście ze Stanford w najlepszym wypadku potrzebowało 15 500 kroków, a aż dla 5 korpusów 200 000 kroków i, jak podają autorzy, około 60 h obliczeń. Model GCS okazał się zatem nieporównywalnie efektywniejszy, znajdując w większości wypadków gramatyki o wyższych wartościach kompetencji ogólnej i pozytywnej.

5.6. Badanie symulacyjne

O ile nie zostało to zaznaczone, podczas wszystkich przeprowadzonych eksperymentów uczenie gramatyk języka TOY przebiegało w 50 niezależnych iteracjach programu ($n_{\text{run}} = 50$), z których każda nie przekraczała 5000 kroków ewolucyjnych ($n_{\max} = 5000$). Zestaw uczący był zrównoważony i zawierał 20 przykładów zdań pozytywnych i 20 przykładów zdań negatywnych.

Parametry modelu, niemodyfikowane w danej serii eksperymentów, miały następujące domyślne wartości:

Zmienne decyzyjne:

f_{GA} = tak – zmienna zezwalająca na uruchomienie algorytmu genetycznego,

f_{GA}^{1} = ruletka – rodzaj zastosowanej selekcji podczas wyboru pierwszej produkcji,

f_{GA}^{2} = ruletka – rodzaj zastosowanej selekcji podczas wyboru drugiej produkcji,

f_{kor} = nie – zmienna zezwalająca na uruchomienie korekcji gramatyki,

f_{cs} = tak – zmienna zezwalająca na uruchomienie operatora pokrycia startowego,

f_{cp} = tak – zmienna zezwalająca na uruchomienie operatora pokrycia pełnego,

f_{cu} = nie – zmienna zezwalająca na uruchomienie operatora pokrycia uniwersalnego

Parametry ciągłe

$n_{\max} = 5000$ – maksymalna liczba kroków ewolucyjnych,

$n_{\text{run}} = 50$ – liczba iteracji,

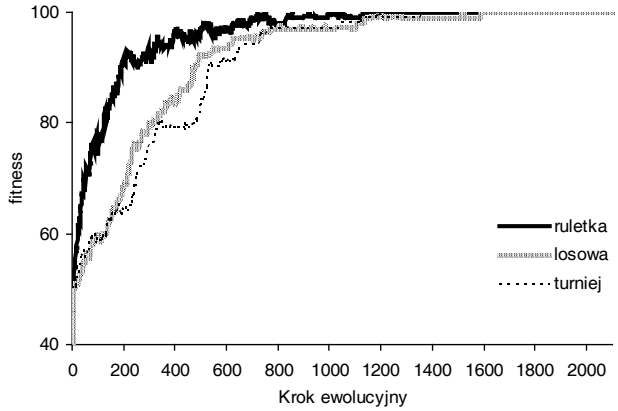
$n_P = 40$ – rozmiar populacji,
$n_{\text{start}} = 30$ – liczba początkowych produkcji nieterminalnych,
$n_N = 19$ – liczba symboli nieterminalnych,
$p_k = 0{,}2$ – prawdopodobieństwo krzyżowania dla algorytmu genetycznego,
$p_m = 0{,}8$ – prawdopodobieństwo mutacji dla algorytmu genetycznego,
$p_i = 0$ – prawdopodobieństwo inwersji dla algorytmu genetycznego,
$p_a = 0$ – prawdopodobieństwo zastosowania operatora pokrycia agresywnego,
$cf = 18$ – współczynnik ścisku,
$cs = 3$ – podpopulacja ścisku,
$ba = 1$ – kwota bazowa,
$raf = 0{,}5$ – współczynnik zmniejszania kwoty bazowej,
$n_{\text{elit}} = 0$ – wielkość elity,
$w_p = 1$ – waga rozbioru zdania poprawnego,
$w_n = 2$ – waga rozbioru zdania niepoprawnego,
$w_c = 1$ – waga funkcji klasycznej klasyfikatora,
$w_f = 0$ – waga funkcji płodności klasyfikatora,
$f_0 = 0{,}5$ – miara użyteczności klasyfikatora niebiorącego udziału w parsowaniu.

5.6.1. Metody selekcji

Seria przeprowadzonych badań miała dać odpowiedź na pytanie, która metoda selekcji spośród ruletki, metody losowej oraz turniejowej jest najefektywniejsza w modelu GCS. Przyjęto parametry modelu identyczne z parametrami zastosowanymi podczas indukcji języków formalnych. Metoda selekcji turniejowa była analizowana dla rozmiaru podpopulacji turniejowej $ts = 4$.

Na rysunku 52 zilustrowano przebieg wartości estymatora *fitness* w czasie indukcji gramatyki dla selekcji ruletkowej, losowej i turniejowej. Wszystkie metody selekcji prowadzą do optymalnego rozwiązania, choć istnieją niewielkie różnice w szybkości zbieżności pomiędzy metodami.

Bardzo podobny przebieg i nachylenie krzywej wykazują metody turniejowa i losowa. Należy jednak pamiętać, że w modelu GCS podczas pojedynczego kroku ewolucyjnego mogą się zmienić jedynie dwie produkcje (patrz podrozdz. 4.11.6), a zatem wynik selekcji obydwu metod jest podobny. Metoda losowa wybiera z całej populacji losowy klasyfikator, który następnie jest przedmiotem operatorów genetycznych oraz ścisku. W metodzie turniejowej wybierany jest, co prawda, klasyfikator o najwyższym przystosowaniu, ale wybierany jest z losowo dobranej podpopulacji o liczności $ts = 4$. W związku z tym zarówno w pierwszej, jak i w drugiej metodzie istotną rolę odgrywa czynnik losowego wyboru. Selekcja ruletkowa wykazuje się najbardziej stromą krzywą zbieżności, a obserwacje potwierdzają wartości syntetycznych estymatorów (tab. 14).



Rys. 52. Porównanie wyników działania różnych metod selekcji
Fig. 52. Comparison of results of different selection methods

Tabela 14. Porównanie efektywności metod selekcji

selekcja	$fitness_{avg}$	$positive_{avg}$	*nEvals*
ruletka	98,5	96,9	209,38
losowa	96,7	93,4	342,90
turniej	96,3	92,5	384,86

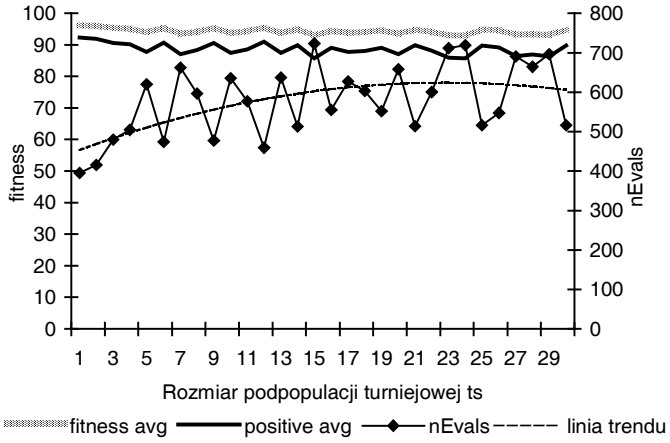
Ruletka ma najwyższą ze wszystkich badanych metod selekcji wartość estymatora $fitness_{avg}$, który jest uśrednioną po liczbie kroków ewolucyjnych kompetencją ogólną gramatyki. Im wyższa wartość tego estymatora, tym szybciej opisywany przez niego proces indukcji dochodzi do gramatyk osiągających wysoką dokładność w klasyfikacji zdań uczących. Podobną interpretację ma estymator $positive_{avg}$, liczący średnią miarę kompetencji pozytywnej gramatyki. Zarówno wartość estymatora $positive_{avg}$, jak i *nEvals* jest najwyższa dla selekcji ruletkowej, która w każdym kroku ewolucyjnym wybiera do genetycznej transformacji produkcje w taki sposób, że prawdopodobieństwo wyboru produkcji jest wprost proporcjonalne do jej przystosowania.

Proces selekcji produkcji to dopiero pierwszy etap przetwarzania ewolucyjnego w modelu GCS, po którym następują równie istotne etapy zastosowania operatorów genetycznych oraz wymiany produkcji z wykorzystaniem ścisku (patrz rys. 20). Pełna odpowiedź na pytanie dotyczące efektywności poszczególnych metod selekcji nie jest zatem możliwa bez przebadania wpływu pozostałych mechanizmów ewolucyjnych modelu.

5.6.2. Selekcja turniejowa

Analiza wpływu selekcji turniejowej na indukcję gramatyki powinna uwzględnić możliwe różne wartości rozmiaru podpopulacji turniejowej *ts*. Na rysunku 53 przed-

stawiono dokładność i koszt procesu indukcji w zależności od rozmiaru podpopulacji *ts*. Ponieważ eksperymenty prowadzono przy liczbie początkowych produkcji nieterminalnych n_{start} = 30, zmieniano parametr *ts* w przedziale [1, 30]. Wykres krzywej kosztu indukcji został uzupełniony o wielomianową linię trendu. Eksperymenty przeprowadzono przy domyślnych wartościach paramaterów modelu stosowanych również w indukcji języków formalnych (a zatem dla wartości współczynnika ścisku *cf* = 18 oraz podpopulacji ścisku *cs* = 3).

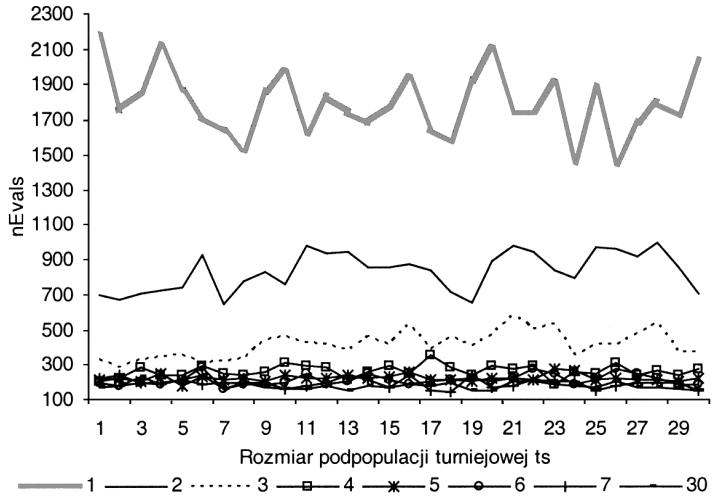


Rys. 53. Wpływ rozmiaru podpopulacji *ts* na dokładność i koszt indukcji (*cf* = 18, *cs* = 3)
Fig. 53. Influence of tournament subpopulation *ts* on accuracy and cost of induction (*cf* = 18, *cs* = 3)

Uzyskane krzywe zbieżności mogą być na pierwszy rzut zaskakujące, gdyż przeczą przyjętym w literaturze przedmiotu twierdzeniom, że nacisk selektywny jest wprost proporcjonalny do rozmiaru podpopulacji turniejowej, również w uczących się systemach klasyfikujących (Kharbat i in. 2005). Krzywa kosztu indukcji *nEvals* wyraźnie rośnie wraz ze wzrostem *ts*, co dobitnie ilustruje krzywa trendu. Model GCS potrzebuje zatem coraz większej liczby kroków ewolucyjnych, aby znaleźć rozwiązanie. Intuicyjnie natomiast można by oczekiwać odwrotnego zjawiska, czyli coraz szybszego znajdywania gramatyki wraz ze zwiększaniem podpopulacji turniejowej. Wzrost podpopulacji *ts* równoznaczny jest bowiem z selekcją najlepszej produkcji w każdorazowo większej populacji produkcji, co niewątpliwie zwiększa prawdopodobieństwo wyboru najlepszej produkcji w całej populacji. Również krzywe dokładności indukcji, tj. $fitness_{avg}$ oraz $positive_{avg}$, dosyć nieoczekiwanie nieznacznie opadają razem ze wzrostem liczności podpopulacji turniejowej.

Odpowiedzi na powyższą zagadkę należy poszukiwać w znacznie szerzej przeprowadzonych badanich modelu, uwzględniających cały proces wymiany produkcji w populacji. Selekcja turniejowa (ale również każda inna) rozpoczyna ten proces poprzez wyselekcjonowanie osobników (produkcji) przeznaczonych do ewolucyjnej

modyfikacji. Proces wymiany kończy natomiast procedura ścisku, która wyszukuje produkcje do zastąpienia. Najprawdopodobniej więc za takie zachowanie ewolucji w modelu odpowiada właśnie schemat ze ściskiem, który wprowadza nową produkcję do populacji w miejsce najbardziej podobnej. Wyniki badań powiązania ścisku z selekcją turniejową przedstawiono w tab. 15 i 16 oraz na odpowiadających tabelom rys. 54 i 55. Każda wartość w tabeli i odpowiadający jej punkt na wykresie jest odpowiednią miarą liczoną na podstawie 250 000 liczb (50 niezależnych przebiegów pomnożone przez 5000 kroków ewolucyjnych w każdej iteracji). Aby zmniejszyć liczbę badanych parametrów, założono, że podczas wszystkich indukcji współczynnik ścisku *cf* = 1. Zbadano przebieg procesu indukcji dla podpopulacji *cs* = 1, 2, 3, 4, 5, 6, 7, 30.



Rys. 54. Wpływ rozmiaru podpopulacji *ts* na koszt indukcji (*cf* = 1, *cs* jest zmienną)
Fig. 54. Influence of tournament subpopulation *ts* on induction cost (*cf* = 1, *cs* is variable)

W tabeli 15 zawarto wartości kosztu indukcji *nEvals* w zależności od rozmiaru podpopulacji *cs* i rozmiaru podpopulacji turniejowej *ts* – rys. 54 to graficzna prezentacja tych danych. Charakterystyczne dla modelu są wyraźnie wyższe średnie koszty indukcji dla *cs* = 1, 2, 3. W przypadku jednoelementowej podpopulacji ścisku uśrednione wartości estymatora *nEvals* są nawet stukrotnie większe (!) od tych otrzymanych dla wyższych wartości *cs* (począwszy od wartości *cs* = 4) i to niezależnie od rozmiaru podpopulacji turniejowej *ts*. Również charakterystyczne są krzywe zbieżności dla $cs \leq 3$. Dla wartości *cs* = 1 im wyższa wartość podpopulacji turniejowej *ts*, tym średni koszt indukcji jest niższy, co pozostaje zresztą zgodne z oczekiwanym kierunkiem działania nacisku selektywnego w selekcji turniejowej. Ścisk z parametrami *cf* = 1 oraz *cs* = 1 jest równoznaczny z losowym wyborem produkcji do zastąpienia. Zupełnie inną tendencję mają krzywe kosztów indukcji dla rozmiaru podpopulacji ścisku równej 2 oraz 3. Zwiększanie rozmiaru podpopulacji turniejowej powoduje

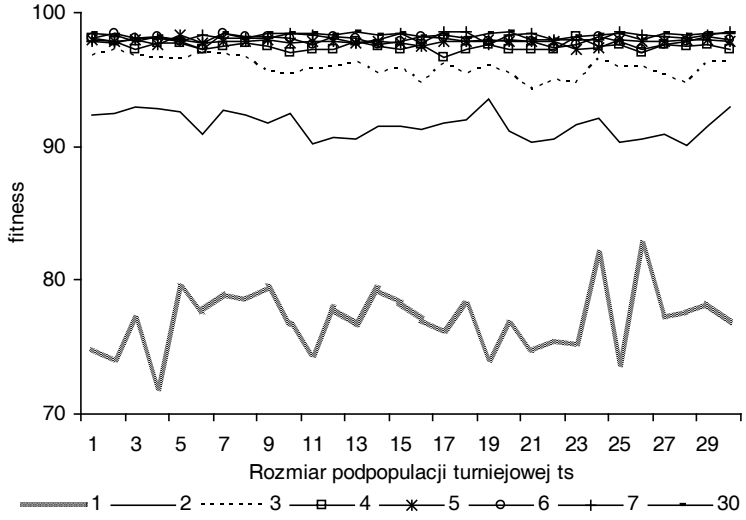
zmniejszanie nacisku selektywnego i zauważalne zwiększanie średniej liczby kroków ewolucyjnych koniecznych do wyewoluowania poprawnej gramatyki. Właśnie to zjawisko zostało zaobserwowane podczas badania wpływu wartości *ts* na indukcję przy domyślnych ustawieniach (patrz rys. 53). Zgodnie z przewidywaniami zatem, wzrost kosztów indukcji proporcjonalny do rozmiaru podpopulacji turniejowej jest wynikiem nie tyle działania selekcji turniejowej, co współdziałającego z nią mechanizmu ścisku.

Tabela 15. Wartości kosztu indukcji dla różnych rozmiarów podpopulacji turniejowej *ts* i ścisku *cs*

Rozmiar *ts*	Koszt indukcji *nEvals* w zależności od rozmiar podpopulacji ścisku *cs*							
	1	2	3	4	5	6	7	30
1	2175,68	695,59	331,92	208,44	217,16	198,36	221,66	170,1
2	1758,19	670,59	283,1	224,56	219,08	175,98	239,34	181,84
3	1860,53	705,3	328,48	283,38	207,14	198,7	198	229
4	2126,58	723,38	344,6	240,88	250,42	184,9	197,42	189,24
5	1872,36	746,04	357,3	244,2	182,32	217,54	213,8	211,82
6	1714,86	923,66	310,94	290,36	236,36	284,9	183,86	237,88
7	1639,14	644,98	316,6	253,3	216,04	161,56	196,9	162,74
8	1523,46	780,94	338,26	237,4	223,84	191,14	200,96	195,54
9	1863,55	827,6	445,58	259,38	207,04	192,02	194,66	167,5
10	1974,64	760,44	464,56	310,2	240,6	199,98	163,62	158,88
11	1625,64	984,14	427,66	289,7	220,58	249,62	180,26	162,4
12	1836,11	939,14	415,14	288,8	227,44	187,2	199,76	180,02
13	1747,78	944,7	378,62	214,9	240,32	202,18	239,68	150,1
14	1680,41	854,28	462,48	257,08	220,3	246,58	212,9	180,34
15	1780,48	852,78	416,14	289,7	236	216,18	171,08	172,22
16	1935,18	878,1	531,42	251,92	260,7	183,62	237,86	184
17	1648,26	838,06	388,94	350,92	217,82	204,14	152,62	182,32
18	1578,63	717,82	458,16	280,66	216,76	215,24	146,7	195,88
19	1918,18	652,36	411,66	243,86	204,28	227,12	245,18	156,48
20	2109,02	890,2	467,96	292,38	221,34	207,42	173,4	154,58
21	1741,47	983,8	582,96	273,22	223,98	219,42	175	203,04
22	1744,92	943,48	505,42	290,4	218,18	274,18	212,84	209,94
23	1916,07	843,48	528,54	186,74	274,86	248,5	203,78	187,92
24	1457	794,28	356,24	255,54	269,52	185,6	206,54	182,74
25	1889,71	973,1	418,4	250,24	210,78	217,12	148,54	170,66
26	1449,65	960,7	413,86	312,78	220,72	279,24	180,66	208,18
27	1681,78	921,94	475,94	245,18	218,26	252,86	194,84	173,52
28	1796,4	999,2	537,92	266,38	211,68	223,04	199	173,4
29	1728,07	855,86	374	245,08	213,45	186,5	200	160,28
30	2033,33	709,6	371,4	273,36	212	185,3	151,1	156,62

Wnioski te potwierdzają krzywe kosztów indukcji dla wartości podpopulacji ścisku większej od 3. Zgodnie z danymi przedstawionymi w tab. 15, a pośrednio również

na rys. 54, koszty indukcji dla tych wartości są już niskie (rzędu 200 kroków) i porównywalne w całym zakresie zmian zarówno parametru *ts*, jak i *cs* większego od 3. Niższe wartości estymatora *nEvals* posiadają wykresy dla wyższych wartości *cs*, choć różnice przestają być praktycznie zauważalne dla $cs \geq 6$. Podpopulacja 6 losowo wybranych produkcji (co stanowi 1/5 całej populacji produkcji nieterminalnych) zapewnia już taki proces zastępowania produkcji, który w efekcie prowadzi do efektywnego uczenia. Na tempo uczenia nie ma już praktycznie wpływu dalsze zwiększanie podpopulacji ścisku *cs*, ale również podpopulacji turniejowej *ts*.



Rys. 55. Wpływ rozmiaru podpopulacji *ts* na dokładność indukcji (*cf* = 1, *cs* jest zmienną)
Fig. 55. Influence of tournament subpopulation *ts* on induction accuracy (*cf* = 1, *cs* is variable)

Analizę kosztów indukcji należy uzupełnić o zbadanie wpływu parametrów ścisku i selekcji turniejowej na dokładność indukcji reprezentowaną przez estymator $fitness_{avg}$. W tabeli 16 i na rys. 55 zaprezentowano wartości dokładności indukcji w zależności od parametrów *cs* oraz *ts*. Uzyskane wyniki potwierdzają spostrzeżenia poczynione podczas badania kosztów indukcji. Parametr *cs* o wartości 4 i wyższej powoduje, że w całym zakresie zmienności rozmiaru podpopulacji turniejowej *ts* oraz *cs* proces indukcji gramatyki uzyskuje wysoką wartość uśrednionej kompetencji ogólnej, a powyżej wartości 5 różnice są praktycznie znikome, a wartość parametru przekracza 98%. Zdecydowanie najniższą wartość przystosowania $fitness_{avg}$ uzyskuje gramatyka dla wartości *cs* = 1 (rzędu siedemdziesięciu kilku procent), z tym że wartość ta rośnie wraz ze wzrostem parametru *ts*. Wyższe wartości estymatora uzyskuje indukcja dla podpopulacji ścisku równej 2 oraz 3 (około 91% oraz odpowiednio 95%), zwiększanie podpopulacji turniejowej powoduje niewielki spadek wartości estymatora.

Tabela 16. Wartości dokładności indukcji dla różnych rozmiarów podpopulacji turniejowej *ts* i ścisku *cs*

Rozmiar *ts*	Dokładność indukcji $fitness_{avg}$ w zależności od rozmiaru podpopulacji ścisku *cs*							
	1	2	3	4	5	6	7	30
1	74,9	92,3	96,8	98,1	98	98,1	97,9	98,4
2	74,1	92,5	97,3	97,9	97,9	98,4	97,7	98,3
3	77,1	93	96,8	97,3	98	98,1	98,1	97,8
4	71,9	92,8	96,6	97,7	97,6	98,2	98,1	98,2
5	79,5	92,6	96,5	97,7	98,3	97,9	97,9	98
6	77,7	90,9	97	97,2	97,7	97,3	98,3	97,7
7	79	92,7	96,9	97,5	97,9	98,5	98,1	98,4
8	78,6	92,3	96,7	97,7	97,9	98,2	98,1	98,1
9	79,5	91,8	95,6	97,5	98	98,2	98,1	98,4
10	76,7	92,5	95,4	97	97,7	98,1	98,4	98,5
11	74,4	90,2	95,8	97,2	97,9	97,6	98,3	98,5
12	77,9	90,7	95,9	97,2	97,8	98,2	98,1	98,3
13	76,7	90,6	96,3	97,9	97,7	98,1	97,7	98,6
14	79,3	91,5	95,5	97,5	97,9	97,6	98	98,3
15	78,4	91,5	95,9	97,2	97,7	97,9	98,4	98,4
16	77	91,3	94,8	97,6	97,5	98,2	97,7	98,2
17	76,2	91,7	96,2	96,6	97,9	98,1	98,6	98,3
18	78,1	92	95,5	97,3	97,9	97,9	98,6	98,1
19	74,1	93,5	96	97,6	98	97,8	97,6	98,5
20	76,7	91,2	95,4	97,2	97,9	98	98,3	98,6
21	74,8	90,3	94,3	97,3	97,8	97,9	98,4	98
22	75,5	90,6	95	97,2	97,9	97,4	98	98
23	75,3	91,6	94,8	98,2	97,3	97,6	98	98,2
24	81,9	92,1	96,5	97,5	97,4	98,2	98	98,3
25	73,7	90,3	95,9	97,6	98	97,9	98,6	98,4
26	82,7	90,5	95,9	97	97,9	97,3	98,3	98
27	77,3	90,9	95,3	97,6	97,9	97,6	98,2	98,4
28	77,6	90,1	94,7	97,5	98	97,9	98,1	98,3
29	78,2	91,5	96,3	97,6	98	98,2	98,3	98,5
30	76,8	93	96,3	97,3	97,9	98	98,6	98,5

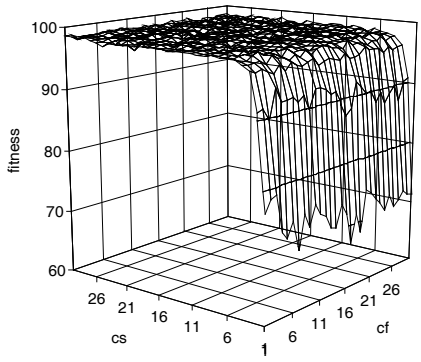
Wprowadzenie mechanizmu ścisku do modelu GCS było podyktowane chęcią zapewnienia oczywistej różnorodności populacji produkcji, ale również zapobieżenia wysokiej epistazie populacji produkcji. Uzyskane wyniki podczas badania związków pomiędzy selekcją turniejową a ściskiem (przy parametrze $cf = 1$) dowodzą, że istotniejszym, z punktu widzenia ewolucji, procesem jest sposób zastępowania osobnika z populacji osobnikiem wcześniej wyselekcjonowanym niż sam mechanizm selekcji osobnika podlegającego operacjom genetycznym. Zastępowanie produkcji jest najmniej efektywne dla indukcji całej gramatyki, gdy dotyczy losowego osobnika ($cs = 1$), efektywność liczona w kosztach i dokładności natomiast rośnie, kiedy wzrasta prawdopodobieństwo doboru osobnika najsłabszego w populacji.

5.6.3. Ścisk

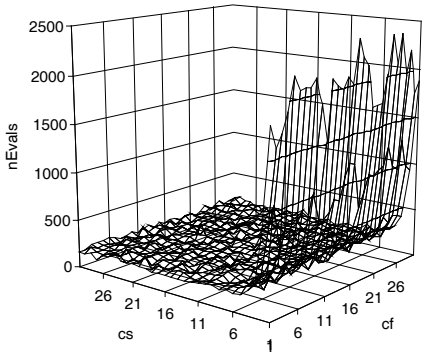
Wnioski wyciągnięte podczas analizy wpływu ścisku i selekcji turniejowej na proces uczenia gramatyki każą przyjrzeć się szczegółowo roli mechanizmu ścisku (zatłoczenia) w modelu GCS. W tym celu wykonano szereg eksperymentów, podczas których zmieniano zarówno wartość współczynnika ścisku *cf*, jak i rozmiar podpopulacji ścisku *cs*. Badania przeprowadzono dla selekcji ruletkowej i domyślnych wartości pozostałych parametrów, które konfigurowały model podczas indukcji języków regularnych, a zatem m.in. liczby początkowych produkcji nieterminalnych $n_{\text{start}} = 30$, pięćdziesiąt niezależnych iteracji programu ($n_{\text{run}} = 50$), w każdej iteracji maksymalnie 5000 kroków ewolucyjnych ($n_{\max} = 5000$).

Uzyskane wyniki zostały przedstawione w układzie przestrzennym – na rys. 56 zmiana wartości dokładności indukcji $fitness_{\text{avg}}$, a na rys. 57 zmiana wartości kosztów indukcji *nEvals*, oba estymatory w zależności od wartości współczynnika ścisku *cf* oraz rozmiaru podpopulacji *cs*. Proces indukcji jest najmniej efektywny w kontekście tempa zbieżności do gramatyki zgodnej ze zbiorem uczącym dla rozmiaru podpopulacji $cs = 1$, osiągając minimum wartości estymatorów $fitness_{\text{avg}}$ oraz *nEvals* dla współczynnika ścisku $cf = 17$ (odpowiednio 67% i 2441 kroków ewolucyjnych).

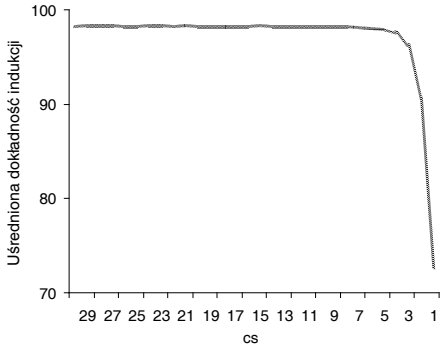
Dla kolejnych wartości *cs* indukcja, niezależnie od *cf*, coraz szybciej dochodzi do coraz wyższych wartości estymatora ogólnej kompetencji $fitness_{\text{avg}}$. Dla $cs = 2$ średnia dokładność indukcji ze wszystkich badanych wartości rozmiarów podpopulacji ścisku wynosi już 90,5%, w kroku następnym dla $cs = 3$ osiąga wartość 96%, by począwszy od kroku 3 utrzymywać się na poziomie około 98,2%. Podobnie rzecz się ma z wartością kosztu indukcji, który dla $cs = 2$ wynosi 942 kroki, 387 kroków dla $cs = 3$, a od $cs = 8$ oscyluje w granicach 170–180 kroków ewolucyjnych. Zjawisko to przedstawiono na rys. 58–59 – na rys. 58 uśrednione wartości estymatorów dokładności, a na rys. 59 kosztu indukcji. Każdy punkt obydwu wykresów jest odpowiednią średnią liczoną z 30 eksperymentów, podczas których zmieniano kolejno współczynnik ścisku *cf*. Praktycznie dla obydwu estymatorów nie odnotuje się istotnego wpływu wartości współczynnika ścisku *cs*, począwszy od $cs = 6$. Również zmiana *cf* dla wyższych wartości współczynnika *cs* nie wpływa praktycznie w żaden sposób na koszt i dokładność indukcji, co prowadzi wprost do oczywistych zaleceń przy wyborze optymalnych wartości tych parametrów. Zwiększanie wartości parametru *cs* powoduje zwiększenie prawdopodobieństwa wyboru produkcji rzeczywiście podobnej do produkcji mającej ją zastąpić. To rosnące prawdopodobieństwo obydwu produkcji jest o tyle istotne, że zmniejsza ryzyko zaburzenia istniejących już w gramatyce ciągów wyprowadzeń. Eksperymenty dowodzą, że nie ma potrzeby czasochłonnego przeglądu jak najliczniejszej populacji w poszukiwaniu podobnej produkcji.



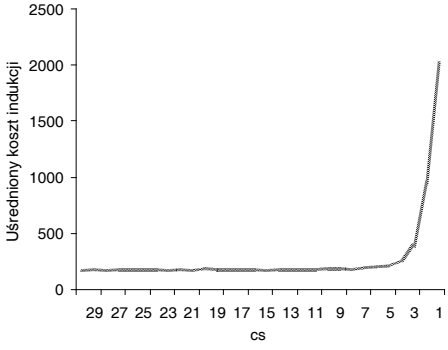
Rys. 56. Wykres przestrzenny zmiany dokładności indukcji $fitness_{avg}$ w zależności od wartości współczynnika ścisku *cf* oraz wielkości podpopulacji *cs*
Fig. 56. 3D graph illustrating change of induction accuracy $fitness_{avg}$ depending on crowding factor *cf* and crowding subpopulation *cs*



Rys. 57. Wykres przestrzenny zmiany kosztu indukcji *nEvals* w zależności od wartości współczynnika ścisku *cf* oraz wielkości podpopulacji *cs*
Fig. 57. 3D graph illustrating change of induction cost *nEvals* depending on crowding factor *cf* and crowding subpopulation *cs*



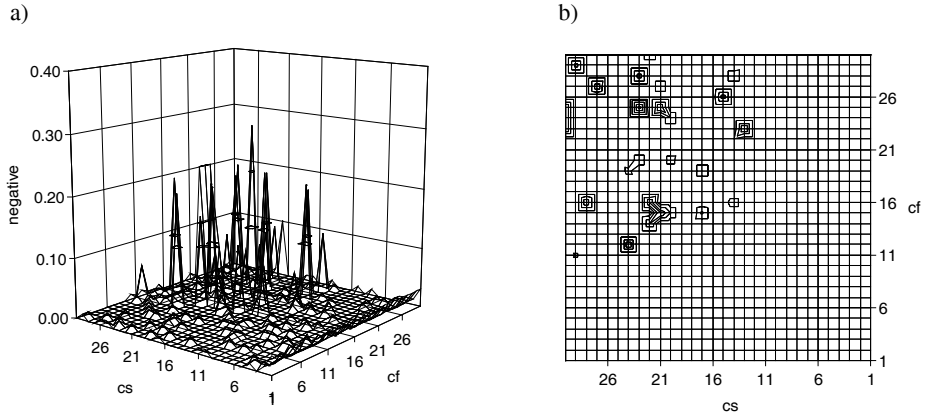
Rys. 58. Wykres uśrednionej dokładności indukcji $fitness_{avg}$ w funkcji zmiany rozmiaru podpopulacji *cs*; $fitness_{avg}$ został uśredniony po współczynniku ścisku *cf* zmienianym w przedziale [1, 30]
Fig. 58. Graph of average induction accuracy $fitness_{avg}$ as a function of crowding subpopulation *cs*; $fitness_{avg}$ was averaged over crowding factor *cf* changed in range [1, 30]



Rys. 59. Wykres uśrednionego kosztu indukcji *nEvals* zmiany rozmiaru podpopulacji *cs*; *nEvals* został uśredniony po współczynniku ścisku *cf* zmienianym w przedziale [1, 30]
Fig. 59. Graph of average induction cost *nEvals* as a function of crowding subpopulation *cs*; *nEvals* was averaged over crowding factor *cf* changed in range [1, 30]

Do ciekawych wniosków prowadzi obserwacja zachowania się wartości kompetencji negatywnej *negative* w funkcji zmiany parametrów *cf* oraz *cs* (rys. 60). W całym badanym zakresie zmian obydwu parametrów, estymator *negative* nie przekroczył

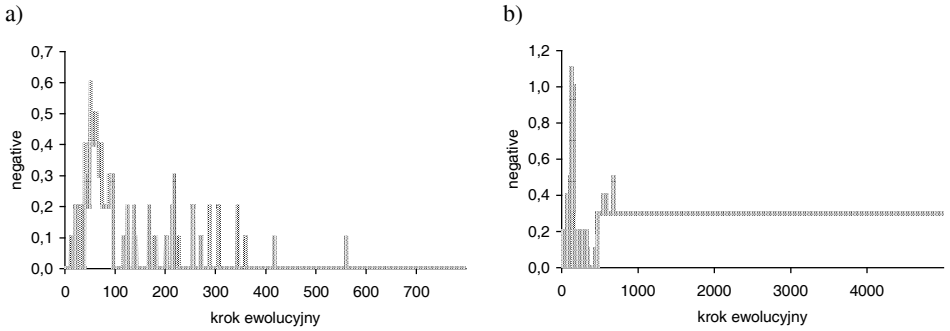
niskiej wartości 0,3%, co oznacza, że w najgorszym przypadku podczas uczenia gramatyki średnio niepoprawnie klasyfikowanych było mniej niż 3 zdania na 1000. Przestrzenny wykres zmian wartości estymatora przedstawiony na rys. 60a zwraca uwagę na wyróżniające się z otoczenia skoki wartości estymatora w prawym górnym rogu, odpowiadającym wysokim wartościom współczynnika ścisku *cf* oraz wielkości podpopulacji *cs*. Dokładne położenie owych „pików" wartości estymatora *negative* ilustruje dobrze wykres poziomicowy zamieszczony na rys. 60b.



Rys. 60. Zmiana kompetencji negatywnej $negative_{avg}$ w zależności od wartości współczynnika ścisku *cf* oraz wielkości podpopulacji *cs*: a) wykres przestrzenny, b) wykres poziomicowy (poziomica co 0,05%)

Fig. 60. Change of negative competence $negative_{avg}$ depending on crowding factor *cf* and crowding subpopulation *cs*: a) 3D graph b) contour graph (contour line every 0.05%)

Podpopulacja ścisku *cs* określa rozmiar losowanego z całej populacji zbioru produkcji, spośród których wybierana jest najgorsza, natomiast współczynnik ścisku *cf* mówi, ile razy pobierane są próbki takich zbiorów. W efekcie otrzymujemy populacje o rozmiarze *cf*. Zwiększanie podpopulacji ścisku zwiększa prawdopodobieństwo wyboru słabego osobnika nie tylko lokalnie, ale również i globalnie. Podobny efekt niesie ze sobą wzrost wartości współczynnika *cf*. W konsekwencji otrzymujemy podpopulację produkcji o niskiej wartości przystosowania. Zatem wysokie wartości współczynników *cs* i *cf* zwiększają prawdopodobieństwo wyboru, a w konsekwencji zastępowania, bardzo słabego klasyfikatora. Generalnie jest to pozytywna własność metody ścisku, ale może się również zdarzyć, że zastępowana słaba produkcja gramatyki uczestniczy w pełnym rozbiorze zdania negatywnego. Zamiana takiej produkcji produkcją podobną – a na tym opiera się mechanizm ścisku – może utrzymać zdolność całej gramatyki do rozbioru również przykładów spoza języka.



Rys. 61. Zmiana estymatora *negative*: a) *cf* = 22 i *cs* = 25, b) *cf* = 25 i *cs* = 23
Fig. 61. Change of *negative* estimator: a) *cf* = 22 and *cs* = 25, b) *cf* = 25 and *cs* = 23

Na rysunku 61 pokazano zachowanie się estymatora *negative* w procesie uczenia dla parametrów ścisku *cf* = 25 i *cs* = 22 (rys. 61a) oraz *cf* = 25 i *cs* = 23 (rys. 61b), a więc jednego z „pików" wartości $negative_{avg}$ przedstawionych na rys. 60. Widać wyraźnie w obu przypadkach, jak model, poszukując ewolucyjnie dobrych produkcji, bada również w pierwszych krokach produkcje biorące udział parsowaniu przykładów negatywnych. Rysunek 61a jest ilustracją procesu indukcji, który uzyskuje niską wartości estymatora $negative_{avg}$. Na początku uczenia wartość negatywnej kompetencji gramatyki gwałtownie wzrasta, ale już od kroku 53 następuje stopniowe wytłumianie niepożądanych własności indukowanej gramatyki, aż do zaniku „złych produkcji" w populacji w kroku 562. Na rysunku 61b przedstawiono indukcję gramatyki, podczas której uzyskano stosunkową wysoką wartość estymatora $negative_{avg}$ ($negative_{avg}$ = 0,28%) Po skoku wartości komptencji negatywnej w pierwszych krokach ewolucyjnych, następuje początkowo szybki zanik produkcji parsujących zdania negatywne, lecz w kolejnych krokach ewolucyjnych pojawiają się ponownie w zbiorze produkcji i już do końca ewolucji model nie jest w stanie ich całkowicie zastąpić produkcjami podobnymi, ale nieparsującymi zdań negatywnych.

5.6.4. Operatory pokrycia

W modelu GCS wprowadzono pięć operatorów pokrycia:
- operator pokrycia terminalnego,
- operator pokrycia startowego,
- operator pokrycia pełnego,
- operator pokrycia agresywnego,
- operator pokrycia uniwersalnego.

Operator pokrycia terminalnego jest uruchamiany przez procedurę parsowania zdania automatycznie i w związku z tym nie będzie badany jego wpływ na proces

indukcji gramatyki. Aby umożliwić pełny, a jednocześnie zwięzły opis kombinacji wszystkich pozostałych operatorów pokrycia, zastosowano czteroliterowy skrót na oznaczenie występowania bądź też nie w parametrach modelu każdego z operatorów. Zapis **s p a u** oznaczać będzie, że zastosowano wszystkie operatory (**s**-tartowy, **p**-ełny, **a**-gresywny, **u**-niwersalny), wyłączenie operatora sygnalizuje kreseczka „-" w miejsce jego pozycji w skrócie. Skrót uzupełniony jest dodatkowo wartością prawdopodobieństwa p_a, gdy w doświadczeniu użyto operatora agresywnego.

Tabela 17. Porównanie kosztów i dokładności indukcji dla różnych kombinacji operatorów pokrycia

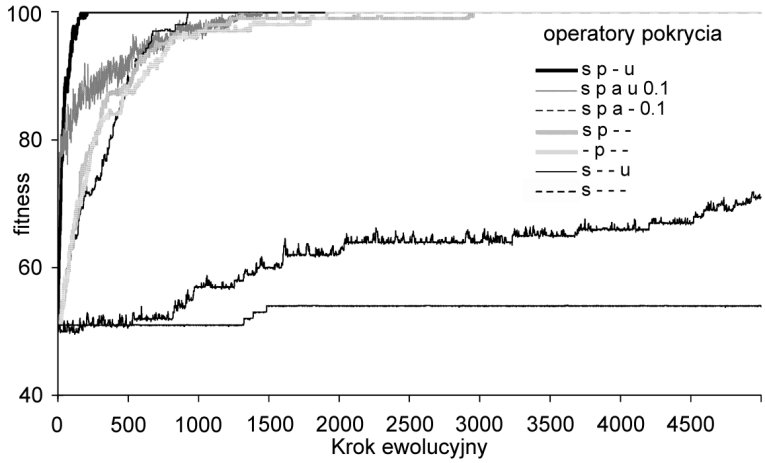
Operator pokrycia	$fitness_{avg}$	$positive_{avg}$	*nEvals*	*nSuccess*
sp-u	99,60	99,30	55,54	50/50
spau 0,1	98,10	99,80	429,54	50/50
spa- 0,1	96,80	93,70	329,46	50/50
sp--	96,60	93,20	352,12	50/50
-p--	96,40	92,80	373,52	50/50
s--u	61,90	24,00	2217,10	21/50
s---	53,20	6,33	1048,25	3/50
s-a- 0,1	50,70	1,35	1654,00	1/50

W tabeli 17 zestawiono wyniki indukcji gramatyki TOY dla różnych kombinacji zastosowanych w eksperymentach operatorów pokrycia. Wartości pozostałych parametrów modelu we wszystkich eksperymentach były takie same jak podczas indukcji języków regularnych (m.in. $n_{start} = 30$, $n_{run} = 50$, $n_{max} = 5000$). Tabelę uporządkowano wg najwyższych wartości uzyskanej dokładności indukcji $fitness_{avg}$. Wyniki uzupełnia graficzna reprezentacja przebiegu kompetencji ogólnej *fitness* dla każdej z badanych kombinacji operatorów (rys. 62).

Najwyższe wartości syntetycznych estymatorów dokładności $fitness_{avg}$ oraz $positive_{avg}$ uzyskano dla włączonych operatorów pokrycia startowego, pełnego i uniwersalnego. Wykres krzywej zbieżności *fitness* dla tej kombinacji operatorów pokrycia jest wyraźnie bardziej stromy od pozostałych krzywych, a średnia liczba kroków, w których model GCS znajduje zgodną ze zbiorem uczącym gramatykę, jest sześciokrotnie (!) mniejsza od drugiej w kolejności pod względem tempa zbieżności kombinacji operatorów.

Na drugiej pozycji uplasowała się kombinacja spau 0,1, która w stosunku do poprzedniej różni się dodatkowo uruchomionym operatorem pokrycia agresywnego z $p_a = 0{,}1$. Wykres zbieżności dla tej kombinacji również bardzo stromo pnie się do góry w funkcji początkowych kroków ewolucyjnych, by w okolicach 500 kroku praktycznie połączyć się z wykresami trzech kolejnych kombinacji operatorów pokrycia. Bardzo podobne do siebie wartości syntetycznych estymatorów oraz zbliżony przebieg uśrednionego dopasowania mają następujące kombinacje operatorów:

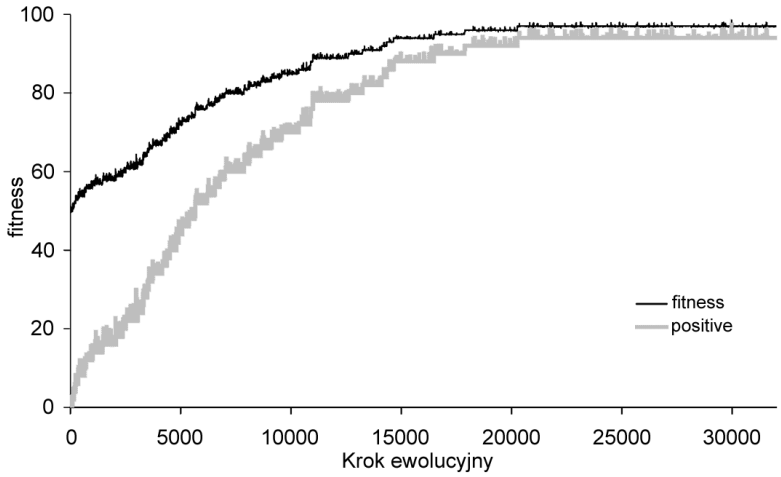
spa-0,1, sp--, -p--. Elementem wspólnym tych trzech kombinacji jest występowanie pokrycia pełnego, co prowadzi do wniosku o jego decydującym znaczeniu w tych kombinacjach. O roli właśnie operatora pokrycia pełnego świadczą również ostatnie trzy z badanych kombinacji, czyli s--u, s--, s-a-0,1, w których operator pokrycia pełnego był wyłączony. Dla tych kombinacji uzyskano istotnie mniejsze wartości wszystkich analizowanych estymatorów, a nie wszystkie iteracje eksperymentów kończyły się sukcesem. Przebieg krzywej zbieżności estymatora *fitness* dla kombinacji s--- oscyluje wokół linii 50% z zauważalnym skokiem w okolicach 1300 kroku, który jest wynikiem znalezienia w 3 z 50 iteracji poszukiwanej gramatyki (pierwsza z zakończonych sukcesem iteracji zakończyła się w kroku 1388, kolejna w 1322, a ostatnia w 1483). Uzupełnienie pokrycia startowego operatorem pokrycia uniwersalnego wprowadza zauważalne polepszenie rezultatów. Krzywa kombinacji s--u pnie się wolno w górę przez wszystkie kroki ewolucyjne, osiągając średnią wartość $fitness_{avg}$ = 61,90%, niewątpliwe ograniczoną przyjętą maksymalną liczbą kroków w pojedynczej indukcji. Z 50 iteracji eksperymentu niemalże już połowa kończy się wyuczeniem prawidłowej gramatyki.



Rys. 62. Zmiana estymatora *fitness* dla różnych kombinacji operatorów pokrycia
Fig. 62. Change of *fitness* estimator for different combinations of covering operators

Aby zweryfikować poczynione wyżej spostrzeżenie dotyczące przerwanego procesu indukcji, wykonano eksperyment, w którym zwiększono maksymalną liczbę kroków ewolucyjnych do 50 000. Na rysunku 63 wykreślono przebieg kompetencji pozytywnej i negatywnej dla kombinacji operatorów pokrycia s--u i n_{max} = 50 000. Model dla tak wydłużonego okresu ewolucji osiąga następujące wartości estymatorów dokładności i kosztu indukcji: $fitness_{avg}$ = 91,40%, $positive_{avg}$ = 82,90%, *nEvals* = 8601,28 (*minEvals* =123, *maxEvals* = 40750), *nSuccess* = 50/50. Operator pokrycia

uniwersalnego jest zatem w stanie praktycznie samodzielnie, bez wspomagania działaniem operatora pełnego, zapewnić w odpowiednio długim czasie indukcję gramatyki w każdej iteracji.

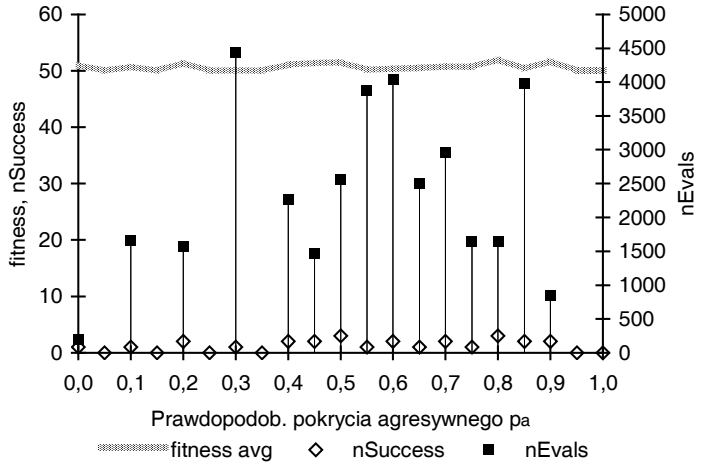


Rys. 63. Zmiana estymatora *fitness* i *negative* dla kombinacji operatorów pokrycia s--u i $n_{\max} = 50\,000$
Fig. 63. Change of *fitness* and *negative* estimators for combination of covering operators s--u and $n_{\max} = 50\,000$

Należy zauważyć, że operator ten istotnie poprawia tempo zbieżności procesu indukcji w każdym z przebadanych przypadków, a do takiego wniosku skłania porównanie uśrednionej kompetencji ogólnej następujących par s--- i s--u (53,20% do 61,90%), sp-- i sp-u (96,60% do 99,60%) oraz spa-0,1 i spau0,1 (96,80% do 98,10%). Decydującym jednak w sposób zasadniczy operatorem jest operator pokrycia pełnego. Brak operatora pokrycia pełnego w badanych kombinacjach operatorów drastycznie obniża zdolność modelu do indukcji gramatyki. Świadczą o tym niskie wartości estymatorów dokładności indukcji, wysokie wartości kosztu indukcji oraz liczne iteracje, podczas których model nie był w stanie wyuczuć się gramatyki. Kolejny wniosek jaki można wysnuć, analizując wyniki, to niewielki wpływ na proces indukcji jaki ma operator startowy, ale również nieznaczna rola w uczeniu operatora pokrycia agresywnego. Ze względu jednak na parametryczny charakter operatora agresywnego, zbadano jego wpływ na proces indukcji z uwzględnieniem zmiany w pełnym dopuszczalnym zakresie wartości prawdopodobieństwa p_a.

Analizę wpływu operatora agresywnego na uczenie gramatyki rozpoczęto od kombinacji s-a-, czyli wyłączonych operatorów pokrycia pełnego i uniwersalnego. Wstępne eksperymenty wykazały, że kombinacja operatorów startowego i agresywnego ma niskie zdolności uczenia gramatyki (patrz tab. 17). Zbadano zachowanie się modelu GCS dla prawdopodobieństwa operatora pokrycia agresywnego p_a zmieniane-

go w przedziale [0, 1] z krokiem 0,05. Wyniki zebrano w tab. 18, a przedstawiono na rys. 64. W całym badanym przedziale zmienności parametru p_a nie uzyskano lepszego wyniku niż 3 zakończone sukcesem iteracje na 50 (dla $p_a = 0{,}5$ oraz $p_a = 0{,}8$). Dla 6 z 20 sprawdzanych wartości parametru proces indukcji w żadnej iteracji nie znalazł rozwiązania, w tym dla dwóch najwyższych wartości $p_a = 0{,}95$ i $p_a = 1{,}0$ (dlatego też wartości parametru *nSuccess* na rys. 64 reprezentują słupki wartości, a nie linia ciągła).



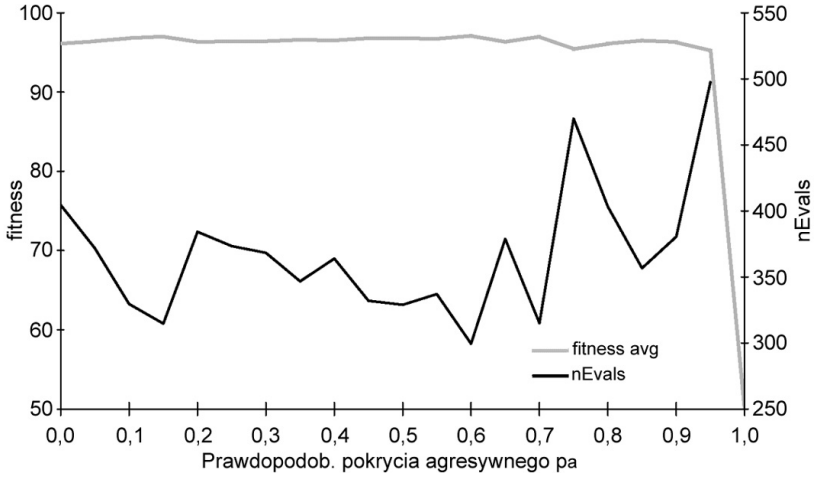
Rys. 64. Wpływ prawdopodobieństwa pokrycia agresywnego p_a na dokładność i koszt indukcji (kombinacja operatorów pokrycia s-a-)
Fig. 64. Influence of aggressive covering probability p_a on accuracy and cost of induction (combination of covering operators s-a-)

Tabela 18. Wartości estymatorów dokładności i kosztu indukcji dla zmiennego prawdopodobieństwa pokrycia agresywanego p_a oraz kombinacji operatorów pokrycia s-a-

p_a	$fitness_{avg}$	*nEvals*	*nSuccess*	p_a	$fitness_{avg}$	*nEvals*	*nSuccess*
0	51	192	1/50	0,55	50,2	3882	1/50
0,05	50	–	0/50	0,6	50,4	4041,5	2/50
0,1	50,7	1654	1/50	0,65	50,5	2502	1/50
0,15	50	–	0/50	0,7	50,8	2954,5	2/50
0,2	51,4	1563,5	2/50	0,75	50,7	1643	1/50
0,25	50	–	0/50	0,8	52	1642,33	3/50
0,3	50,1	4441	1/50	0,85	50,4	3986	2/50
0,35	50	–	0/50	0,9	51,7	839	2/50
0,4	51,1	2265,5	2/50	0,95	50	–	0/50
0,45	51,4	1461	2/50	1	50	–	0/50
0,5	51,5	2555,67	3/50				

Obserwując przebieg estymatora $fitness_{\text{avg}}$ na rys. 64, można stwierdzić, że operator pokrycia agresywnego nie wpływa w sposób znaczący na dokładności i koszt indukcji gramatyki dla kombinacji operatorów pokrycia s-a-.

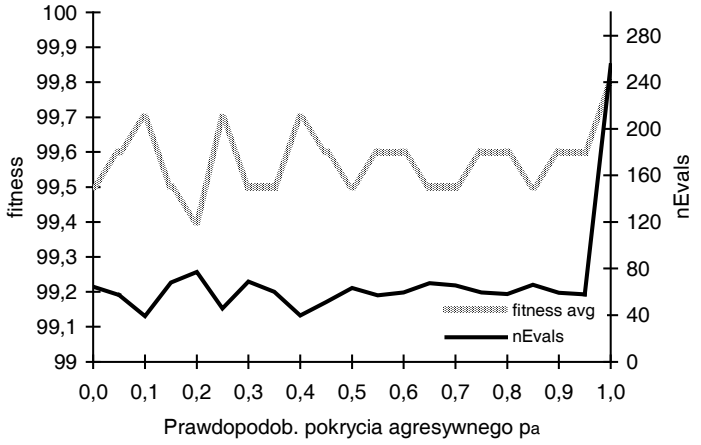
W następnej serii eksperymentów przebadano wpływ operatora pokrycia agresywnego dla kombinacji operatorów spa-, czyli z dodatkowo, w stosunku do poprzednich eksperymentów, włączonym operatorem pokrycia pełnego. Uzyskane wyniki przedstawiono na rys. 65. Przebieg wykresu uśrednionej kompetencji ogólnej $fitness_{\text{avg}}$ jest niezależny od zmiany wartości prawdopodobieństwa p_a, natomiast średni koszt indukcji *nEvals* ma dwa duże skoki wartości dla $p_a = 0{,}75$ i $p_a = 0{,}95$. Dla $p_a = 1{,}00$ model GCS nie jest w stanie w żądanej iteracji eksperymentu znaleźć gramatyki. Wnioski, które się nasuwają, są podobne jak w przypadku analizy wpływu operatora na poprzednio badaną kombinację – operator pokrycia agresywnego nie wpływa w sposób znaczący na dokładności i koszt indukcji gramatyki dla kombinacji operatorów pokrycia spa-, a dla uruchamianego w każdym kroku ewolucyjnym operatora ($p_a = 1{,}00$) model GCS traci całkowicie zdolności uczenia.



Rys. 65. Wpływ prawdopodobieństwa pokrycia agresywnego p_a na dokładność i koszt indukcji (kombinacja operatorów pokrycia spa-)
Fig. 65. Influence of aggressive covering probability p_a on accuracy and cost of induction (combination of covering operators spa-)

W ostatniej serii eksperymentów zmieniano prawdopodobieństwo operatora pokrycia agresywnego dla kombinacji operatorów pokrycia spau, czyli z włączonym zarówno operatorem pokrycia pełnego, jak i operatorem uniwersalnym. Wyniki przedstawiono na rys. 66. Wykresy wartości estymatorów $fitness_{\text{avg}}$ oraz *nEvals* wskazują jednoznacznie, że również i dla tej kombinacji operatorów operator pokrycia agresywnego nie wpływa znacząco na jakość procesu indukcji gramatyki. W przeciwień-

stwie jednak do poprzednio badanej kombinacji, dla $p_a = 1{,}00$ model GCS nie traci zdolności uczenia, a jedynie dosyć drastycznie (czterokrotnie!) rośnie średnia liczba kroków potrzebnych do znalezienia gramatyki.



Rys. 66. Wpływ prawdopodobieństwa pokrycia agresywnego p_a na dokładność i koszt indukcji (kombinacja operatorów pokrycia spau)
Fig. 66. Influence of aggressive covering probability p_a on accuracy and cost of induction (combination of covering operators spau)

Podsumowując wykonane eksperymenty, można stwierdzić, że na proces indukcji największy wpływ ma operator pokrycia pełnego, który jest w stanie samodzielnie zapewnić uczeniu dobre tempo zbieżności. Zdecydowanie mniejsze tempo zbieżności ma operator pokrycia uniwersalnego, który potrzebuje około dziesięciokrotnie dłuższego okresu uczenia, aby w każdej iteracji eksperymentu indukować prawidłową gramatykę. Połączenie operatora pełnego i uniwersalnego daje najlepsze wyniki kosztów i dokładności indukcji. Brak obydwu operatorów czyni proces uczenia praktycznie niemożliwym. Operatory pokrycia startowego oraz agresywnego mają znikomy wpływ na indukcję gramatyki. Dopiero w parze z operatorem pełnym lub uniwersalnym, proces indukcji staje się możliwy i nieprzypadkowy.

Nieznaczna rola operatora pokrycia startowego w uczeniu gramatyki staje się zrozumiała, gdy spojrzymy na zasadę działania tego operatora. Zadaniem operatora startowego jest bowiem utworzenie produkcji typu $S \rightarrow a$ dla jednoelementowego zdania pozytywnego. Niewielka, a w dużych zbiorach uczących znikoma, obecność zdań o długości 1 oraz stosunkowo prosta reguła do wyuczenia w sposób istotny ograniczają wpływ operatora startowego na proces indukcji gramatyki.

Nietrywialnym wnioskiem, jaki nasuwa się podczas analizy uzyskanych wyników, jest spostrzeżenie braku wpływu na indukcję operatora pokrycia agresywnego prawie w całym zakresie zmienności wartości prawdopodobieństwa p_a. Można by oczekiwać, że tworzona automatycznie brakująca produkcja nieterminalna powinna w sposób zauważalny wspo-

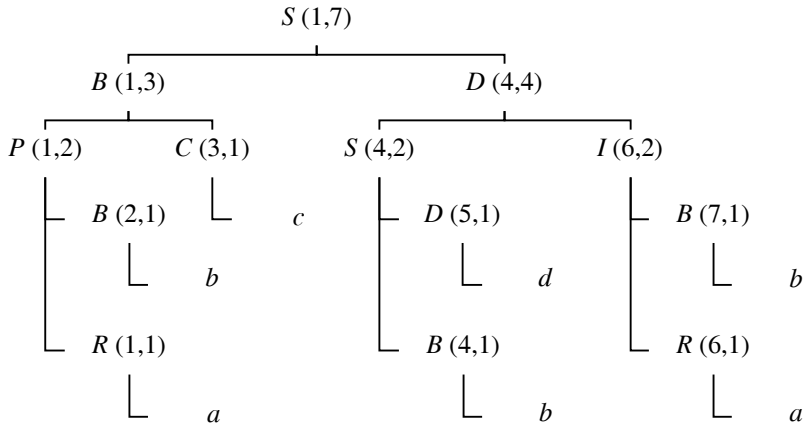
magać proces uczenia. Jednak o ile prawą stronę nowej produkcji tworzy jedno z dopuszczalnych rozwinięć rozbioru zdania, o tyle symbol nieterminalny stanowiący lewą stronę produkcji jest dobierany przez operator losowo. Trudno zatem oczekiwać w takiej sytuacji ukierunkowanego procesu poszukiwania odpowiedniego zestawu epistatycznie powiązanych ze sobą reguł parsujących zdanie. Co więcej, jeżeli prawdopodobieństwo zastosowania operatora $p_a = 1$, to uczenie gramatyki zostaje całkowicie zahamowane i tylko obecność operatora pokrycia uniwersalnego może podtrzymać proces uczenia (kosztem zwiększonego okresu uczenia). Powodem jest zapewne w tej sytuacji zbyt częsta, bo odbywająca się przy każdorazowym napotkaniu braku możliwości dalszego rozbioru analizowanego fragmentu zdania, losowa wymiana materiału genetycznego. Nowo utworzona przez operator pokrycia agresywnego produkcja jest dodawna ze ściskiem do istniejącego zbioru produkcji, zastępując jedną z produkcji gramatyki.

Operator pokrycia pełnego działa w sposób podobny do operatora agresywnego, gdyż również tworzy produkcję, której lewa strona opisuje rozwinięcie analizowanego ciągu. Jednak prawa strona produkcji nie jest losowa, ponieważ operator pokrycia pełnego uruchamiany jest tylko podczas wypełniania ostatniej analizowanej komórki tablicy CYK. Zatem jedynym symbolem, jaki może się pojawić w produkcji jest symbol startowy gramatyki. Operator pokrycia pełnego niejako zamyka tą ostatnią produkcją proces indukcji zestawu produkcji zdolnych do rozbioru przetwarzanego zdania, przyśpiesza – i to w sposób radykalny, jak wykazują eksperymenty – ewolucję gramatyki zgodnej ze zbiorem uczącym.

Ostatni z analizowanych operatorów pokrycia – operator pokrycia uniwersalnego działa na zupełnie innej zasadzie niż operator agresywny czy też pełny. Operator pokrycia uniwersalnego dodaje do zbioru produkcji dodatkowe produkcje terminalne, które wyprowadzają symbole terminalne gramatyki z jednego symbolu nieterminalnego. Działanie tego operatora jest wzorowane na mechanizmie uczącego się systemu klasyfikującego, w którym w warunku klasyfikatora może wystąpić symbol nieistotny (*don't care*), zastępujący dowolny symbol komunikatu. Jak dowiodły badania empiryczne, operator pokrycia uniwersalnego zapewnia modelowi samodzielnie możliwości uczenia, a w połączeniu z innymi operatorami istotnie poprawia zbieżność ewolucji. Poniżej zamieszczono gramatykę wyewoluowaną przez model z kombinacją operatorów pokrycia s--u:

1. $S \rightarrow BD$
2. $S \rightarrow IB$
3. $S \rightarrow BD$
4. $R \rightarrow a$
5. $R \rightarrow c$
6. $R \rightarrow b$
7. $R \rightarrow d$
8. $P \rightarrow BO$
9. $P \rightarrow RB$
10. $B \rightarrow B$
11. $I \rightarrow IR$
12. $I \rightarrow RB$
13. $D \rightarrow SI$
14. $D \rightarrow d$
15. $D \rightarrow BO$
16. $D \rightarrow RB$
17. $D \rightarrow BB$
18. $C \rightarrow c$
19. $B \rightarrow SB$
20. $B \rightarrow PC$
21. $B \rightarrow b$
22. $A \rightarrow a.$

Wcześniejsze eksperymenty dowiodły, że sam operator pokrycia startowego jest za słaby, by model był w stanie wyuczyć się gramatyki podczas każdej iteracji. Dopiero uruchomienie operatora pokrycia uniwersalnego umożliwia skuteczny proces uczenia. Na rysunku 67 przedstawiono drzewo rozbioru pozytywnego zdania *abcbdab* otrzymane podczas działania modelu GCS z kombinacją operatorów pokrycia s--u. W węzłach drzewa rozbioru podano symbol nieterminalny wraz z jego położeniem w tablicy CYK (*kolumna*, *wiersz*). Drzewo rozbioru wyjaśnia rolę operatora uniwersalnego w procesie indukcji gramatyki. Widać bowiem wyraźnie, że użycie przez gramatykę symbolu uniwersalnego *R* potęguje zdolności eksploracyjne modelu. Drzewo rozbioru na rys. 67 jest bowiem w istocie schematem rozbioru wszystkich ciągów postaci **bcbd*b*, gdzie symbol * oznacza dowolny symbol terminalny.



Rys. 67. Drzewo rozbioru zdania *abcbdab* (kombinacja operatorów pokrycia s--u); symbol nieterminalny *R* jest symbolem uniwersalnym
Fig. 67. Parsing tree for the sentence *abcbdab* (combination of covering operators s--u); symbol *R* stands for don't care symbol

5.6.5. Liczba początkowych produkcji nieterminalnych

W dotychczasowych eksperymentach stosowano arbitralnie przyjętą liczbę 30 początkowych produkcji nieterminalnych (n_{start} = 30). Zasadne jest zbadanie, na ile i w jakim zakresie liczba ta ma wpływ na proces indukcji gramatyki. Gramatyka TOY, która jest przedmiotem badań symulacyjnych modelu GCS, opisana jest 6 produkcjami nieterminalnymi oraz 4 produkcjami wyprowadzającymi symbole terminalne. Należy zatem oczekiwać, że udane uczenie powinno zachodzić, począwszy od 6 reguł nieterminalnych. Na rysunku 68 przedstawiono zależność estymatorów $fitness_{avg}$ (oznaczone w tabeli przez symbol f_{avg}), $positive_{avg}$ (p_{avg}), *nEvals* oraz *nSuccess* (*nS*) od liczby początkowych produkcji nieterminalnych n_{start} (ekspery-

menty prowadzone przy domyślnym zestawie parametrów). W tabeli 19 zebrano wyniki eksperymentów. Model GCS był w stanie znaleźć gramatykę zgodną ze zbio-

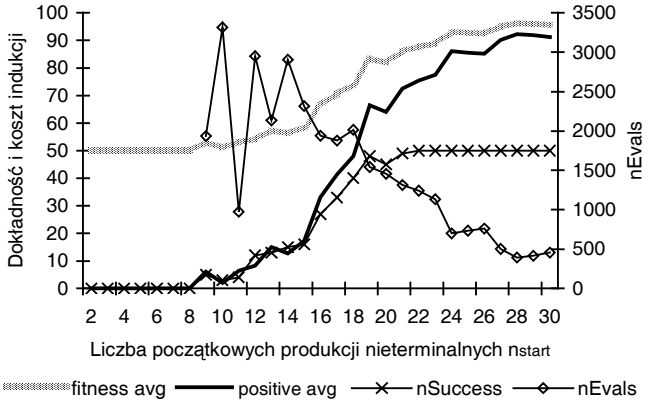
Tabela 19. Wartości estymatorów dokładności i kosztu indukcji dla zmiennej liczby początkowych produkcji nieterminalnych n_{start}

n_{start}	f_{avg}	p_{avg}	*nEvals*	*nS*	n_{start}	f_{avg}	p_{avg}	*nEvals*	*nS*
2	50,0	0,0	–	0/50	17	70,7	41,4	1879,3	33/50
3	50,0	0,0	–	0/50	18	74,0	48,0	2014,7	40/50
4	50,0	0,0	–	0/50	19	83,3	66,5	1543,1	48/50
5	50,0	0,0	–	0/50	20	82,0	64,0	1456,6	45/50
6	50,0	0,0	–	0/50	21	86,2	72,5	1313,9	49/50
7	50,0	0,0	–	0/50	22	87,7	75,4	1241,4	50/50
8	50,0	0,0	–	0/50	23	88,8	77,6	1134,2	50/50
9	53,1	6,1	1937,8	5/50	24	93,1	86,2	700,0	50/50
10	51,0	2,1	3316,3	3/50	25	92,8	85,5	735,0	50/50
11	53,2	6,5	973,3	4/50	26	92,5	85,1	761,4	50/50
12	54,1	8,2	2950,1	12/50	27	95,1	90,2	501,5	50/50
13	57,5	15,0	2134,3	13/50	28	96,2	92,3	395,2	50/50
14	56,3	12,7	2905,1	15/50	29	96,0	91,9	414,1	50/50
15	58,6	17,3	2315,1	16/50	30	95,6	91,2	455,9	50/50
16	66,6	33,2	1939,0	27/50					

rem uczącym dopiero dla 9 produkcji nieterminalnych i to tylko w 5 iteracjach na 50. Jest to zatem minimalna liczność populacji niezbędna do ewolucji gramatyki TOY. Poniżej zapisano przykład wyindukowanego zbioru produkcji:

1. $S \rightarrow SM$
2. $S \rightarrow QM$
3. $Q \rightarrow AB$
4. $M \rightarrow CB$
5. $C \rightarrow DA$
6. $A \rightarrow a$
7. $B \rightarrow b$
8. $D \rightarrow d$
9. $C \rightarrow c$.

Model znalazł gramatykę, która składa się z 5 produkcji nieterminalnych, o jedną produkcję mniej od gramatyki wzorcowej. Dopiero od 22 produkcji startowych model indukuje gramatykę w każdej iteracji eksperymentu. Krzywe zbieżności kompetencji ogólnej i pozytywnej rosną jednak ponad owe 22 produkcje, osiągając nasycenie w okolicach 28 reguł startowych. Analogicznie krzywa zbieżności koszt indukcji *nEvlas* maleje do liczby 28 reguł, po czym następuje jej stabilizacja. Dalszy wzrost liczby produkcji nie powoduje już poprawy wartości żadnego z analizowanych estymatorów.



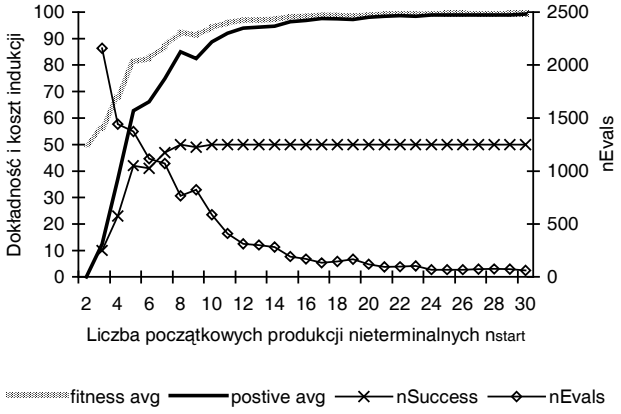
Rys. 68. Wpływ liczby początkowych produkcji nieterminalnych n_{start} na dokładność i koszt indukcji
Fig. 68. Influence of number of initial nonterminal productions n_{start} on accuracy and cost of induction

Zwiększanie liczby początkowych produkcji nieterminalnych sprawia jednak, że indukowane gramatyki mogą składać się z większej liczby produkcji. Przykładowo, proces uczenia z początkowymi 30 produkcjami startowymi zakończył się znalezieniem gramatyki złożonej z 11 produkcji nieterminalnych:

1. $S \rightarrow KB$
2. $S \rightarrow HB$
3. $S \rightarrow AS$
4. $S \rightarrow KD$
5. $S \rightarrow KG$
6. $K \rightarrow JA$
7. $J \rightarrow GD$
8. $G \rightarrow AB$
9. $H \rightarrow SC$
10. $D \rightarrow AG$
11. $A \rightarrow BC$
12. $A \rightarrow a$
13. $B \rightarrow b$
14. $C \rightarrow c$
15. $D \rightarrow d$.

Powyżej opisane eksperymenty dotyczyły indukcji z zestawem domyślnym parametrów. Parametrem, który ma istotny wpływ na liczność populacji, jest operator pokrycia uniwersalnego. Na rysunku 69 wykreślono przebieg krzywych estymatorów $fitness_{avg}$, $positive_{avg}$, *nEvals* oraz *nSuccess* w zależności od liczby początkowych produkcji nieterminalnych n_{start} z uruchomionym operatorem pokrycia uniwersalnego. Porównanie odpowiednich wykresów z rys. 68 oraz rys. 69 wskazuje na istotne różnice w przebiegu procesu indukcji z wyłączonym oraz włączonym operatorem uniwersalnym. Przede wszystkim model GCS wzmocniony operatorem pokrycia uniwersalnego znajduje gramatykę zgodną ze zbiorem uczącym już dla populacji 3 początkowych reguł nietermi nalnych i to w 10 iteracjach na 50. Poniżej umieszczono wyindukowaną gramatykę, która składa się jedynie z 2 produkcji nieterminalnych:

1. $S \rightarrow AB$
2. $B \rightarrow BR$
3. $A \rightarrow a$
4. $B \rightarrow b$
5. $C \rightarrow c$
6. $D \rightarrow d$
7. $R \rightarrow a$
8. $R \rightarrow b$
9. $R \rightarrow c$
10. $R \rightarrow d.$



Rys. 69. Wpływ liczby początkowych produkcji nieterminalnych n_{start} na dokładność i koszt indukcji – włączony operator pokrycia uniwersalnego
Fig. 69. Influence of number of initial nonterminal productions n_{start} on accuracy and cost of induction with don't care covering on

Gramatyka rozpoznaje zdania poprawne po rozpoczynającym je dwuliterowym podciągu *ab*. Każde inne zdanie jest przykładem niepoprawnym. Pomimo tego, że zbiór produkcji jest zredukowany do dwóch produkcji, gramatyka ta uzyskuje w testach generalizacji dokładność *Gen* = 99,80%!

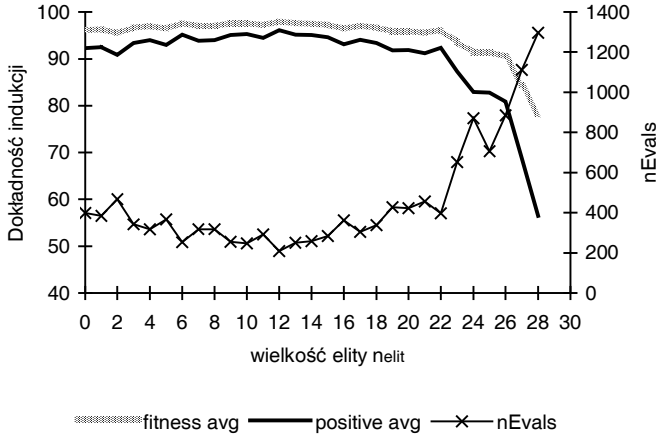
Zwiększanie parametru n_{start} powoduje w tempie logarytmicznym zmniejszanie się średniej liczby kroków niezbędnych do wyuczenia gramatyki, a w tempie potęgowym wzrost wartości syntetycznych estymatorów kompetencji. Wartości estymatora *nEvals* dla $n_{start} > 23$ nie przekraczają wartości 72 kroków ewolucyjnych, w porównaniu do znacznie ponad 400 kroków podczas indukcji bez operatora pokrycia uniwersalnego.

5.6.6. Wielkość elity

W modelu GCS zaimplementowano również sukcesję elitarną. Parametr n_{elit} określa liczbę najlepszych produkcji ze względu na wartość przystosowania, podczas ewolucji przechodzą one w postaci niezmienionej do następnego kroku ewolucyjnego. Innymi słowy produkcje te nie podlegają operatorom genetycznymi i mechanizmowi ścisku (patrz **procedure Algorytm genetyczny**, rys. 30).

Oczywiście analiza wpływu wielkości elity na indukcję gramatyki musi uwzględniać dwa inne parametry, a mianowicie liczbę początkowych produkcji nieterminal-

nych, które określają wielkość ewoluowanej populacji, oraz minimalną liczbę produkcji poszukiwanej gramatyki. Na rysunku 70 wykreślono zależność parametrów *fitness*$_{avg}$, *positive*$_{avg}$, *nEvals* od wielkości elity n_{elit}. Eksperymenty były przeprowadzane dla domyślnego zestawu parametrów, a więc m.in. dla 30 początkowych produkcji nieterminalnych. Zwiększanie wielkości elity nie wpływa w sposób istotny na tempo zbieżności ewolucji, choć można zauważyć nieznaczne wyższe wartości estymatorów



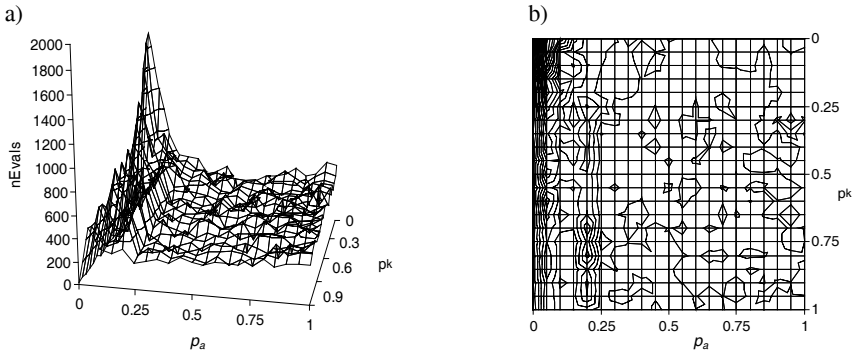
Rys. 70. Wpływ wielkości elity n_{elit} na dokładność i koszt indukcji
Fig. 70. Influence of elite size n_{elit} on accuracy and cost of induction

dokładności indukcji i jednocześnie niższe wartości kosztu indukcji dla $n_{elit} = 12$ (*fitness*$_{avg}$ = 98%, *positive*$_{avg}$ = 96,1%, *nEvals* = 209,68). Od wartości $n_{elit} = 23$ następuje gwałtowne pogorszenie wartości estymatorów aż do utarty przez model zdolności uczenia, począwszy od $n_{elit} = 29$. Interpretując otrzymane wyniki, należy pamiętać, że $n_{elit} = 23$ oznacza 7 produkcji, które podlegają w każdym kroku ewolucyjnym operacjom genetycznym. W zestawieniu z 6 produkcjami gramatyki wzorcowej lub 5 produkcjami gramatyki o minimalnej liczbie produkcji bez symbolu uniwersalnego (patrz podrozdz. 5.6.5) nie jest to liczba zapewniająca prawidłowy nacisk selektywny. Dla $n_{elit} = 29$ tylko jedna produkcja z całej populacji w każdym kroku ewoluuje, co w połączeniu z losowym doborem początkowych produkcji praktycznie uniemożliwia eksplorację rozwiązań.

5.6.7. Krzyżowanie i mutacja

W każdej metodzie ewolucyjnej operatory genetyczne krzyżowania i mutacji pełnią dominującą rolę. Nie inaczej jest w przypadku modelu GCS, który dostosowuje działanie operatorów genetycznych do reprezentacji osobnika, którym jest produkcja bezkontekstowej gramatyki w postaci normalnej Chomsky'ego (patrz **procedure Algorytm genetyczny**, rys. 30). Aby zbadać rolę obydwu operatorów, w modelu wykonano szereg

symulacji, podczas których zmieniano w przedziale [0, 1] wartość prawdopodobieństwa krzyżowania p_k oraz wartość prawdopodobieństwa mutacji p_m. Badania przeprowadzono dla domyślnych wartości parametrów. Wyniki przedstawiono na rys. 71–74.

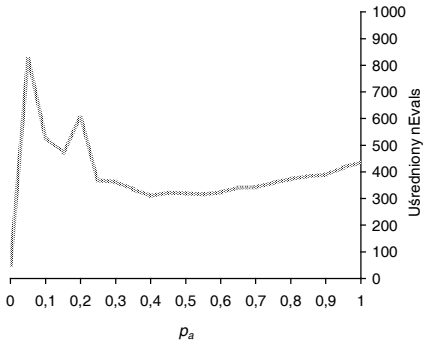


Rys. 71. Zmiana kosztu indukcji *nEvals* w zależności od prawdopodobieństwa krzyżowania p_k oraz prawdopodobieństwa mutacji p_a: a) wykres przestrzenny, b) wykres poziomicowy (poziomica co 100 kroków)
Fig. 71. Change of induction cost *nEvals* depending on crossover probability p_k and mutation probability p_a: a) 3D graph, b) contour graph (contour line every 100 steps)

Na rysunku 71 przedstawiono zmianę wartości kosztu indukcji *nEvals* w funkcji zmiany prawdopodobieństwa mutacji i krzyżowania. W powstałym krajobrazie można wyróżnić kilka charakterystycznych obszarów:

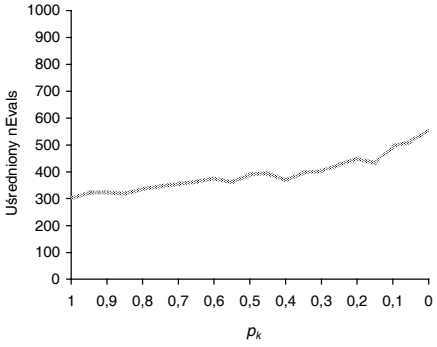
- skok wartości kosztu indukcji dla obszaru ograniczonego przez $p_a \in$ <0,05, 0,1> oraz $p_k \in$ [0, 0,25]. Najwyższą wartość *nEvals* = 1726,65 indukcja przyjmuje dla $p_a = 0{,}05$ i $p_k = 0$. Obszar najwyższych wartości jest ograniczony stromym zboczem wzdłuż $p_a = 0$;
- dla $p_a = 0$ oraz $p_k = 0$ i $p_k = 0{,}05$ estymator nie ma przypisanej wartości, gdyż proces uczenia przestaje być możliwy;
- najniższe wartości estymator przyjmuje dla $p_a = 0$ i $p_k > 0{,}05$. Zestawiając jednak ten obszar rysunku z odpowiednim obszarem na rys. 74, można stwierdzić, że dla wskazanych wartości prawdopodobieństw proces indukcji uzyskuje najniższe, niewiele ponad pięćdziesięcioprocentowe wartości estymatora $fitness_{\text{avg}}$. Oznacza to, że jedynie nieliczne iteracje zakończyły się sukcesem;
- charakterystyczna jest dolina niższych wartości estymatora wzdłuż $p_a = 0{,}15$ i $p_k > 0{,}5$ ograniczona po obu stronach wyższymi zboczami;
- wyższe wartość estymatora widoczne są również w obszarze wysokich wartości prawdopodobieństwa mutacji $p_a > 0{,}95$;
- pomiędzy dwoma obszarami bardzo wysokich wartości estymatora kosztu dla $p_a < 0{,}3$ i niższych dla $p_a > 0{,}95$ (patrz rys. 72) znajduje się obszar niskich wartości, wśród których najniższą wartość *nEvals* = 226,3 ma punkt o współrzędnych $p_a = 0{,}5$ i $p_k = 0{,}8$.

Krajobraz wartości estymatora kosztu uzupełniają rys. 72–73, na których wykreślono uśrednione wartości estymatora w funkcji zmiany odpowiednio prawdopodobieństwa mutacji oraz prawdopodobieństwa krzyżowania. Każdy punkt obydwu wykresów jest odpowiednią średnią liczoną z eksperymentów, podczas których zmieniano wartość p_k przy stałej p_a (rys. 72) oraz zmieniano wartość p_a przy stałej p_k (rys. 73).



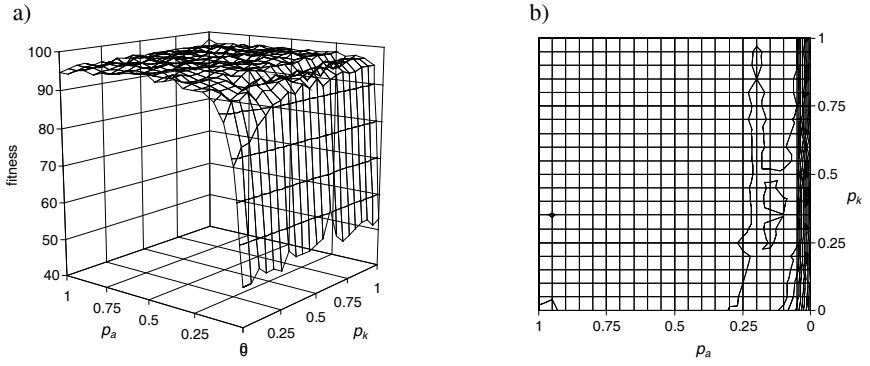
Rys. 72. Wykres uśrednionego kosztu indukcji *nEvals* w funkcji zmiany prawdopodobieństwa mutacji p_a; *nEvals* został uśredniony po prawdopodobieństwie krzyżowania p_k zmienianym w przedziale [0, 1]

Fig. 72. Graph of average induction cost *nEvals* as a function of mutation probability p_a; *nEvals* was averaged over crossover probability p_k changed in range [0, 1]



Rys. 73. Wykres uśrednionego kosztu indukcji *nEvals* w funkcji zmiany prawdopodobieństwa krzyżowania p_k; *nEvals* został uśredniony po prawdopodobieństwie mutacji p_a zmienianym w przedziale [0, 1]

Fig. 73. Graph of average induction cost *nEvals* as a function of crossover probability p_k; *nEvals* was averaged over mutation probability p_a changed in range [0, 1]



Rys. 74. Zmiana kompetencji ogólnej $fitness_{avg}$ w zależności od prawdopodobieństa krzyżowania p_k oraz prawdopodobieństwa mutacji p_a: a) wykres przestrzenny, b) wykres poziomicowy (poziomica co 5%)

Fig. 74. Change of general competence $fitness_{avg}$ depending on crossover probability p_k and mutation probability p_a: a) 3D graph b) contour graph (contour line every 5%)

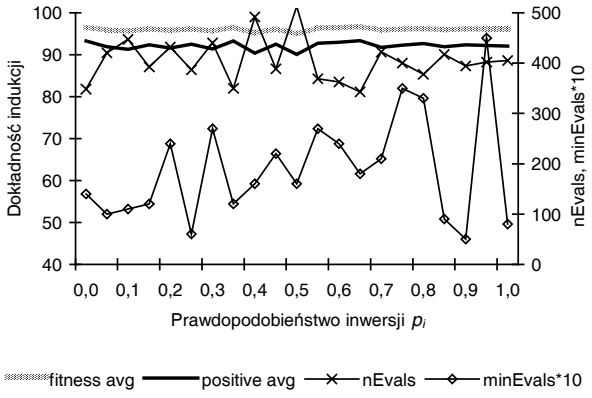
Obydwa rysunki wraz z rys. 71 potwierdzają poczynione wcześniej spostrzeżenie (patrz podrozdz. 5.3), że mutacja w modelu GCS, podobnie jak w innych systemach ewolucyjnych o niedużych populacjach, ma dominującą rolę. To zmiana prawdopodobieństwa mutacji bardzo szybko wyprowadza wartości estymatora *nEvals* z obszaru wysokich wartości lub też, patrząc na to zjawisko z drugiej strony, bardzo szybko wprowadza estymator w rejon wyższych wartości, podczas gdy zmiana prawdopodobieństwa krzyżowania, a dokładnej jego wzrost, powoduje zdecydowanie wolniejsze opuszczanie takich obszarów.

Na rysunku 72 widoczne jest charakterystyczne obniżenie wartości estymatora dla $p_a = 0{,}15$ oraz powolny jego wzrost w przedziale $p_a \in [0{,}35, 1]$.

Na rysunku 74 przedstawiono wpływ prawdopodobieństwa mutacji i krzyżowania na wartość estymatora dokładności $fitness_{avg}$. W krajobrazie można wyróżnić dwa obszary rozgraniczone prostą $p_a = 0{,}25$. Dla wyższych wartości prawdopodobieństwa mutacji wartość syntetycznej kompetencji ogólnej jest w całym obszarze wysoka i niezależna od parametrów p_a i p_k. Dla prawdopodobieństwa mutacji $p_a < 0{,}25$ następuje gwałtowne załamanie powierzchni tworzonej przez wartości $fitness_{avg}$, aż do osiągnięcia najniższych wartości wyznaczanych przez $p_a = 0$. Oznacza to, że dla zerowego prawdopodobieństwa mutacji tylko kilku iteracjom na 50 udaje zakończyć się odnalezieniem gramatyki. Jednak te nieliczne kończone sukcesem iteracje trwają wyjątkowo krótko (por. rys. 72 – wartość uśrednionego parametru *nEvals* równego 50,48 kroków ewolucyjnych dla $p_a = 0$).

5.6.8. Inwersja

Do operatorów genetycznych stosowanych przez model GCS zalicza się również inwersja. Operator ten wprowadzono do modelu ze względu na jego prawdopodobne niezaburzanie związków epistatycznych pomiędzy produkcjami. Na rysunku 75 przedstawiono wpływ zmiany wartości prawdopodobieństwa inwersji p_i na dokładność i koszt indukcji.



Rys. 75. Wpływ prawdopodobieństwa inwersji p_i na dokładność i koszt indukcji
Fig. 75. Influence of inversion probability p_i on accuracy and cost of induction

Obserwacja wykreślonych krzywych skłania do wniosku, że operator inwersji nie ma większego wpływu na proces indukcji gramatyki. W całym badanym przedziale zmienności prawdopodobieństwa operatora nie widać istotnych zmian w przebiegu wartość estymatorów $fitness_{avg}$, $positive_{avg}$ oraz *nEvals*. Krzywe oscylują wokół średnich wartości z charakterystycznymi dwoma wyższymi wartościami *nEvals* = 492,04 oraz *nEvals* = 512,8 dla prawdopodobieństwa iteracji równego odpowiednio 0,4 oraz 0,5. Brak istotnego wpływu operatora inwersji na proces uczenia można tłumaczyć tym, że zamiana kolejności symboli nieterminalnych po prawej stronie produkcji w istocie nie powoduje znaczących zmian w ewolucji poza zamianą kolejności wyprowadzanych z każdego symbolu nieterminalnego poddrzew. Dosyć charakterystyczną krzywą ma estymator *minEvals*, który na wykresie umieszczono z dziesięciokrotnie powiększonymi wartościami. Zwraca uwagę wzrastająca wraz z wartością prawdopodobieństwa p_i amplituda zmian najmniejszej liczby kroków, w których proces znajduje poprawne rozwiązanie, z zauważalną tendencją wzrostową linii trendu. Świadczyć to może o losowym wpływie inwersji symboli na „szybkie" znajdywanie gramatyki; zamiana miejsc symboli po prawej stronie reguły ten proces może radykalnie przyśpieszyć – co ilustrują gwałtowne spadki krzywej, lub też opóźnić. Inwersję można traktować w takich sytuacjach jako makromutacje.

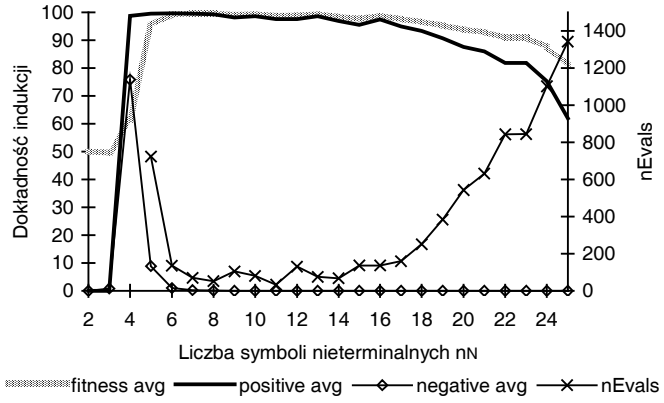
5.6.9. Liczba symboli nieterminalnych

Jednym z podstawowych parametrów modelu GCS jest liczba symboli nieterminalnych n_N. Parametr ten wskazuje na maksymalną możliwą liczbę różnych symboli nieterminalnych, które mogą zostać użyte przez model w trakcie ewolucji. Podczas implementacji modelu przyjęto zasadę, że ostatni dopuszczalny symbol nieterminalny oznaczać będzie symbol startowy gramatyki. W związku z tym w dotychczasowych eksperymatach stosowano $n_N = 19$ z tego względu, że dziewiętnastym symbolem alfabetu bez polskich znaków diakrytycznych jest symbol *S*. W programie gcs założono również, że jeżeli podczas indukcji włączony jest operator pokrycia startowego, to symbolem uniwersalnym jest przedostatni symbol w dopuszczalnym zbiorze symboli nieterminalnych (dla $n_N = 19$ to symbol *R*). Na rysunku 76 zobrazowano wyniki zbioru eksperymentów, w czasie których badano wpływ liczby symboli nieterminalnych n_N na proces indukcji gramatyki. Wszystkie doświadczenia przeprowadzano dla domyślnego zestawu parametrów.

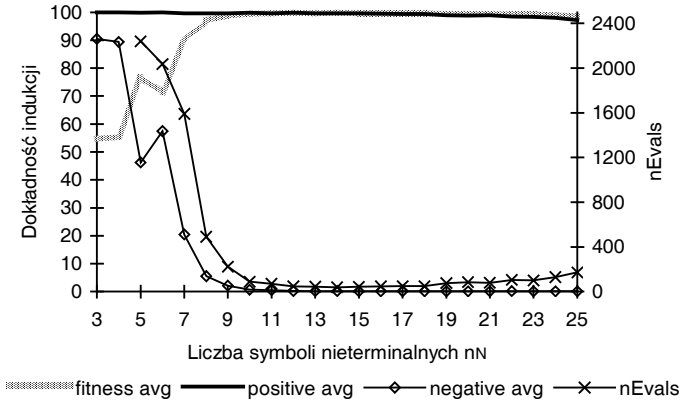
W wykresach estymatorów $fitness_{avg}$, $positive_{avg}$, $negative_{avg}$ oraz *nEvals* można wyróżnić charakterystyczne obszary:

- brak zdolności uczenia dla $n_N < 5$,
- opadające linie wykresów dla estymatorów $fitness_{avg}$ oraz $positive_{avg}$ dla $n_N > 16$,
- „siodło" linii wykresu estymatora *nEvals*; najniższe wartości estymator uzyskuje dla n_N z przedziału [6, 17], dla niższych i wyższych wartości liczby symboli nieterminalnych wartość estymatora kosztu gwałtownie rośnie.

Jeśli nie jest zaskoczeniem brak zdolności modelu do uczenia dla liczby symboli nieterminalnych mniejszej od 5 (wynika to wprost z minimalnej liczby produkcji indukowanej gramatyki TOY), to zastanawiać może wzrost kosztu uczenia połączony ze zmniejszaniem dokładności uczenia dla wyższych wartości parametru n_N ($n_N > 16$). Podobnie, jak to miało miejsce podczas analizy zależności procesu uczenia od wielkości elity, można stwierdzić, że powyżej określonej wartości parametru n_N zostają zachwiane prawidłowe proporcje pomiędzy eksploatacją a eksploracją modelu, na korzyść coraz silniejszej eksploracji, o czym świadczyć może wzrastający okres dochodzenia do prawidłowego rozwiązania. Zwiększona liczba symboli nieterminalnych powoduje wzrost liczby możliwych drzew rozbiorów, a co za tym idzie wzmocnienie eksploracji rozwiązań.



Rys. 76. Wpływ liczby symboli nieterminalnych n_N na dokładność i koszt indukcji
Fig. 76. Influence of number of nonterminal symbols n_N on accuracy and cost of induction



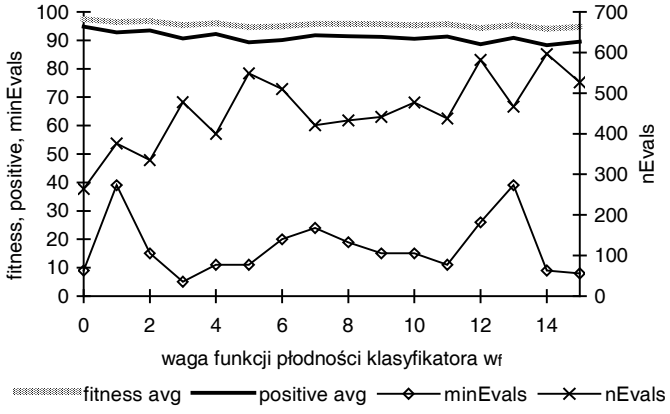
Rys. 77. Wpływ liczby symboli nieterminalnych n_N na dokładność i koszt indukcji – włączony operator pokrycia uniwersalnego
Fig. 77. Influence of number of nonterminal symbols n_N on accuracy and cost of induction with don't care covering on

Na rysunku 77 wykreślono wyniki eksperymentów przeprowadzonych przy włączonym operatorze pokrycia uniwersalnego. Porównując wykresy wartości estymatorów z operatorem uniwersalnym (rys. 77) oraz bez tego operatora (rys. 76), można stwierdzić następujące różnice:

- model z operatorem wraz ze wzrostem liczby symboli nieterminalnych zdecydowanie wolniej traci wysoką efektywność uczenia, o czym świadczy bardzo powoli opadająca linia estymatorów dokładności i równie powoli wznosząca się linia estymatora kosztu;
- nieznaczny spadek efektywności występuje w przypadku większej liczby symboli nieterminalnych ($n_N > 19$) niż eksperymentów bez symbolu uniwersalnego ($n_N > 16$);
- model jest w stanie wyuczyć się gramatyki dla 5 symboli nieterminalnych, chociaż gramatyka charakteryzuje się jeszcze wysoką wartością uśrednionej kompetencji negatywnej (rzędu 50%). Wzrost liczby symboli nieterminalnych powoduje radykalny spadek wartości estymatorów *nEvals* oraz $negative_{\text{avg}}$, które dla $n_N = 10$ osiągają swoje wartości minimalne.

5.6.10. Płodność

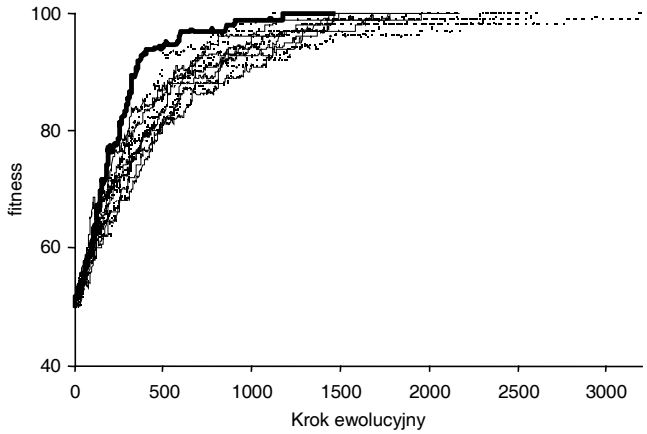
Zadaniem płodności w modelu GCS jest uwzględnianie w funkcji przystosowania produkcji jej pozycji w ciągu wyprowadzeń. Im wcześniej w danym ciągu wyprowadzeń jest produkcja, tym jest bardziej „płodna". Definicja miary przystosowania produkcji umożliwia uwzględnienie położenia produkcji w ciągu, poprzez przypisywanie



Rys. 78. Wpływ wagi funkcji płodności w_f na dokładność i koszt indukcji ($w_p = 1$, $w_n = 2$, $w_c = 1$, $f_0 = 0{,}5$)
Fig. 78. Influence of fertility weight w_f on accuracy and cost of induction ($w_p = 1$, $w_n = 2$, $w_c = 1$, $f_0 = 0.5$)

bardziej płodnym produkcjom dodatkowej wartości użyteczności (reprezentowanej przez składową f_f we wzorze na całkowite przystosowanie produkcji (32)). Dzięki takiemu zabiegowi można wymusić na procesie indukcji gramatyki – poprzez odpowiednie sterowanie wagą funkcji płodności w_f – aby w trakcie ewolucji produkcje bardziej płodne były mniej narażone na usunięcie lub tylko modyfikację. Na rysunku 78 przedstawiono zależność estymatorów $fitness_{\text{avg}}$, $positive_{\text{avg}}$, *minEvals* oraz *nEvals* od zmiany wartości parametru w_f.

Wzrost wartości wagi funkcji płodności powoduje zauważalny wzrost średniej liczby kroków ewolucyjnych, w których model dochodzi do rozwiązania. Równocześnie nieznacznie opadają krzywe estymatorów dokładności kompetencji ogólnej i pozytywnej. Nie jest obserwowalna jakaś istotna zmiana minimalnej liczby kroków *minEvals* wraz ze zmianą w_f. Wnioski, które można wyciągnąć, analizując syntetyczne miary procesu indukcji, potwierdzane są również przez wiązkę krzywych zbieżności kompetencji ogólnej *fitness*, wykreślonych dla $w_f \in [1, 15]$, rys. 79.



Rys. 79. Zmiana kompetencji ogólnej *fitness* dla różnych wag funkcji płodności – linia pogrubiona dla $w_f = 0$, linie przerywane dla $w_f \in <1, 15>$ ($w_p = 1$, $w_n = 2$, $w_c = 1$, $f_0 = 0{,}5$)
Fig. 79. Change of general competition fitness for different fertility weights – solid line for $w_f = 0$, dashed lines for $w_f \in <1, 15>$ ($w_p = 1$, $w_n = 2$, $w_c = 1$, $f_0 = 0.5$)

Krzywa dla wyłączonej w procesie indukcji płodności została wykreślona pogrubioną linią. Zauważalny jest bardziej stromy przebieg krzywej dla $w_f = 0$ od przebiegu krzywych, w których obliczana jest płodność produkcji. Wzrost wartości w_f generalnie zmniejsza nachylenie krzywej zbieżności. Najwyższe wartości *maxEvals* – czyli te, które najdłużej dochodziły do gramatyki zgodnej ze zbiorem uczącym – eksperymenty dla $w_f = 14$ (*maxEvals* = 2883) i $w_f = 10$ (*maxEvals* = 3183).

Wytłumaczenia „tłumiących" proces indukcji własności mechanizmu płodności należy poszukiwać w… mechanizmie ścisku. Co prawda zwiększone wartości funkcji

przystosowania płodnych reguł zwiększają szanse ich wyboru przez operatory genetyczne, to jednak za umieszczenie genetycznie modyfikowanych produkcji w zbiorze produkcji gramatyki odpowiada właśnie ścisk. Mechanizm ścisku dobiera do zastąpienia produkcje najsłabsze z wylosowanej wcześniej podpopulacji, w związku z czym produkcje płodne, dodatkowo punktowane, mają mniejsze prawdopodobieństwo, aby zostać wymienione. Powodować to może rosnącą wraz z wagą płodności reguł stagnację populacji, „osiadanie w minimum lokalnym”[120], a w konsekwencji spadek dokładności indukcji i wzrost jej kosztu.

Przeprowadzono również eksperymenty, w których badano wpływ zmiany parametrów *ba* i *raf* na proces indukcji w zakresie [0, 15]. Nie została zaobserwowana żadna istotna zmiana wartości estymatorów indukcji podczas zmiany wartości tych parametrów.

5.6.11. Wagi funkcji dopasowania produkcji

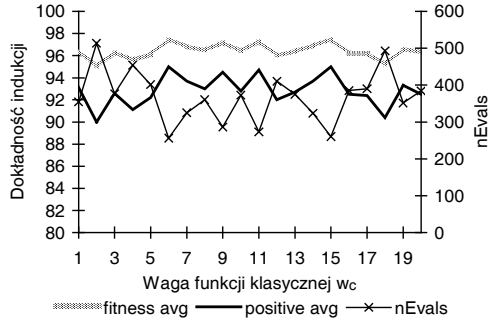
Badana w poprzednim rozdziale waga funkcji płodności w_f jest jedną ze składowych wzoru na przystosowanie produkcji (patrz wzór (32)). Drugą składową jest tzw. funkcja klasyczna f_c, która wchodzi do ogólnego wzoru ze współczynnikiem w_c. W funkcji f_c występują natomiast wagi w_p i w_n, które określają wpływ rozbioru zdania pozytywnego i negatywnego na wartość funkcji, wzór (33). Jeżeli klasyfikator nie bierze udziału w parsowaniu zdania, to otrzymuje stałą wartość f_0.

Na rysunku 80 przedstawiono zależność dokładności i kosztu indukcji od wartości wagi funkcji klasycznej dla $w_p = 1$, $w_n = 2$, $f_0 = 0{,}5$. Waga funkcji płodności została ustawiona na średnią wartość $w_f = 8$. Trudno doszukać się w przebiegu krzywych zbieżności badanych estymatorów $fitness_{\text{avg}}$, $positive_{\text{avg}}$ i *nEvals* zależności od wartości wagi w_c, może poza nieco niższymi wartościami kosztu indukcji i wyższymi wartościami dokładności dla $w_c \in [6, 15]$. Tę niezależność procesu indukcji od wagi funkcji klasycznej można wyjaśnić, odwołując się do rys. 78, ilustrującego wpływ wagi funkcji płodności w_f na dokładność i koszt indukcji. Jak zauważono w rozdziale poprzednim, wzrost wartości współczynnika w_f pociąga za sobą co prawda wzrastający koszt indukcji i malejącą dokładność, jednak ta zmiana nie jest gwałtowna. Świadczy to o niewielkim wpływie składowej f_f na wartość funkcji f, co w konsekwencji przekłada się na niezależność funkcji przystosowania produkcji f od wagi w_c.

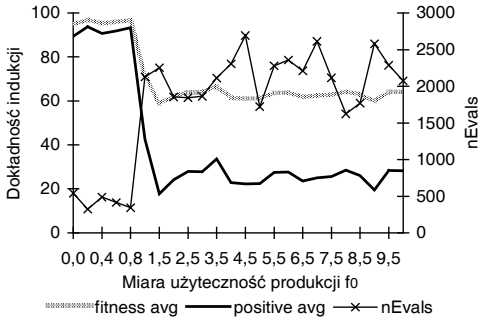
W kolejnej serii eksperymentów zbadano wpływ miary użyteczności reguły, która nie bierze udziału w rozbiorze zdania na proces indukcji. Uzyskane wyniki przedstawiono na rys. 81. Wykres ten wskazuje wyraźnie na optymalny przedział wartości parametru f_0, który nie przekracza wartości 1. Powyżej tej wartości proces indukcji bardzo wyraźnie zwiększa koszty indukcji gramatyki, skacząc z wartości oscylujących

[120] Tak naprawdę trudno mówić w przypadku ewolucyjnego poszukiwania gramatyki o minimach czy też maksimach lokalnych.

w przedziale 300–500 kroków do wartości z przedziału 1800–2600. Również dla $f_0 > 0{,}8$ odnotować można istotny spadek wartości estymatorów dokładności, w przypadku estymatora $fitness_{\text{avg}}$ średnio o około 30%.



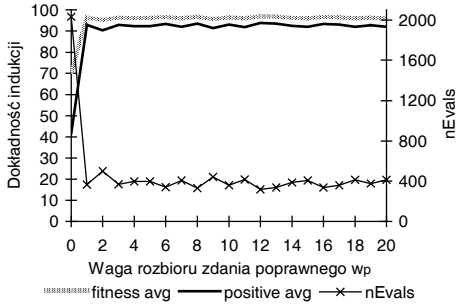
Rys. 80. Wpływ wagi funkcji klasycznej w_c na dokładność i koszt indukcji ($w_p = 1$, $w_n = 2$, $f_0 = 0{,}5$, $w_f = 8$)
Fig. 80. Influence of classic function weight w_c on accuracy and cost of induction ($w_p = 1$, $w_n = 2$, $f_0 = 0.5$, $w_f = 8$)



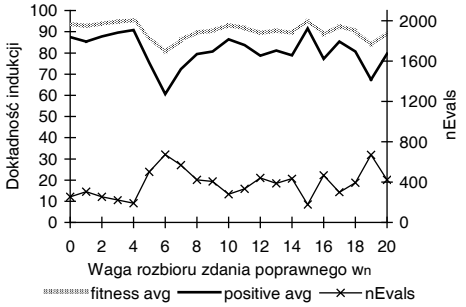
Rys. 81. Wpływ miary użyteczności klasyfikatora niebiorącego udziału w parsowaniu f_0 na dokładność i koszt indukcji ($w_p = 1$, $w_n = 2$, $w_c = 1$, $w_f = 0$)
Fig. 81. Influence of fitness of classifier not used in parsing f_0 on accuracy and cost of induction ($w_p = 1$, $w_n = 2$, $w_c = 1$, $w_f = 0$)

Następna seria eksperymentów miała dać odpowiedź na pytanie, na ile proces indukcji gramatyki jest wrażliwy na zmianę wartość wag rozbioru zdań pozytywnych i negatywnych. Rezultaty badań zamieszczono na rys. 82 i rys. 83. Symulacje przeprowadzono przy wyłączonej płodności produkcji ($w_f = 0$) oraz zrównoważonym zbiorze uczącym składającym się 20 zdań pozytywnych i 20 zdań negatywnych. Analizując wykres zbieżności krzywych estymatorów w funkcji zmiany wagi rozbioru zdania poprawnego w_p, zauważalny jest skok wartości estymatorów przy zmianie parametru z wartości 0 na 1. Dla wartości $w_p = 0$ estymatory mają następujące wartości: $fitness_{\text{avg}} = 70{,}8\%$, $positive_{\text{avg}} = 41{,}7\%$, $nEvals = 2032{,}97$. Świadczą one o wyraźnych trudnościach, jakie proces indukcji napotyka podczas ewoluowania poprawnej gramatyki przy jednoczesnym nieuwzględnianiu rozbioru zdań pozytywnych. Z 50 iteracji eksperymentu 35 kończy się sukcesem. Dla $w_p > 1$ krzywe estymatorów oscylują wokół średnich wartości, bez żadnej wyraźnej tendencji wzrostowej czy też malejącej, i z bardzo niewielką amplitudą. Dla $fitness_{\text{avg}}$ średnia dopasowania w badanym przedziale zmienności wynosi 95% ze średnim odchyleniem 2,2%, dla $positive_{\text{avg}}$ średnia to 90,1% ze średnim odchyleniem 4,4%, wreszcie dla *nEvals* średnia wynosi 463 kroki ze średnim odchyleniem 146 kroków. Należy jednak pamiętać o znacznych różnicach w wartościach estymatorów dla $w_p = 0$. Nie uwzględniając wartości estymatorów dla $w_p = 0$, otrzymujemy następujące odpowiednie średnie: 96,3% ±0,3 ($fitness_{\text{avg}}$), 92,6% ±0,7 ($positive_{\text{avg}}$) oraz 384 ±34 (*nEvals*).

Nieco inaczej wygląda przebieg tych samych badanych estymatorów $fitness_{avg}$, $positive_{avg}$, *nEvals* w funkcji zmiany wagi rozbioru zdań negatywnych (rys. 83).



Rys. 82. Wpływ wagi rozbioru zdania poprawnego w_p na dokładność i koszt indukcji ($w_n = 2$, $w_c = 1$, $f_0 = 0{,}5$, $w_f = 0$)
Fig. 82. Influence of weight of proper sentence parsing w_p on accuracy and cost of induction ($w_n = 2$, $w_c = 1$, $f_0 = 0.5$, $w_f = 0$)



Rys. 83. Wpływ wagi rozbioru zdania niepoprawnego w_n na dokładność i koszt indukcji ($w_p = 1$, $w_c = 1$, $f_0 = 0{,}5$, $w_f = 0$)
Fig. 83. Influence of weight of improper sentence parsing w_n on accuracy and cost of induction ($w_p = 1$, $w_c = 1$, $f_0 = 0.5$, $w_f = 0$)

Przede wszystkim krzywe estymatorów kompetencji oscylują wokół niższych wartości niż na rys. 82 i ze znacznie większą amplitudą; krzywa estymatora $fitness_{avg}$ wokół średniej 90,5% ze średnim odchyleniem 2,7%, a krzywa estymatora $positive_{avg}$ wokół średniej 81% z odchyleniem 5,4%. Nie obserwuje się również żadnej różnicy wartości badanych estymatorów dla $w_n = 0$. Dla $w_n > 18$ zarysowuje się natomiast zauważalny spadek dokładności indukcji oraz wzrost kosztu indukcji.

6. Zastosowanie modelu GCS w genomice

Genomika jest jednym z obszarów bioinformatyki, koncentrującym się na modelach i narzędziach zorientowanych na odczytywanie i analizę sekwencji kwasów nukleinowych. Jeżeli przyjąć, że sekwencję kwasów nukleinowych można opisać formalnym językiem, to sekwencjonowanie biosekwencji za pomocą metod lingwistycznych można wykorzystać do:

- edycji sekwencji,
- poszukiwania miejsc restrykcyjnych,
- projektowania starterów i sond oligonukletydowych,
- analizowania składu nukleotydowego i aminokwasowego,
- wyznaczania punktu izoelektrycznego białka,
- poszukiwania rejonów kodujących, granic intron-ekson,
- translacji, poszukiwania otwartej ramki odczytu,
- analizie charakteru poszczególnych części sekwencji białka,
- przewidywania struktury dwu- i trzeciorzędowej białek i kwasów nukleinowych lub ich fragmentów,
- tworzenia interfejsów z biologicznymi bazami danych.

Zadaniem modelu GCS jest ewolucyjna indukcja gramatyki bezkontekstowej, którą z kolei można opisywać sekwencję DNA czy RNA (Searls 1993, 2002). Możliwości zastosowania modelu w biologii obliczeniowej zostaną przedstawione na przykładzie zadania rozpoznawania sekwencji telomerowej u człowieka oraz poszukiwania regionu promotorowego u bakterii *E. coli*.

6.1. Rozpoznawanie sekwencji telomerowej

6.1.1. Preliminaria biologiczne

Telomer to element strukturalny chromosomu występujący na jego końcach, a zatem w każdej komórce człowieka występują w sumie 92 telomery (w rozdziale wykorzystano artykuł *Telomer (genetyka)* w Wikipedii http://pl.wikipedia.org/wiki/Telomer_genetyka, licencja: CC-BY-SA 3.0, autorzy: http://pl.wikipedia.org/w/index.php?title=Telomer_

(genetyka)&action=history). Sekwencje telomerowe nie są przekazywane w sposób genetyczny, lecz są enzymatycznie dobudowywane. Telomer zbudowany jest z kilku tysięcy zasad nukleinowych i związanych z nimi białek. Sekwencja składająca się na telomer człowieka zbudowana jest z nukleotydów: TTAGGG (gdzie T oznacza tyminę, A – adeninę, G – guaninę). Telomer nie zawiera żadnych genów i nie koduje żadnych białek. Zasady nukleinowe na końcu telomeru ułożone są na kształt „koniczyny", zawierającej dużą liczbę guaniny. Sekwencja nukleotydów w telomerze jest niezmienna i powtarzalna. U ludzi i innych kręgowców sekwencja ta jest taka sama. Sekwencje telomerowe są ewolucyjnie konserwatywne i wyglądają bardzo podobnie u wielu gatunków zwierząt i roślin (patrz tab. 20). W miarę posuwania się w stronę geometrycznego środka chromosomu sekwencja zaczyna ulegać subtelnym zmianom – to tzw. obszar subtelomerowy zawierający zarówno sekwencje niekodujące, jak i kodujące. Sekwencje występujące w obszarze subtelomerowym mogą stopniowo coraz mniej przypominać podstawową. Zamiast powtórzeń TTAGGG mogą pojawiać się podjednostki takie, jak: TAGGG, TTTGGG, TTAAGG itp. Telomer i obszar subtelomerowy tworzą wspólnie tzw. końcowy fragment restrykcyjny (*terminal restriction fragment*, TRF). W miarę posuwania w stronę środka chromosomu sekwencje stają się coraz bardziej różnorodne, aż stają się unikatowe i bardzo złożone – zaczynają kodować białka. Są to pierwsze geny, tzw. geny okołotelomerowe. Długość telomeru zależy od wieku organizmu. Telomery skracają się podczas każdego cyklu replikacji DNA komórki. Niektóre komórki (np. komórki płciowe, z których powstają plemniki i komórki jajowe) zawierają telomerazę – enzym, który potrafi odbudowywać telomery. Dzięki temu długość telomerów nie zmniejsza się z pokolenia na pokolenie. Gdyby komórki płciowe nie wydłużały telomerów swoich chromosomów, po upływie kilku pokoleń nowe organizmy stałyby się bezpłodne, a ich pozbawione telomerów chromosomy sklejałyby się ze sobą, co zwiększałoby częstość szkodliwych mutacji.

Tabela 20. Sekwencje telomerowe u różnych gatunków zwierząt i roślin

Organizm	Sekwencje powtórzone (5′–3′)
Człowiek, mysz	TTAGGG
Pantofelek *Paramecium*	TTGGG(T/G)
Plasmodium	TTAGGG(T/C)
Arabidopsis	TTTAGGG
Jedwabnik	TTAGG
Glista ludzka	TTAGGC
Chlamydomonas	TTTTAGGG
Drożdże piekarnicze	TTAC(A)(C) G(1–8)

Telomer ma cztery zasadnicze funkcje:

1) ochronę końca chromosomu przed uszkodzeniem lub nieprawidłową rekombinacją,
2) umożliwienie całkowitej replikacji chromosomu,

3) nadzorowanie ekspresji genów,

4) wspomaganie organizacji chromosomów w trakcie podziałów komórki.

Dodatkowo telomer spełnia rolę zegara komórkowego. Być może również ułatwia ustawianie się parami chromosomów homologicznych. Dwie pierwsze funkcje są niezbędne dla bezbłędnego dziedziczenia. Pozostałe umożliwiają kontrolowany dostęp do genów. W związku z problemem replikacji końca kluczowe znaczenie ma funkcja druga. Bez systematycznego skracania telomerów redukcji musiałby ulegać obszar kodujący, a to oczywiście prowadziłoby do zniszczenia i utraty genów. Trzecia funkcja – nadzorowanie ekspresji genów leżących w pobliżu końca chromosomu nie została jeszcze w pełni zbadana. Faktem jest, że skracanie telomeru wywołuje zmiany ekspresji genów.

6.1.2. Indukcja gramatyki „telomerowej"

Model GCS można zastosować do indukcji gramatyki „telomerowej" – gramatyki rozpoznającej sekwencje telomerowe występujące u człowieka. Taka sekwencja nukleotydowa może być opisana wyrażeniem regularnym $(TTAGGG)^+$, gdzie litery T, A i G stanowią litery alfabetu poszukiwanego języka.

Podczas uczenia zastosowano zestaw parametrów modelu, który poza mniejszą liczbą iteracji ($n_{run} = 10$) nie różni się od zestawu stosowanego podczas badań symulacyjnych.

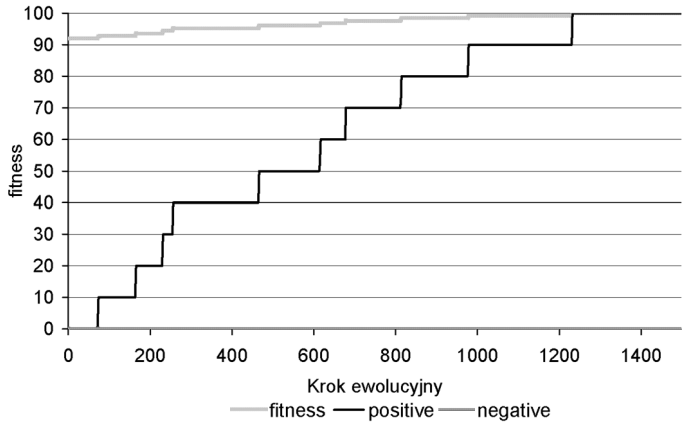
Zbiór uczący U składał się ze 100 sekwencji uczących, w tym 8 przykładów sekwencji telomerowych, o długościach odpowiednio: 6, 12, 18, 24, 30, 36, 42, 48 nukleotydów, i 92 przykładów sekwencji DNA wygenerowanych losowo i niezawierających ciągu telomerowego.

Podczas wszystkich 10 iteracji model znajdował gramatykę zgodną ze zbiorem uczącym, uzyskując następujące wartości estymatorów: $fitness_{\text{avg}} = 99{,}1\%$, $positive_{\text{avg}} = 89\%$, $nEvals = 552{,}2$, $minEvals = 78$, $maxEvals = 1233$. Na rysunku 84 wykreślono przebieg kompetencji ogólnej *fitness*, pozytywnej *positive* i negatywnej *negative* w funkcji kolejnych kroków ewolucyjnych.

Krzywa estymatora fitness przez 72 kroki ewolucyjne utrzymuje się na poziomie 92% przy jednoczesnym braku akceptacji sekwencji pozytywnych. Oznacza to, że gramatyka pierwszych kroków ewolucyjnych podczas wszystkich iteracji nie jest w stanie poprawnie sparsować przykładów pozytywnych, ale równocześnie nie parsuje przykładów negatywnych. Krzywa kompetencji pozytywnej układa się w charakterystyczne schodki. Skoki zmieniają wartość estymatora *positive* o 10%.

W wyniku uczenia model znalazł gramatykę zawierającą 4 produkcje nieterminalne i 3 produkcje terminalne wyprowadzające symbole terminalne t, g i a, odpowiadające tyminie, adeninie i guaninie:

1. $S \rightarrow SQ$
2. $S \rightarrow AG$
3. $Q \rightarrow AG$
4. $A \rightarrow TQ$
5. $A \rightarrow a$
6. $G \rightarrow g$
7. $T \rightarrow t$.



Rys. 84. Indukcja gramatyki „telomerowej"
Fig. 84. Telomer grammar induction

Pierwsza produkcja nieterminalna $S \rightarrow SQ$ odpowiedzialna jest za iterację ciągu *ttaggg*. Pozostałe produkcje nieterminalne wyprowadzają iterowany ciąg *ttaggg* w sposób następujący $S \rightarrow AG \rightarrow TQG \rightarrow TAGG \rightarrow TTQGG \rightarrow TTAGGG \rightarrow ttaggg$.

Do testów generalizacji gramatyki użyto zbioru testowego, składającego się z 65 534 sekwencji o długości nieprzekraczającej 200 symboli, w tym 17 sekwencji telomerowych i 65 517 sekwencji nietelomerowych. Uzyskano stuprocentowe przystosowanie gramatyki.

6.2. Poszukiwanie regionów promotorowych

6.2.1. Preliminaria biologiczne

Predykacja regionów promotorowych jest obecnie samoistnym problemem genomiki obliczeniowej (Handley 1995, Pedersen i in. 1999, Hannenhalli i Levy 2001, Ohler i Niemann 2001). Wynika to z faktu, iż sekwencja promotorowa jest decydująca w regulacji genowej, a ta z kolei zdaje się być jednym z najbardziej ekscytujących zagadnień biologii molekularnej. Promotor DNA można zdefiniować jako sekwencję nukleotydową DNA, do której przyłącza się polimeraza RNA, aby rozpocząć transkrypcję – czyli proces syntezy RNA na matrycy DNA. W organizmach prokariotycznych[121] transkrypcja regulowana jest tylko przez jedną polimerazę RNA, podczas gdy

[121] Organizmy prokariotyczne to organizmy jednokomórkowe, pozbawione jądra komórkowego, mitochondriów i cytoszkieletu; do grupy tej należą bakterie i sinice (cyjanobakterie) (Kofta 1999).

w organizmach eukariotycznych[122] istnieją aż trzy typy polimerazy RNA (Polimeraza RNA I, Polimeraza RNA II i Polimeraza RNA III), czyniąc proces ekspresji genów znacznie bardziej złożonym niż u prokarionów.

Klasycznym już dzisiaj zadaniem przewidywania regionów kodujących i niekodujących w sekwencjach DNA jest predykcja regionów promotorowych u prokariotycznej bakterii *E. coli* (pałeczki okrężnicy) (Towell i in. 1990, Handley 1995, Chen i in. 2002). W (Towell i in. 1990) zaproponowano zwarty zapis regionu promotorowego, wywiedziony na podstawie (Harley i Reynolds 1987). W tabeli 21 podano reguły opisujące możliwe formuły sekwencji promotorowych.

Tabela 21. Formalny zapis sekwencji promotorowych bakterii *E. coli*, na podstawie (Towell i in. 1990)

1.	promotor	:-kontakt, konformacja.
2.	kontakt	:-minus_35, minus_10.
3.	minus_35	:-@-37 „cttgac”.
4.	minus_35	:-@-36 „ttgxca”.
5.	minus_35	:-@-36 „ttgaca”.
6.	minus_35	:-@-36 „ttgac”.
7.	minus_10	:-@-14 „tataat”.
8.	minus_10	:-@-13 „taxaxt”.
9.	minus_10	:-@-13 „tataat”.
10.	minus_10	:-@-12 „taxxxt”.
11.	konformacja	:-@-45 „aaxxa”.
12.	konformacja	:-@-45 „axxxa”, @-4 „t”, @-28 „txxxtxaaxxtx”.
13.	konformacja	:-@-49 „axxxxt”, @-1 „a”, @-27 „txxxxaxxtxtg”.
14.	konformacja	:-@-47 „caaxttxac”, @-22 „gxxxtxc”, @-8 “gcgccxcc”.

Pierwsza reguła tabeli 21 oznacza, że sekwencja promotorowa zawiera dwa regiony: kontakt i konformację. Reguła druga definiuje region kontakt jako obszar składający się z dwóch regionów minus_35 oraz minus_10. Kolejne reguły podają alternatywne określenia odpowiednich regionów. Podana definicja sekwencji promotorowej zakłada, że tworzy ją 57 nukleotydów, z których 50 poprzedza gen promotorowy, a 6 występuje za nim. Zgodnie z przyjętą w literaturze biologicznej zasadą, lokalizację poszczególnych nukleotydów określa się względem miejsca rozpoczynania transkrypcji. W specyfikacji regionów symbol *x* oznacza dowolny nukleotyd (*a*, *c*, *t* lub *g*). Regułę 11 czyta się w sposób następujący: 45 nukleotydów, przed miejscem rozpoczęcia transkrypcji musi wystąpić *adenina*. Następnym nukleotydem również musi być *adenina*, po niej mogą wystąpić dwa dowolne nukleotydy. Na pozycji 41 musi pojawić się

[122] Organizmy eukariotyczne to organizmy jednokomórkowe i wielokomórkowe o złożonej strukturze wewnętrznej komórek, w których występuje jądro komórkowe, mitochondria i cytoszkielet (Kofta 1999).

znów *adenina*. Towell i in. (1990), na podstawie definicji sekwencji promotorowej zawartej w tab. 21, podali 53 przykłady poprawnych sekwencji promotorowych oraz 53 przykłady sekwencji, które nie tworzą regionu promotorowego[123]. Zbiór ten został zastosowany przez model GCS do indukcji gramatyki opisującej region promotorowy bakterii *E. coli*[124].

6.2.2. Indukcja gramatyki „promotorowej"

Podobnie jak miało to miejsce podczas uczenia gramatyki opisującej sekwencję telomorową, tak i w przypadku indukcji gramatyki rozpoznającej region promotorowy, zastosowano standardowy zestaw parametrów. Problem predykcji sekwencji promotorowej jest jednak znacznie bardziej złożony od poszukiwania gramatyki regularnej. Stąd też przyjęto, że przewidywany maksymalny rozmiar populacji $n_P = 150$, a liczba początkowych produkcji nieterminalnych $n_{start} = 130$. Eksperymenty prowadzono dla maksymalnie 5000 kroków ewolucyjnych ($n_{max} = 5000$) oraz przy 10 iteracjach ($n_{run} = 10$). Założono również, że poszukiwana jest najlepsza pod względem kompetencji ogólnej gramatyka, gdyż trudno, na podstawie doniesień literaturowych oraz analizy reguł tworzących region promotorowy (patrz tab. 21), oczekiwać indukcji gramatyki zgodnej w 100% ze zbiorem uczącym. Podczas 10 niezależnych iteracji eksperymentu uzyskano średnią najlepszą kompetencję językową $fitness_{max} = 74{,}53\%$, przy kompetencji pozytywnej $positive = 62{,}26\%$, kompetencji negatywnej $negative = 13{,}21\%$ oraz $Evals = 1221$.

Testy generalizacji indukowanej gramatyki przeprowadzono na ręcznie przygotowanym zbiorze testowym, składającym się z 18 przykładów pozytywnych i 18 przykładów negatywnych, wszystkie o długości 57 symboli. Zbiór testowy został przygotowany na bazie zbioru uczącego w ten sposób, że w losowo wybranych 18 sekwencjach promotorowych zmieniono symbole w miejscach nieistotnych dla rozpoznawania regionu, tworząc w ten sposób przykłady testowe pozytywne, a w losowo wybranych 18 sekwencjach niepromotorowych zmieniono symbole na dowolnych pozycjach, generując w ten sposób testowe przykłady niepoprawne[125]. Testy generalizacji przyniosy następujące rezultaty: $nGen = 77{,}80\%$, $nGen_{pos} = 61{,}10\%$ oraz $nGen_{neg} = 5{,}56\%$. Wyewoluowana gramatyka bardzo dobrze odrzuca sekwencje niepromotorowe, gorzej radzi sobie z parsowaniem sekwencji promotorowych.

Aby móc porównać uzyskane wyniki z doniesieniami literaturowymi, należy przekształcić estymatory modelu GCS. W pracy (Towell i in. 1990) używane jest pojęcie

[123] Warto zwrócić uwagę, że ciąg DNA nie jest sekwencją losową tworzących go nukleotydów. Z tego też powodu zbiór sekwencji niepromotorowych nie został przygotowany jako losowa permutacja ciągu 57-literowego, lecz jest fragmentem bakteriofaga T7 bakterii *E. coli*.

[124] Zbiór uczący dostępny jest na stronie http://www.irisa.fr/symbiose/people/coste/gi_benchs.html

[125] Opisana metoda tworzenia zbiorów uczących/testowych dla algorytmów rozpoznających regiony kodujące jest powszechnie stosowana przez bioinformatyków.

Patrz www.icgeb.res.in/~whotdr/to_port/presentations/chari-promoter-pred-30sep.ppt

Error Rate określające sumę niepoprawnie sklasyfikowanych przykładów z ogólnej liczby dostępnych wszystkich sekwencji. Ocena jakości rozpoznawania pojedynczej sekwencji wykonana była metodą *leave-one-out*[126]. Uzyskana podczas testów wartość *nGen* = 77,80% oznacza, że 24 przykłady ze 106 są niepoprawnie klasyfikowane. Towell i in. porównali jakość klasyfikacji 5 różnych metod. W tabeli 22 zestawiono rezultat osiągnięty przez indukowaną przez model GCS gramatykę bezkontekstową z wynikami hybrydowego algorytmu łączącego system regułowy z siecią neuronową KBANN (Towell i in. 1990), klasyczną siecią neuronową stosującą wsteczną propagację (SB), modelem uczenia ID3, metodą k najbliższych sąsiadów z $k = 3$ (kNN) oraz metodą dopasowania wzorców zastosowaną w (O'Neill 1989).

Tabela 22. Porównanie jakości klasyfikacji regionu promotorowego *E. coli*

Model	*Error Rate*
KBANN	4/106
SB	8/106
O'Neill	12/106
kNN	13/106
ID3	19/106
GCS	24/106

Model GCS uzyskał z porównywalnych metod najniższą jakość klasyfikacji, bliską algorytmowi drzew decyzyjnych ID3, chociaż należy pamiętać, że testowanie modelu nie odbywało się techniką *leave-one-out*, ale na zupełnie niezależnym zbiorze danych[127].

W literaturze bioinformatycznej stosuje się również dwie miary statystyczne *czułość Sn* (*sensitivity*) i *swoistość Sp* (*specificity*) opisane wzorami:

$$Sn = \frac{TP}{TP + FN}, \tag{60}$$

$$Sp = \frac{TN}{TN + FP}, \tag{61}$$

[126] W metodzie *leave-one-out* zbiór uczący wykorzystywany jest w całości jako zbiór testowy. Trenowanie następuje na zbiorze uczącym pozbawionym jednego przykładu, a testowanie na wcześniej wykluczonym ze zbioru treningowego przykładzie. Procedura powtarzana jest dla każdego przykładu ze zbioru uczącego, a końcowy wynik jakości klasyfikacji jest sumą błędnych pojedynczych rozpoznań podzieloną przez moc zbioru uczącego.

[127] Jak podaje (Leung i in. 2001) – rzeczywista skuteczność takich metod, jak HCV, InduceNet czy właśnie KBANN, testowana na niezależnym, wcześniej nieobserwowanym zbiorze danych, wynosi *Accuracy* = 57% (gdzie *Accuracy* = 1 – *Error Rate*). Dla porównania, uzyskany przed model GCS wynik *Error Rate* = 24/106 oznacza osiągnięcie wartości *Accuracy* na poziomie 77%.

gdzie:

TP – liczba przykładów poprawnych, prawidłowo sklasyfikowanych (*true positives*),

FN – liczba przykładów poprawnych, nieprawidłowo sklasyfikowanych (*false negatives*).

TN – liczba przykładów niepoprawnych, prawidłowo sklasyfikowanych (*true negatives*),

FP – liczba przykładów niepoprawnych, nieprawidłowo sklasyfikowanych (*false positives*).

W kontekście realizowanego zadania, *czułość* określa prawdopodobieństwo detekcji regionu promotorowego przy założeniu, że jest obecny w analizowanej sekwencji, natomiast *swoistość* to prawdopodobieństwo, że rozpoznany zostanie region niepromotorowy podczas analizy sekwencji, w której nie ma promotora. Nowoczesne programy komputerowe rozpoznające bakteryjne regiony promotorowe osiągają miary na poziomie 80%, przy założeniu równolicznych zbiorów uczących (jako przykład może służyć uruchomiony w 2003 r. program BPROM firmy SofBerry[128]). Po przeliczeniu estymatorów modelu GCS uzyskujemy następujące wartości: $TP = 11$, $FN = 7$, $TN = 17$, $FP = 1$ i w konsekwencji $Sn = 0{,}61$[129] oraz $Sp = 0{,}94$. O ile otrzymana *czułość* jest niższa od oczekiwanej, o tyle *swoistość* modelu (a raczej indukowanej gramatyki) jest na bardzo wysokim poziomie i może być stosowana do rozpoznawania rejonów niepromotorowych. Oczekuje się, że wyższą wartość czułości model może osiągnąć po istotnym wydłużeniu okresu uczenia.

[128] http://www.softberry.com/berry.phtml?topic=bprom&group=programs&subgroup=gfindb

[129] *Czułość* równoważna jest estymatorowi *positive*.

7. Podsumowanie

Wnioskowanie gramatyczne to intensywnie rozwijana metoda uczenia maszynowego, która dzisiaj dopracowała się już własnych metod, narzędzi i celów badawczych. Maszynowe indukowanie gramatyk to jednocześnie bardzo ważne zagadnienie praktyczne, które znajduje zastosowanie począwszy od nauk kognitywistycznych, jak uczenie języka, po biologię molekularną, zwłaszcza od kiedy człowiek zaczął badać swój własny genom.

Poszukiwanie odpowiedniej gramatyki czy też ekwiwalentnego jej automatu to w istocie przeszukiwanie olbrzymiej przestrzeni potencjalnych rozwiązań. Stosowanie metod ewolucyjnych zdaje się być w takim wypadku wyjątkowo zasadne. Monografia podejmuje ciągle mało znaną w polskim piśmiennictwie tematykę wnioskowania gramatycznego, nie zatrzymując się jednak na opisie aktualnego stanu badań, ale proponując oryginalny model ewolucyjny bazujący na koncepcji uczących się systemów klasyfikujących.

Poniżej przedstawiono krótkie podsumowanie ważniejszych wyników ujętych w monografii oraz wskazano niektóre kierunki dalszych badań.

Najważniejsze wyniki szczegółowe monografii:

- Opracowano na podstawie doniesień literaturowych aktualny stan badań w zakresie indukcji gramatyki bezkontekstowej (podrozdz. 1.5).
- Zaproponowano nowy sposób kategoryzacji uczących się systemów klasyfikujących (podrozdz. 3.2).
- Opisano architekturę i cykl uczenia systemu klasyfikującego ACS, używając jednakowych pojęć zastosowanych przy opisie systemów ZCS, XCS oraz LCS (podrozdz. 3.3.4).
- Wprowadzono propozycję oryginalnego modelu ewolucyjnego GCS dedykowanego indukcji gramatyki bezkontekstowej (rozdz. 4).
- Zaproponowano tzw. mechanizm płodności produkcji, który wraz z mechanizmem ścisku oraz operatorem genetycznym inwersji ma przeciwdziałać wysokiej epistazie populacji produkcji modelu GCS (podrozdz. 4.3.2).
- Zdefiniowano nowe operatory pokrycia dostosowane do użytej w modelu GCS metody parsowania (podrozdz. 4.6 oraz 4.11.5).
- Podano algorytmiczny zapis działania modelu GCS (podrozdz. 4.11).

- Zdefiniowano estymatory dokładności i kosztu indukcji (podrozdz. 5.1).
- Przeprowadzono indukcję języków regularnych z tzw. zbioru Tomity (podrozdz. 5.3). Wyniki dokładności generalizacji i kosztu indukcji modelu GCS okazały się dla każdego z badanych języków zdecydowanie lepsze od ewolucyjnego modelu indukcji automatów DFA (Luke i in. 1999). W porównaniu z modelami o stałym rozmiarze danych (Lucas i Reynolds 2005) koszt indukcji modelu GCS okazał się w dwóch z siedmiu badanych przypadków istotnie niższy, a dla języka L5 porównywalny. Dokładność generalizacji była porównywana również z metodą nieewolucyjną EDSM – obecnie najlepszą ze znanych w literaturze metod indukcji automatów skończonych. Model GCS dla czterech języków uzyskał najlepsze wyniki generalizacji ze wszystkich porównywalnych metod, a dla języka L4 taki sam poziom generalizacji jak metody nSmart (Lucas i Reynolds 2005) oraz EDSM. Wyniki indukcji dwóch pozostałych języków plasowały model GCS na drugiej pozycji wśród porównywanych pięciu metod, z wynikami zdecydowanie wyższymi od 90%.
- Przeprowadzono indukcję wybranych języków bezkontekstowych (podrozdz. 5.4). Dla każdego z badanych języków model GCS znalazł gramatykę zgodną ze zbiorem uczącym i zdolną do generalizacji. Koszt indukcji gramatyki modelu GCS okazał się niższy od kosztu uzyskanego w (Bianchi 1996) podczas uczenia tych samych trzech języków. Model Bianchiego jest przykładem klasycznej architektury systemu klasyfikującego z uproszczoną wersją algorytmu kubełkowego. Również porównanie kosztu indukcji i dokładności generalizacji podczas indukcji czterech języków bezkontekstowych przez modele ewolucyjne Lanhorsta (1995) wypadają z korzyścią dla modelu GCS. Koszt indukcji modelu GCS, mierzony liczbą poprawnie zakończonych iteracji eksperymentów, okazał się również wyższy od ewolucyjnych metod indukujących stochastyczne gramatyki bezkontekstowe.
- Przeprowadzono indukcję dziewięciu obszernych morfosyntaktycznie oznakowanych korpusów językowych wyjętych z literatury angielskojęzycznej (podrozdz. 5.5). W wyniku wnioskowania otrzymano gramatyki, które wskazały na nietrywialne własności badanego języka naturalnego, jak przykładowo często występujące bigramy, czy też rolę rodzajnika. Uzyskane rezultaty porównano z podejściem ewolucyjnym, ale opartym o algorytm genetyczny (Aycinena i in. 2003). W przypadku pięciu korpusów model GCS wyindukował gramatykę o wyższej wartości maksymalnego dopasowania gramatyki, a dla czterech pozostałych korpusów różnice nie były większe od 5%. Jeszcze lepiej wypada porównanie kosztów indukcji – w najgorszym przypadku model GCS potrzebował niecałych 600 kroków ewolucyjnych i jednej godziny do znalezienia gramatyki o wyższym dostosowaniu, podczas gdy porównywana metoda w najlepszym wypadku potrzebowała ponad 15 000 kroków i 40 h do wyewoluowania gramatyki zgodnej z korpusem w niecałych 90%.
- Przeprowadzono szereg eksperymentów mających na celu zbadanie własności modelu GCS (podrozdz. 5.6). Wszystkie symulacje przeprowadzono na bazie poszukiwania języka bezkontekstowego TOY. Zbadano wpływ na proces indukcji metody

selekcji, ścisku, operatorów pokrycia, liczby początkowych produkcji nieterminalnych i liczby symboli nieterminalnych, wielkości elity, krzyżowania i mutacji, inwersji, płodności oraz wag funkcji dopasowania produkcji. Uzyskano szereg szczegółowych rezultatów wskazujących na różny stopień wrażliwości modelu na poszczególne parametry.

Najefektywniejszą metodą selekcji w modelu GCS jest selekcja ruletkowa, natomiast pomiędzy selekcją losową a turniejową są nieznaczne różnice. Wynika to z zastosowanego w modelu sposobu wyboru jedynie dwóch produkcji do genetycznego przetwarzania, który ogranicza istotnie różnice w zastosowanych metodach selekcji. Interesującym i zaskakującym zjawiskiem jest zmniejszanie się nacisku selektywnego wraz ze wzrostem podpopulacji turniejowej. Ta pozornie sprzeczna z doniesieniami literaturowymi sytuacja jest efektem współdziałającego z selekcją mechanizmu ścisku. Istotną rolę ścisku, a dokładniej wartości parametru podpopulacji ścisku w modelu GCS, potwierdziła seria kolejnych doświadczeń, podczas których badano zależność kosztu i jakości indukcji od parametrów zatłoczenia.

Zbadano również rolę operatorów pokrycia. Stwierdzono, że największy wpływ ma operator pokrycia pełnego, który jest w stanie samodzielnie zapewnić uczeniu dobre tempo zbieżności. Operator pokrycia uniwersalnego w porównaniu z operatorem pełnego pokrycia potrzebuje około dziesięciokrotnie dłuższego okresu uczenia, aby w każdej iteracji eksperymentu indukować prawidłową gramatykę. Połączenie operatora pełnego i uniwersalnego pozwala na najefektywniejsze uczenie, a brak obydwu operatorów praktycznie ten proces uniemożliwia. Operatory pokrycia startowego oraz agresywnego mają znikomy wpływ na indukcję gramatyki.

Proces indukcji zależny jest od liczby początkowych produkcji nieterminalnych, której minimalna liczba w sposób oczywisty zależy od rozmiarów indukowanej gramatyki. Wpływ liczby produkcji na mechanizm uczenia jest zasadniczo różny dla indukcji z włączonym i wyłączonym operatorem pokrycia uniwersalnego. Proces uczenia przestaje zależeć od początkowej liczby produkcji zdecydowanie szybciej dla uczenia stosującego operator uniwersalny.

Rozmiar populacji elitarnej w znacznym przedziale wartości praktycznie nie wpływa na koszt i jakość indukcji, choć zgodnie z przewidywaniami dla wysokich wartości skutecznie blokuje proces uczenia.

Przestrzenne i przekrojowe wykresy zależności estymatorów indukcji od parametrów mutacji i krzyżowania wykazują jednoznacznie, że mutacja w modelu GCS, podobnie jak w innych modelach ewolucyjnych o niedużych populacjach – a do takich należy model GCS – ma dominującą rolę.

Operator inwersji, którego zadaniem miało być zapobieganie negatywnym skutkom własności epistatycznych populacji produkcji, nie ma większego wpływu ma uczenie. Również płodność, chroniąca produkcje stosowane wcześniej w ciągu wyprowadzeń, ma niespodziewany wpływ na proces indukcji. Zwiększanie roli „płodnych" produkcji w gramatyce powoduje… tłumienie tempa indukcji. Spowodowane

jest to stagnacją populacji wynikającą z kolei z mechanizmu ścisku, który dobierając do wymiany produkcje słabe, rzadziej bierze pod uwagę dodatkowe nagradzane przez model produkcje „płodne".

Interesującą własność modelu zaobserwowano podczas badania zależności procesu uczenia od używanej liczby symboli nieterminalnych. Oczywista była utrata zdolności uczenia modelu poniżej pewnej liczby symboli, gdyż wynika to bezpośrednio z minimalnej liczby opisującej indukowaną gramatykę. Dosyć niespodziewany efekt natomiast to wzrost kosztu uczenia połączony ze zmniejszaniem dokładności uczenia dla wyższych wartości analizowanego parametru. Większa liczba symboli nieterminalnych powoduje wzrost liczby możliwych drzew rozbiorów, a co za tym idzie wzmocnienie eksploracji rozwiązań kosztem eksplotacji. Dodany do uczenia operator pokrycia uniwersalnego wyraźnie proces ten hamuje.

Podczas eksperymentów zbadano również wpływ współczynników funkcji dopasowania pojedynczej produkcji na tempo i dokładność indukcji gramatyki.

- Zweryfikowano użyteczność modelu GCS – poza udanym zastosowaniem modelu w inżynierii lingwistycznej (podrozdz. 5.5) – podczas indukcji gramatyki opisującej sekwencje telomerowe u człowieka oraz regiony promotorowe u bakterii *E. coli* (rozdz. 6). Model GCS bezbłędnie nauczył się gramatyki telomerowej, opisywanej wyrażeniem regularnym, średnio w niewiele ponad 500 krokach ewolucyjnych. Rozpoznawanie regionów promotorowych w genomie prokariotycznej bakterii okazało się zdecydowanie trudniejszym zadaniem. Właściwości wyindukowanej gramatyki uprawniają do wyciągnięcia wniosku, że model w obecnej implementacji może być zastosowany na wysokim poziomie *swoistości* do rozpoznawania regionów nienależących do sekwencji promotorowych.

Monografia nie wyczerpuje oczywiście problematyki ewolucyjnych metod wnioskowania gramatycznego, w tym zaproponowanego nowego modelu ewolucyjnego. Wśród kierunków dalszych badań można wymienić następujące:

- Przystosowanie modelu GCS do indukcji stochastycznej gramatyki bezkontekstowej. Zmiany wymagałaby m.in. reprezentacja pojedynczej produkcji, która musiałaby zostać uzupełniona o prawdopodobieństwo produkcji, metoda parsowania i wyliczania dopasowania gramatyki, zbiór uczący, w którym konieczne byłoby pojawienie się obok każdego przykładu parametru określającego jego częstość występowania. Niewątpliwą korzyścią, i to szczególnie istotną w praktycznych zastosowaniach modelu, byłaby możliwość rezygnacji podczas uczenia z przykładów negatywnych.
- Implementacja modelu GCS w postaci zbioru koewolucyjnych systemów. Wstępne publikowane wyniki doświadczeń z koewolucji prostych uczących się systemów klasyfikacyjnych są bardzo zachęcające (Bull i in. 2005).
- Dalsze badania nad zastosowaniem modelu GCS w genomice obliczeniowej. Gramatyką bezkontekstową opisuje się wiele biosekwencji będących w centrum zainteresowania biologów molekularnych, jak chociażby regiony promotorowe organizmów eukariotycznych.

- Dalsze badania nad uczącymi się systemami klasyfikującymi, ze szczególnym uwzględnieniem modelu ACS. Analiza literatury wskazuje, że jest to ciągle jeden z mniej eksploatowanych modeli, a jednocześnie zaimplementowany w nim niezwykle interesujący mechanizm uczenia antycypacyjnego daje nadzieje na oryginalne zastosowania.

Wydaje się potrzebnym, analizując stan wiedzy i piśmiennictwa, opracowanie w przyszłości podręcznika dotyczącego uczących się systemów klasyfikujących.

Załącznik A

Zbiory uczące dla języków Tomity w formacie *abbadingo*.

L1

```
16 2
1 1 a
1 2 a a
1 3 a a a
1 4 a a a a
1 5 a a a a a
1 6 a a a a a a
1 7 a a a a a a a
1 8 a a a a a a a a
0 1 b
0 2 a b
0 2 b a
0 2 b b
0 3 b a a
0 3 a a b
0 8 a a a a a a a b
0 8 a b a a a a a a
```

L2

```
15 2
1 2 a b
1 4 a b a b
1 6 a b a b a b
1 8 a b a b a b a b
1 14 a b a b a b a b a b a b a b
0 1 a
0 1 b
0 2 a a
0 2 b b
0 2 b a
0 3 a b a
0 3 a b b
0 5 a b a a b
0 7 a b b a b a b
0 9 a a b a b a b a b
```

L3

```
24 2
1 1 a
1 1 b
1 2 b a
1 2 a a
1 2 b b
1 3 a b b
1 3 a a b
1 3 a a a
1 3 b b b
1 6 a b b a b b
1 15 a a b b b b b a a a b b b b a
1 18 a a a a b a a b b b a b b a a a a b b
0 2 a b
0 3 a b a
0 3 b a b
0 4 a b a b
0 4 a b a a
0 4 a a a b
0 5 a b b b a
0 6 a a a b a b
0 7 a b b a b b b
0 8 a a a a a b b b
0 11 b a a a b b a a a b a
0 12 a a b a a a b b b a a b
```

L4

```
19 2
1 1 a
1 1 b
1 2 a b
1 2 b a
1 2 b b
1 3 b a b
1 5 a a a b b
1 6 a b b a b b
1 10 b a b b a b b a b b
1 12 b b a a a a a a a b a b b
0 3 b b b
0 4 b b b a
0 4 b b b b
0 5 b b b b b
0 5 a a b b b
0 9 b b b b b b b b b
0 10 a b a b b a b b b a
0 11 a a a a a b b b b a a
0 16 a a b a b a b b b b b a b a a a
```

L5

```
21 2
1 2 a a
1 2 b b
1 4 a b b a
1 4 b a b a
1 4 a b a b
1 4 b b b b
1 6 a a a a a a
1 10 a b b b a a a a b a
1 16 a b b a a b b b b a a a a b a b
0 1 a
0 1 b
0 2 a b
0 2 b a
0 3 b a a
0 3 a a a
0 3 b a b
0 4 a b b b
0 4 b b b a
0 9 b b b b b b b b b
0 10 a a a b b a b a b b
0 12 b a b a a a a a a a a b
```

L6

```
21 2
1 2 a b
1 2 b a
1 3 a a a
1 4 a a b b
1 5 a b a a a
1 6 a b a b a b
1 6 b b b b b b
1 9 a b b a b b a b b
1 10 b a a a a b a a a a
0 1 a
0 1 b
0 2 a a
0 2 b b
0 3 a b a
0 3 b a a
0 4 a a a a
0 5 a a b b a
0 6 b a b a a a
0 8 b b b b b b b b
0 10 a b b a b b a b b a
0 11 a b a a a a b a a a a
```

L7

```
20 2
1 1 a
1 1 b
1 2 a b
1 2 b a
1 2 b b
1 3 b b b
1 4 b a b a
1 5 a a a a a
1 5 b b a b b
1 8 b b a a b b a a
1 12 b a a a a a b a a a a a
1 13 b b b b a b b b b a a a a
0 4 a b a b
0 5 a b a b a
0 6 b a b a b b
0 6 a b a b b a
0 7 a b a a b a b
0 10 b a b a b a b a b a
0 11 b b a a b b a a b b b
0 12 a b b a b b a a b a b a
```

Załącznik B

Załącznik przedstawia kolejne etapy transformacji przykładowego fragmentu korpusu językowego (*Alicja w krainie czarów*) do formatu *abbadingo*.

Etap 0 – źródłowy korpus językowy

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice "without pictures or conversation?"
So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.
There was nothing so very remarkable in that;
nor did Alice think it so very much out of the way to hear the Rabbit say to itself, "Oh dear!
Oh dear!
I shall be late!" (when she thought it over afterwards, it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural);
but when the Rabbit actually took a watch out of its waistcoat-pocket, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.
In another moment down went Alice after it, never once considering how in the world she was to get out again.
The rabbit-hole went straight on like a tunnel for some way, and then dipped suddenly down, so suddenly that Alice had not a moment to think about stopping herself before she found herself falling down a very deep well.
Either the well was very deep, or she fell very slowly, for she had plenty of time as she went down to look about her and to wonder what was going to happen next.
…

Etap 1 – korpus językowy oznakowany *taggerem* Brilla

Alice/NNP was/VBD beginning/VBG to/TO get/VB very/RB tired/VBN of/IN sitting/VBG by/IN her/PRP$ sister/NN on/IN the/DT bank,/NN and/CC of/IN having/VBG nothing/NN to/TO do:/VB once/RB or/CC twice/RB she/PRP had/VBD peeped/VBN into/IN the/DT book/VB her/PRP$ sister/NN was/VBD reading,/VBG but/CC it/PRP had/VBD no/DT pictures/NNS or/CC conversations/NNS in/IN it,/NN "and/NN what/WP is/VBZ the/DT use/NN of/IN a/DT book,"/NN thought/VBD Alice/NNP "without/NN pictures/NNS or/CC conversation?"/NN
So/RB she/PRP was/VBD considering/VBG in/IN her/PRP$ own/JJ mind/NN (as/NNS well/RB as/IN she/PRP could,/VBP for/IN the/DT hot/JJ day/NN made/VBD her/PRP$ feel/NN very/RB sleepy/JJ and/CC stupid),/NN whether/IN the/DT pleasure/NN of/IN making/VBG a/DT daisy-chain/NN would/MD be/VB worth/JJ the/DT trouble/NN of/IN getting/VBG up/IN and/CC picking/VBG the/DT daisies,/NN when/WRB suddenly/RB a/DT White/NNP Rabbit/NNP with/IN pink/JJ eyes/NNS ran/VBD close/VB by/IN her./CD
There/EX was/VBD nothing/NN so/RB very/RB remarkable/JJ in/IN that;/NN
nor/CC did/VBD Alice/NNP think/VBP it/PRP so/RB very/RB much/RB out/IN of/IN the/DT way/NN to/TO hear/VB the/DT Rabbit/NNP say/VBP to/TO itself,/VB "Oh/NN dear!/NN
Oh/UH dear!/NN
I/PRP shall/MD be/VB late!"/VBN (when/VBN she/PRP thought/VBD it/PRP over/IN afterwards,/NN it/PRP occurred/VBD to/TO her/PRP that/IN she/PRP ought/MD to/TO have/VB wondered/VBN at/IN this,/NN but/CC at/IN the/DT time/NN it/PRP all/DT seemed/VBD quite/RB natural);/VBP
but/CC when/WRB the/DT Rabbit/NNP actually/RB took/VBD a/DT watch/NN out/IN of/IN its/PRP$ waistcoat-pocket,/JJ and/CC looked/VBD at/IN it,/NN and/CC then/RB hurried/VBD on,/NNP Alice/NNP started/VBD to/TO her/PRP$ feet,/NN for/IN it/PRP flashed/VBD across/IN her/PRP$ mind/NN that/IN she/PRP had/VBD never/RB before/IN seen/VBN a/DT rabbit/NN with/IN either/DT a/DT waistcoat-pocket,/JJ or/CC a/DT watch/NN to/TO take/VB out/IN of/IN it,/NN and/CC burning/VBG with/IN curiosity,/NN she/PRP ran/VBD across/IN the/DT field/NN after/IN it,/NN and/CC fortunately/RB was/VBD just/RB in/IN time/NN to/TO see/VB it/PRP pop/VB down/RP a/DT large/JJ rabbit-hole/NN under/IN the/DT hedge./JJ
In/IN another/DT moment/NN down/RB went/VBD Alice/NNP after/IN it,/NN never/RB once/RB considering/VBG how/WRB in/IN the/DT world/NN she/PRP was/VBD to/TO get/VB out/IN again./CD
The/DT rabbit-hole/NN went/VBD straight/JJ on/IN like/IN a/DT tunnel/NN for/IN some/DT way,/NN and/CC then/RB dipped/VBD suddenly/RB down,/VBP so/RB suddenly/RB that/IN Alice/NNP had/VBD not/RB a/DT moment/NN to/TO think/VB about/IN stopping/VBG herself/PRP before/IN she/PRP found/VBD herself/PRP falling/VBG down/RP a/DT very/RB deep/JJ well./CD
Either/CC the/DT well/RB was/VBD very/JJ deep,/NN or/CC she/PRP fell/VBD very/JJ slowly,/NN for/IN she/PRP had/VBD plenty/NN of/IN time/NN as/IN she/PRP went/VBD down/RB to/TO look/VB about/IN her/PRP$ and/CC to/TO wonder/VB what/WP was/VBD going/VBG to/TO happen/VB next./JJ …

Etap 2 – korpus z usuniętymi słowami języka naturalnego

NNP VBD VBG TO VB RB VBN IN VBG IN PRP$ NN IN DT NN CC IN VBG NN TO VB RB CC RB PRP VBD VBN IN DT VB PRP$ NN VBD VBG CC PRP VBD DT NNS CC NNS IN NN NN WP VBZ DT NN IN DT NN VBD NNP NN NNS CC NN
RB PRP VBD VBG IN PRP$ JJ NN NNS RB IN PRP VBP IN DT JJ NN VBD PRP$ NN RB JJ CC NN IN DT NN IN VBG DT NN MD VB JJ DT NN IN VBG IN CC VBG DT NN WRB RB DT NNP NNP IN JJ NNS VBD VB IN CD
EX VBD NN RB RB JJ IN NN
CC VBD NNP VBP PRP RB RB RB IN IN DT NN TO VB DT NNP VBP TO VB NN NN
UH NN
PRP MD VB VBN VBN PRP VBD PRP IN NN PRP VBD TO PRP IN PRP MD TO VB VBN IN NN CC IN DT NN PRP DT VBD RB VBP
CC WRB DT NNP RB VBD DT NN IN IN PRP$ JJ CC VBD IN NN CC RB VBD NNP NNP VBD TO PRP$ NN IN PRP VBD IN PRP$ NN IN PRP VBD RB IN VBN DT NN IN DT DT JJ CC DT NN TO VB IN IN NN CC VBG IN NN PRP VBD IN DT NN IN NN CC RB VBD RB IN NN TO VB PRP VB RP DT JJ NN IN DT JJ
IN DT NN RB VBD NNP IN NN RB RB VBG WRB IN DT NN PRP VBD TO VB IN CD
DT NN VBD JJ IN IN DT NN IN DT NN CC RB VBD RB VBP RB RB IN NNP VBD RB DT NN TO VB IN VBG PRP IN PRP VBD PRP VBG RP DT RB JJ CD
CC DT RB VBD JJ NN CC PRP VBD JJ NN IN PRP VBD NN IN NN IN PRP VBD RB TO VB IN PRP$ CC TO VB WP VBD VBG TO VB JJ
…

Etap 3 – korpus ze zredukowanym zestawem tagów

N V V P V R V P V P J N P T N T P V N P V R T R N V V P T V J N V V T N V T N T N P N N N V T N P T N V N N N T N
R N V V P J J N N R P N V P T J N V J N R J T N P T N P V T N V V J T N P V P T V T N R R T N N P J N V V P J
T V N R R J P N
T V N V N R R R P P T N P V T N V P V N N
O N
N V V V V N V N P N N V P N P N V P V V P N T P T N N T V R V
T R T N R V T N P P J J T V P N T R V N N V P J N P N V P J N P N V R P V T N P T T J T T N P V P P N T V P N N V P T N P N T R V R P N P V N V P T J N P T J
P T N R V N P N R R V R P T N N V P V P J
T N V J P P T N P T N T R V R V R R P N V R T N P V P V N P N V N V P T R J J
T T R V J N T N V J N P N V N P N P N V R P V P J T P V N V V P V J
…

Etap 4 – korpus w formacie *abbadingo* uzupełniony o losowo wygenerowane niepoprawne ciągi tagów

```
2024 7
1 55 d a b b e c c a a d e a b e f c a b c a d c f a e f a e b f a b b c f a e b e f b f a d d f a a e c
a b b e c
0 10 b c d g b e b d a a
1 8 f b a d d c e a
0 9 b c g d f e f f f
1 2 g a
0 6 e c c g d c
1 31 a b b b b a b a e a a b e a e a b e b b e a f e f a a f b d b
0 11 f g g g e f e c c b f
1 21 e f a d b a e a d d b d e f a a b e b e c
0 9 d g a b g f c f f
1 39 f a b c e e f a e f a f d b d b d d e a b d f a e b e b a e a b a b e f d c c
0 15 a c f b e a g c a d g e a d b
1 22 a a b e b d f b e a a b b a f a b d c e b a
0 10 b d c g c c c a b d
1 19 d a b e f a e f a f b e a b b e a f c
0 12 a c f b c b c c g a e c
1 13 a b e f a e c e f a e a b
0 12 f g b f g f d e f b b a
1 41 a b b a a f e c c a a b b a b d b e b f a e a e b a d b e b a e c e f a e a b c c
0 14 f b a c f b d f b g b d f g
1 8 d c b f a a e a
0 14 g c d a d b g e e b d f a d
1 25 a a b b a e a d e a b e f a e f a a b d c a a a e
0 6 b g e g d f
1 13 a a d c a a b e f a a b c
0 13 c d d a e c b f e d e a g
1 11 a b b b d e f a e f c
0 11 f e e f b c b e c d c
1 10 a a e a b b a e f a
0 8 d d d e c a b a
1 16 d c a b e b e e f a f b e c a b
0 15 a c e c d c d d b e b d c g f
1 23 a a b f a a f a a a b e b e a c a e a b e f a
0 14 g e d b e e b a f b c c f a
1 18 b a b a b b a a a f c c a a b a e a
0 7 b d c d d c d
…
```

Załącznik C

Załącznik zawiera reguły przekształcenia wyrażenia regularnego *WR* na zbiór produkcji gramatyki bezkontekstowej w postaci normalnej Chomsky'ego.

Reguła 1
Jeżeli WR: *a*, to stwórz produkcję typu
$A \rightarrow a$.

Reguła 2
Jeżeli WR: *WR1 WR2*, to stwórz produkcje typu
$A \rightarrow <WR1>< WR2>$
oraz stwórz produkcje dla *WR1* i *WR2*.

Reguła 3
Jeżeli WR: *WR1* | *WR2*, to stwórz produkcje typu
$A \rightarrow <WR1>$
$A \rightarrow <WR2>$
oraz stwórz produkcje dla *WR1* i *WR2*.

Reguła 4
Jeżeli WR: $WR1^+$, to stwórz produkcje typu
$A \rightarrow <WR1>\ A$
$A \rightarrow <WR1>$
oraz stwórz produkcje dla *WR1*.

Reguła 5
Jeżeli WR: $WR1^*\ WR2$, to stwórz produkcje typu
$A \rightarrow <WR1>\ A$
$A \rightarrow <WR1>$
$B \rightarrow <WR2>$
$C \rightarrow AB$
$C \rightarrow <WR2>$
oraz stwórz produkcje dla *WR1* i *WR2*.

Bibliografia

ABE N., MAMITSUKA H. (1997), *Predicting protein secondary structure using stochastic tree grammars*, Machine Learning, 29, 275–301.

ADRIAANS P.W. (1999), *Learning shallow context-free languages under simple distributions*, ILLC Report PP-1999-13, Institute for Logic, Language and Computation, Amsterdam.

AHLUWALIA M., BULL L. (1999), *A Genetic Programming-based Classifier System*, w: Proc. Genetic and Evolutionary Computation Conf. GECCO-99, red. W. Banzhaf i in., Morgan Kaufmann, San Francisco, CA, 11–18.

AHO A.V., SETHI R., ULLMAN J.D. (1986), *Compilers: Principles, Techniques and Tools*, Addison-Wesley, Reading, MA.

AHONEN H., MANNILA H., NIKUNEN E. (1994), *Forming grammars for structured documents: An application of grammatical inference*, w: Grammatical Inference and Application ICGI-94, red. R. Carrasco, J. Oncina, Springer, Berlin, Heidelberg, 153–167.

AMENGUAL J.C., BENEDI J.M., CASACUBERTA F., CASTANO A., CASTELLANOS A., JIMENEZ V.M., LLORENS D., MARZAL A., PASTOR M., PRAT F., VIDAL E., VILAR J.M. (2001), *The EUTRANS-I speech translation system*, Machine Translation, 15, 75–103.

ANGELINE P. (1994), *Evolutionary Algorithms and Emergent Intelligence*, PhD Thesis, Computer Science Department, Ohio State University.

ANGELINE P. (1997), *An alternative to indexed memory for evolving programs with explicit state representations*, w: Proc. 2nd Conf. Genetic Programming (GP97), red. J.R. Koza i in., Morgan Kaufmann, San Francisco, CA, 423–430.

ANGLUIN D. (1981), *A note on the number of queries needed to identify regular languages*, Information and Control, 51, 76–87.

ANGLUIN D. (1982), *Inference of reversible languages*, JACM 2(3), 741–765.

ANGLUIN D. (1987a), *Learning regular sets from queries and counterexamples*, Information and Computation, 75, 87–106.

ANGLUIN D. (1987b), *Learning k-bounded context-free grammars*, Yale Technical Report, RR–557.

ANGLUIN D. (1988), *Queries and concept learning*, Machine Learning, 2(4), 319–342.

ANGLUIN D. (2001), *Queries revisited*, w: Proc. ALT 2001, LNCS 2225, red. N. Abe i in., Springer, Berlin, Heidelberg, 12–31.

ANGLUIN D. SMITH C.H., (1983), *Inductive Inference: Theory and Models*, Computing Surveys, 15, 3, 237–269.

ARABAS J. (2001), *Wykłady z algorytmów ewolucyjnych*, WNT, Warszawa.

ARIKAWA S., SHINOHARA T., YAMAMOTO A. (1992), *Learning elementary formal system*, Theoretical Computer Science, 2(1), 91–113.

ARIMURA H, SAKAMOTO H., ARIKAWA S. (2001), *Efficient learning of semi-structured data from queries*, Proc. ALT 2001, LNCS 2225, Springer, Berlin, Heidelberbg, 315–331.

ARTHUR W.B., HOLLAND J., LEBARON B., PALMER R., TAYLER P. (1996), *Asset Pricing Under Endogenous Expectations in an Artificial Stock Market*, Technical Report, Santa Fe Institute.

ASSELMEYER T., EBELING W. (1997), *Unified description of evolutionary strategies over continuous parameter spaces*, BioSystems, 41(3), 167–178.

ATWELL E., ARNEL S., DEMETRIOU G., HANLON S., HUGHES J., JOST U., POCOCK R., SOUTER C., UEBERLA L. (1993), *Multi-level Disambiguation Grammar Inferred from English Corpus, Treebank and Dictionary*, w: Grammatical Inference – Theory, Applications And Alternatives, red. S. Lucas, Colloquium Proceedings, No. 1993/092, London, UK, 91–98.

AYCINENA M., KOCHENDERFER M.J., MULFORD D.C. (2003), *An evolutionary approach to natural language grammar induction*, Final Project for CS224N: Natural Language Processing, Stanford University.

BAKER J.K. (1979), *Trainable grammars for speech recognition*, w: Speech Communication Papers for the 97th Meeting of the Acoustical Society of America, red. D.H. Klatt, J.J. Wolf, MIT, Cambridge, Mass, 547–550.

BALCAZAR J.L., DIAZ J., GAVALDÁ R., WATANABE O. (1994), *The query complexity of learning DFA*, New Generat. Comput., 12, 337–358.

BANZHAF W., NORDIN P., KELLER R.E., FRANCONE F.D. (1998), *Genetic programming. An Introduction. On the Automatic Evolution of Computer Programs and its Applications*, Morgan Kaufmann, San Francisco, CA.

BARRY A. (2000), *XCS Performance and Population Structure within Multiple-Step Environments*, PhD Thesis, Queens University, Belfast.

BERMAN P., ROOS R. (1987), *Learning one-counter languages in polynomial time*, Proc. 28th FOCS, 61–67.

BERNARD M., HABRARD A. (2001), *Learning stochastic logic programs*, Proc. Int. Conf. on Inductive Logic Programming, 19–26.

BERTOLO S. (2001), *A Brief Overview of Learnability*, w: Language Acquisition and Learnability, red. S. Bertolo, Cambridge University Press, 1–14.

BETHKE A.D. (1980), *Genetic algorithms as function optimizers*, PhD Thesis, University of Michigan.

BEYER H.G. (1995), *Toward a theory of evolution strategies: On the benefits of sex – the* $(\mu/\mu, \lambda)$ *theory*, Evolutionary Computation, 3(1), 81–111.

BIANCHI D. (1996), *Learning Grammatical Rules from Examples Using a Credit Assignment Algorithm*, Proc. 1st Online Workshop on Soft Computing (WSC1), Nagoya.

BODÉN M., WILES J. (2000), *Context-free and context-sensitive dynamics in recurrent neural networks*, Connection Science, 12(3), 197–210.

BODÉN M., WILES J. (2000), *On learning context-free and context-sensitive languages*, IEEE Transactions on Neural Networks, 13(2), 491–493.

BONARINI A. (2000), *An Introduction to Learning Fuzzy Classifier Systems*, w: Learning Classifier Systems: From Foundations to Applications, red. P.L. Lanzi i in., LNCS 1813, Springer, Berlin, Heidelberg, 83–106.

BONELLI P., PARODI A. (1991), *An Efficient Classifier System and its Experimental Comparison with two other Representative Learning Methods on Three Medical Domains*, w: Proc. 4th Int. Conf. Genetic Algorithms, ICGA91, red. L.B. Booker, R.K. Belew, Morgan Kaufmann, San Francisco, CA, 288–295.

BONELLI P., PARODI A., SEN S., WILSON S. (1990), *NEWBOOLE: A Fast GMBL System*, Int. Conf. on Machine Learning, San Mateo, CA, Morgan Kaufmann, 153–159.

BONGARD J., LIPSON H. (2005), *Active Coevolutionary Learning of Deterministic Finite Automata*, J. of Machine Learning Research, 6, 1651–1678.

BONNEMA R., BOD R., SCHA R. (1997), *A DOP model for semantic interpretation*, w: Proc. Association for Computational Linguistics/European Chapter of the Association for Computational Linguistics, Madrid, Sommerset, NJ: Association for Computational Linguistics, 159–167.

BOOKER L.B. (1982), *Intelligent Behavior as an Adaptation to the Task Environment*, PhD Dissertation, University of Michigan.

BOOKER L.B. (1989), *Triggered rule discovery in classifier systems*, w: Proc. 3rd Int. Conf. Genetic Algorithms, ICGA-89, red. J.D. Schaffer, Morgan Kaufmann, George Mason University, San Francisco, CA, 265–274.

BOOKER L.B., GOLDBERG D., HOLLAND J. (1989), *Classifier Systems and Genetic Algorithms*, Artificial Intelligence, 40, 235–282.

BOOTH T.L., THOMPSON R.A. (1973), *Applying probability measures to abstract languages*, IEEE Trans. Comput., C-22(5), 442–450.

BOSTRÖM H. (1996), *Theory-Guided Induction of Logic Programs by Inference of Regular Languages*, Proc. 13th Int. Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA, USA.

BOSTRÖM H. (1998), *Predicate Invention and Learning from Positive Examples Only*, Proc. 10th European Conf. Machine Learning, LNAI 1398, Springer, Berlin, Heidelberg, 226–237.

BRILL E. (1993), *A Corpus-Based Approach to Language Learning*, PhD Dissertation, Department of Computer and Information Science, University of Pennsylvania.

BRILL E. (1993), *Automatic grammar induction and parsing free text: A transformation-based approach*, Proc. ACL, Gimbus, OH, 31, 259–265.

BRISCOE E.J. (2000), *Grammatical Acquisition: Inductive Bias and Coevolution of Language and the Language Acquisition Device*, Language, 76(2), 245–296.

BULL L. (1999), *On using ZCS in a Simulated Continuous Double-Auction Market*, w: Proc. Genetic and Evolutionary Computation Conference GECCO-99, red. W. Banzhaf i in., Morgan Kaufmann, San Francisco, CA, 83–90.

BULL L., O'HARA T.O. (2002), *Accuracy-based Neuro and Neuro-Fuzzy Classifier Systems*, w: Proc. Evolutionary Computation Conference 2002, red. W. Langdon i in., Morgan Kaufmann, New York, USA, 905–911.

BULL L., STUDLEY M., BAGNALL T., WHITTLEY I. (2005), *On the use of Rule-sharing in Learning Classifier System Ensembles*, The 2005 IEEE Congress, Vol. 1, 612–617.

BURAGO A. (1994), *Learning structurally reversible context-free grammars from queries and counterexamples in polynomial time*, Proc. 7th COLT, 140–146.

BURNARD L. (1995), *Users Reference Guide for the British National Corpus*.

BUTZ M., GOLDBERG D.E., STOLZMANN W. (1999), *New Challenges for an Anticipatory Classifier System: Hard Problems and Possible Solutions*, Technical Report 99019, the Illinois Genetic Algorithms Laboratory.

BUTZ M., GOLDBERG D.E., STOLZMANN W. (2002), *The Anticipatory Classifier System and Genetic Generalization*, Natural Computing, 1(4), 427–467.

BUTZ M., WILSON S.W. (2000), *An Algorithmic Description of XCS*, IlliGAL Report No. 2000017, University of Illinois at Urbana-Champaign.

CAO Y.J., IRESON N., BULL L., MILES R. (1999), *Design of a Traffic Junction Controller using a Classifier System and Fuzzy Logic*, Proc. 6th Int. Conf. Computational Intelligence, Theory and Applications, Springer, London, UK.

CARMEL D., MARKOWITCH S. (1998), *Model-based learning of interaction strategies in multi-agent systems*, J. of Experimental and Theoretical Artificial Intelligence, 10(3), 309–332.

CARMEL D., MARKOWITCH S. (1999), *Exploration strategies for model-based learning in multiagent systems*, Autonomous Agents and Multi-agent Systems, 2(2), 141–172.

CARSE B. (1994), *Learning Anticipatory Behaviour using a Delayed Action Classifier System*, w: Evolutionary Computing, AISB Workshop Selected Papers, red. T.C. Fogarty, LNCS 865, Springer, 210–223, London, UK.

CARSE B., FOGARTY T.C. (1994), *A New Approach to Genetics Based Machine Learning in Fuzzy Controller Design*, Int. Symp. on Intelligent Control, IEEE Piscataway, 231–236.

CASACUBERTA F. (1995), *Probabilistic estimation of stochastic regular syntax-directed translation schemes*, w: 6th Spanish Symposium on Pattern Recognition and Image Analysis, red. R. Moreno, 201–297.

CAVICCHIO D.J. (1970), *Adaptive search using simulated evolution*, PhD Thesis, University of Michigan, Ann Arbor.

CHARNIAK E. (1993), *Statistical Language Learning*, MIT Press, Cambridge, MA.

CHARNIAK E. (2000), *A Maximum-Entropy-Inspired Parser*, Proc. NAACL-2000, 132–139.

CHEN S., LESNIK E.A., HALL T.A., SAMPATH R., GRIFFEY R.H., ECKER D.J., BLYN L.B. (2002), *A bioinformatics based approach to discover small RNA genes in the Escherichia coli genome*, BioSystems, 65, 157–177.

CHEN S.F. (1995), *Bayesian grammar induction for language modelling*, Proc. 33rd Annual Meeting of Association for Computational Linguistics, 228–235.

CHIDLOVSKII B. (2002), *Schema extraction from xml collections*, Proc. 2nd ACM/IEEE-CS Joint Conference on Digital Libraries, ACM Press, 291–292.

CHOMSKY N. (1956), *Three models for the description of language*, IRE Trans. Information Theory, Vol. 2, 3, 113–124.

CHOMSKY N. (1959), *On certain formal properties of grammars*, Information and Control, Vol. 2, 2, 137–167.

CHOMSKY N. (1972), *Language and Mind*, Harcourt Brace Jovanovich, New York.

CHROBAK M., UNOLD O. (2000a), *Poszukiwanie gramatyki języka naturalnego jako zadanie optymalizacyjne*, w: IV Krajowa konferencja naukowa nt. Sztuczna inteligencja, SzI – 15 '2000 (badania – zastosowania – rozwój), Wydawnictwo Akademii Podlaskiej, Siedlce–Warszawa, 85–89.

CHROBAK M., UNOLD O. (2000b), *Adaptacyjne poszukiwanie gramatyki języka naturalnego*, w: Inżynieria wiedzy i systemy ekspertowe, T.1., red. Z. Bubnicki i A. Grzech, Oficyna Wydawnicza PWr., Wrocław, 212–219.

CHROBAK M., UNOLD O. (2001), *Natural language grammar inference by genetic search*, Proc. 3rd Int. Conf. on Intelligent Processing and Manufacturing of Materials IPMM – 2001, University of British Columbia, Vancouver, Canada.

CICHOSZ P. (2000), *Systemy uczące się*, WNT, Warszawa.

CIELECKI Ł. (2004), *Zastosowanie systemów XCS w ewolucyjnej indukcji gramatyk bezkontekstowych*, praca magisterska, Politechnika Wrocławska, Wydział Elektroniki.

CLARK A. (2001a), *Unsupervised Language Acquisition: Theory and Practice*, PhD Thesis, University of Sussex.

CLARK A. (2001b), *Unsupervised induction of stochastic context-free grammars using distributional clustering*, w: Proc. CoNLL-2001, red. W. Daelemans, R. Zając, Toulouse, France, 105–122.

CLARK A. (2004), *Grammatical Inference and First Language Acquisition*, Workshop on Psychocomputational Models of Human Language Acquisition, Geneva, Switzerland.

CLIFF D., ROSS S. (1994), *Adding memory to ZCS*, Adaptive Behavior, 3(2), 101–150.

COLLINS M., (1999), *Head-Driven Statistical Models for Natural Language Parsing*, PhD Thesis, University of Pennsylvania.

COLLINS M., DUFFY N. (2002), *New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron*, Proc. ACL 2002, Philadelphia, PA.

COLOMBETTI M., DORIGO M. (1994), *Training agents to perform sequential behavior*, Adaptive Behavior, 2(3), 247–275.

COLOMBETTI M., DORIGO M. (1999), *Evolutionary Computation in Behavior Engineering*, w: Evolutionary Computation: Theory and Applications, Chapter 2, 37–80.

COLOMBETTI M., DORIGO M., BORGHI G. (1996), *Behavior Analysis and Training: A Methodology for Behavior Engineering*, IEEE Transactions on Systems, Man and Cybernetics 26(6), 365–380.

Corcoran A.L., Sen S. (1994), *Using real-valued genetic algorithms to evolve rule sets for classification*, Proc. IEEE-CEC, Orlando, USA, 120–124.

CRESPI-REGHIZZI S. (1971), *An effective model for grammar inference*, Information Processing, 71, 524–529.

CRESPI-REGHIZZI S. (1974), *Non-counting languages and learning*, Proc. NATO Advanced Study Institute on Computer Learning Processes, 171–190.

CRUZ P., VIDAL E. (1998), *Learning regular grammars to model music style: Comparing different coding schemes*, Proc. Grammatical Inference ICGI'98, LNAI 1433, Springer, 211–222.

CYRE W.R. (2002), *Learning Grammars with a Modified Classifier System*, Proc. 2002 World Congress on Computational Intelligence, Honolulu, Hawaii, 1366–1371.

DAS S., GILES C., SUN G.Z. (1992), *Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory*, Proc. 14th Annual Conf. Cognitive Science Society, 791–795.

DAS S., GILES C., SUN G.Z. (1993), *Using prior knowledge in a NNPDA to learn context-free languages*, w: Advances in Neural Information Processing Systems, red. C. Giles i in., 5, 65–72.

DAVIS M.S. (2000), *A Computational Model of Affect Theory: Simulations of Reducer/Augmenter and Learned Helplessness Phenomena*, PhD Thesis, Department of Psychology, University of Michigan.

DE JONG K. (1975), *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, PhD Thesis, University of Michigan, Dissertation Abstracts International, 36(10), 5140B.

DE JONG K. (1998), *Evolutionary Computation: Where we are and where we're headed?* Fundamenta Informaticae, 33, IOS Press, 1–13.

DE LA HIGUERA C. (1997), *Characteristic sets for polynomial grammatical inference*, Machine Learning, 27, 125–138.

DE LA HIGUERA C. (2000), *Current trends in grammatical inference*, w: Advances in Pattern Recognition, red. F.J. Ferri i in., Joint IAPR International Workshops SSPR+SPR'2000, LNCS 1876, Springer, Berlin, 28–31.

DE LA HIGUERA C., JANODET J.C. (2001), *Inference of ω-languages from prefixes*, Proc. ALT 2001, LNCS 2225, Springer, Berlin, 364–378.

DEAN T., BASYE K., KAELBLING L., KOKKEVIS E., MARON O., ANGLUIN D., ENGELSON S. (1992), *Inferring finite automata with stochastic output functions and an application to map learning*, w: Proc. 10th Nat. Conf. on Artificial Intelligence, red. W. Swartout, San Jose, CA, MIT Press, 208–214.

DELGADO M., PEGALAJAR M.C. (2003), *A multiobjective genetic algorithm for obtaining the optimal size of a recurrent neural network for grammatical inference*, Pattern Recognition, 38, 1444–1456.

DOMINGOS P. (1995), *The RISE 2.0 system: A case study in multistrategy learning*, Technical Report 95-2, Department of Information and Computer Science, University of California.

DONNART J.Y., MEYER J.A. (1994), *A hierarchical classifier system implementing a motivationally autonomous animat*, w: From Animals to Animats 3, red. D. Cliff i in., Proc. 3rd Int. Conf. on Simulation of Adaptive Behavior SAB94, MIT Press, Cambridge, USA, 144–153.

DONNART J.Y., MEYER J.A. (1996), *Hierarchical-map Building and Selfpositioning with MonaLysa*, Adaptive Behavior, 5(1), 29–74.

DORIGO M. (1991), *Using Transputers to Increase Speed and Flexibility of Genetic-based Machine Learning Systems*, Microprocessing and Microprogramming, 34, 147–152.

DORIGO M. (1995), *Alecsys and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems*, Machine Learning, 19, 209–240.

DORIGO M., COLOMBETTI M. (1994), *Robot shaping: Developing autonomous agents through learning*, Artificial Intelligence, 2, 321–370.

DORIGO M., COLOMBETTI M. (1998), *Robot Shaping: An Experiment in Behavior Engineering*, MIT Press, Cambridge.

DORIGO M., SCHNEPF U. (1993), *Genetics-based Machine Learning and Behaviour Based Robotics: A New Synthesis*, IEEE Transactions on Systems, Man and Cybernetics, 23(1), 141–154.

DORIGO M., SIRTORI E. (1991), *Alecsys: A Parallel Laboratory for Learning Classifier Systems*, w: Proc. 4th Int. Conf. Genetic Algorithms ICGA91, red. L.B. Booker, R.K. Belew, Morgan Kaufmann, San Francisco, CA, 296–302.

DRUHAN B.B., MATHEWS R.C. (1989), *THIYOS: A Classifier System Model of Implicit Knowledge in Artificial Grammars*, Proc. Ann. Cog. Sci. Soc., Hillsdale, NJ.

DULEWICZ G., UNOLD O. (2002), *Evolving Natural Language Parser with Genetic Programming*, w: Hybrid Information Systems, red. A. Abraham, M. Köppen, Springer Physica, Germany, 361–377.

DULEWICZ G., UNOLD O. (2004), *A Genetic Programming for the Induction of Natural Language Parser*, w: Innovations in Intelligent Systems, red. A. Abraham i in., Studies in Fuzziness and Soft Computing, Vol. 140, Springer, 435–455.

DUPONT P. (1994), *Regular Grammatical Inference from Positive and Negative Samples by Genetic Search*, Grammatical Inference and Application, 2nd Int. Colloquium ICG-94, Springer, 236–245.

DUPONT P. (1996), *Incremental regular inference*, w: Proc. 3rd ICGI-96, red. L. Miclet, C. de la Higuera, LNAI 1147, Springer, 222–237.

DURBIN R., EDDY S., KROGH A., MITCHISON G. (1998), *Biological Sequence Analysis*, Cambridge University Press, Cambridge, UK.

EARLEY J. (1970), *An efficient context-free parsing algorithm*, Communications of the ACM, 13(2), 94–102.

ESCAZUT C., FOGARTY T.C. (1997), *Coevolving Classifier Systems to Control Traffic Signals*, w: Late Breaking Papers at the 1997 Genetic Programming Conference, red. J. Koza, Stanford University, CA.

FASS L.F. (1983), *Learning context-free languages from their structured sentences*, SIGACT News, 2(3), 24–35.

FEDERMAN F., DORCHAK S.F. (1998), *A Study for a Classifier Length and Population Size*, w: Genetic Programming 1998: Proc. 3rd Annual Conf., red. J. Koza i in., Morgan Kaufamann, San Francisco, CA, 629–634.

FELDMAN J., GIPS J., HORNING J.J., REDER S. (1969), *Grammatical inference and complexity*, Technical Report CS-125, Comp. Sci. Dept., Stanford University, CA.

FERNAU H. (2001), *Learning XML grammars*, w: Machine Learning and Data Mining in Pattern Recognition MLDM'01, red. P. Perner, LNCS 2123, Springer, 73–87.

FERNAU H. (2002), *Learning Tree Languages from Text*, w: Proc. COLT 2002, red. J. Kivinen, R.H. Sloan, LNAI 2375, Springer, 153–168.

FOGEL D., MICHALEWICZ Z. (1999), *Why Evolutionary Algorithms?* Bulletin of the European Association for Theoretical Computer Science. European Association for Theoretical Computer Science, Leiden, Holland, Vol. 68, 115–133.

FOGEL L., OWENS A., WALSH M. (1966), *Artificial Intelligence through simulated evolution*, Wiley, New York.

FORREST S. (1985), *A Study of Parallelism in the Classifier System and its Application to Classification in KL-ONE Semantic Networks*, PhD Thesis, University of Michigan, Ann Arbor, MI.

FRANCIS W.N., KUERA H. (1982), *Frequency analysis of English usage. Lexicon and grammar*, Houghton Mifflin, Boston.

FREISLEBEN B., MERZ P. (1996), *Genetic local search algorithms for solving symmetric and asymmetric traveling salesman problems*, Proc. IEEE Int. Conf. on Evolutionary Computation, IEEE-EC96, IEEE Press, 616–621.

FREY P.W., SLATE D.J. (1991), *Letter Recognition Using Holland-Style Adaptive Classifiers*, Machine Learning, 6, 161–182.

FU K.S., BOOTH T.L. (1986), *Grammatical inference: introduction and survey*, IEEE Trans. on Pattern Analysis and Machine Intelligence, 8, 343–375.

GARCIA P., SEGARRA E., VIDAL E., GALIANO I. (1994), *On the use of the morphic generator grammatical inference methodology in automatic speech recognition*, Int. J. of Pattern Recognition and Artificial Intelligence, 4, 667–685.

GAZDAR G., PULLUM G.K. (1985), *Computationally relevant properties of natural languages and their grammars*, New Generation Computing, 3, 273–306.

GÉRARD P., STOLZMANN W., SIGAUD O. (2002), *YACS: a new learning classifier system using anticipation*, Soft Computing, 6, 216–228.

GILDEA D. JURAFSKY D. (1996), *Learning Bias and Phonological-rule Induction*, Computational Linguistics, 22(4), 497–530.

GILES C.L., LAWRENCE S., TSOI A.C. (2001), *Noisy time series prediction using recurrent neural networks and grammatical inference*, Machine Learning, 44(1), 161–183.

GIORDANO J.Y. (1994), *Inference of context-free grammars by enumeration: structural containment as an ordering bias*, w: Grammatical Inference and Applications, Proc. ICGI'94, red. R.C. Carrasco, J. Oncina, LNAI 2484, Springer, Berlin, Heidelberg, 212–221.

GOLD E. (1967), *Language identification in the limit*, Information Control, 10, 447–474.

GOLD E. (1978), *Complexity of automaton identification from given data*, Information and Control, 37(3), 302–320.

GOLDBERG D. (1983), *Computer-Aided Gas Pipeline Operation using Genetic Algorithms and Rule Learning*, PhD Thesis, University of Michigan.

GOLDBERG D. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, Inc., Reading, MA, USA (Algorytmy genetyczne i ich zastosowanie, WNT, Warszawa 1995).

GOLDBERG D., KORB B., DEB K. (1989), *Messy genetic algorithms: motivation, analysis, and first results*, Complex Systems, 4, 415–444.

GOLDMAN S.A., MATHIAS H. (1996), *Teaching a smarter learner*, J. Comput. Sys. Sci., 50(1), 255–267.

GONNET G.H., TOMPA F.W. (1987), *Mind Your Grammar: a New Approach to Modeling Text*, Proc. 13th VLDB'87, Brighton, UK, 339–346.

GRAHAM S.L., HARRISON M.A., RUZZO W.L. (1980), *An improved context-free recognizer*, ACM Transactions on Programming Languages and Systems, 2(3), 415–462.

GREENVER A. (2000), *The use of a Learning Classifier System JXCS*, w: CoIL Challenge 2000 The Insurance Company Case, Technical Report 2000-09, red. P. van der Putten, M. van Someren, Leiden Institute of Adavanced Computer Science.

GREFENSTETTE J.J. (1988), *Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms*, Machine Learning, Kluwer, Boston, Vol. 3, 225–245.

GREFENSTETTE J.J., RAMSEY C.L., SCHULTZ A.C. (1990), *Learning Sequential Decision Rules Using Simulation Models and Competition*, Machine Learning, Kluwer, Boston, Vol. 5, 1990, 355–381.

GRUAU F. (1992), *Cellular Encoding of Genetic Neural Networks*, Technical Report 92-91, Ecole Normale Superieure de Lyon, Institut IMAG.

GRUAU F., WHITLEY L.D., PYEATT L. (1996), *A comparison between cellular encoding and direct encoding for genetic neural networks*, ACGP-1, MIT Press, 81–89.

GRÜNWALD P. (1996), *A minimum description length approach to grammar inference*, w: Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing, red. G. Wermter, E. Riloff, LNAI 1040, 203–216.

GRÜNWALD P. (2005), *A tutorial introduction to the minimum description length principle*, w: Advances in Minimum Description Length: Theory and Applications, red. P. Grünwald i in., MIT Press.

GUERRA-SALCEDO C., CHEN S., WHITLEY L.D., SMITH S. (1999), *Fast and accurate selection using hybrid genetic strategies*, Proc. 1999 Congress on Evolutionary Computation, 177–184.

HANDLEY S. (1995), *Predicting whether or not a nucleic acid sequence is an E. coli promoter region using genetic programming*, Proc. 1st Int. Symp. Intelligence in Neural and Biological Systems (INBS'95), IEEE Comp. Soc. Press, 122–127.

HANNENHALLI S., LEVY S. (2001), *Promoter prediction in the human genome*, Proc. 9th Int. Conf. Intelligent Systems for Molecular Biology, 17(1), Bioinformatics, Copenhagen, Denmark, 90–96.

HARLEY C., REYNOLDS R., (1987), *Analysis of E. Coli Promoter Sequences*, Nucleic Acids Research, 15, 2343–2361.

HARRISON M.A. (1978), *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA.

HARTLEY A. (1999), *Accuracy-based fitness allows similar performance to humans in static and dynamic classification environments*, w: Proc. Genetic and Evolutionary Computation Conference GECCO-00, red. W. Banzhaf i in., Morgan Kaufmann, San Francisco, CA, 266–273.

HEMPHILL C.T., GODFREY J.J., DODDINGTON G.R. (1990), *The ATIS spoken language systems pilot corpus*, w: DARPA Speech and Natural Language Workshop, Morgan Kaufmann, Hidden Valley, PA.

HENRICHSEN P.J. (2002), *GraSp: Grammar Learning from unlabelled speech corpora*, w: Proc. CoNLL-2002, red. D. Roth, A. van den Bosch, Taipei, Taiwan, 22–28.

HOFFMANN F. (1997), *Entwurf von Fuzzy-Reglern mit Genetischen Algorithmen*, Deutscher Universitäts. Verlag, Wiesbaden.

HOFFMANN J. (1993), *Vorhersage und Erkenntnis* (*Anticipation and Cognition*), Goettingen, Germany.

HOLLAND J. (1975), *Adaptation in Natural and Artificial Systems*, Ann Arbor, University of Michigan.

HOLLAND J. (1976), *Adaptation*, w: Progress in theoretical biology, red. R. Rosen, F.M. Snell, New York, Plenum.

HOLLAND J. (1980), *Adaptive Algorithms for Discovering and Using General Patterns in Growing Knowledge Bases*, International Journal of Policy Analysis and Information Systems, 4(3), 245–268.

HOLLAND J. (1986), *Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems*, w: Machine Learning, an artificial intelligence approach, Vol. II, red. R.S. Michalski i in., Morgan Kaufmann, 593–623.

HOLLAND J. (1992), *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge (MA), 2nd edition.

HOLLAND J., BOOKER L.B., COLOMBETTI M., DORIGO M., GOLDBERG D.E., FORREST S., RIOLO R.L., SMITH R.E., LANZI P.L., STOLZMANN W., WILSON S.W. (2000), *What is a Learning Classifier System?* w: Learning Classifier Systems. From Foundations to Applications, red. P.L. Lanzi i in., LNCS 1813, Springer, Berlin, Heidelberg, 3–32.

HOLLAND J.H., REITMAN J.S. (1978), *Cognitive Systems Based on Adaptive Algorithms*, w: Pattern-Directed Inference Systems, red. D.A. Waterman, F. Hayes-Roth, Academic Press, New York.

HOLMES J. (1996), *Evolution-Assisted Discovery of Sentinel Features in Epidemiologic Surveillance*, PhD Thesis, Drexel University.

HOLMES J. (2000), *Learning Classifier Systems Applied to Knowledge Discovery in Clinical Research Databases*, w: Learning Classifier Systems. From Foundations to Applications, red. P.L. Lanzi i in., LNCS 1813, Springer, Berlin. 243–264.

HONG T.W. (2003), *Grammatical Inference for Information Extraction and Visualisation on the Web*, PhD Dissertation, Imperial College London, London, UK.

HONG T.W., CLARK K.L. (2001), *Using grammatical inference to automatic information extraction from the web*, Proc. 5th European Conf. on Principle of Data Mining and Knowledge Discovery PKDD 2001, LNCS 2168, 216–227.

HONOVAR V., DE LA HIGUERA C. (2001), *Introduction*, Machine Learning, 44(1), 5–7.

HOPCROFT J.E., ULLMAN J.D. (1979), *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Publishing Company, Inc. (*Wprowadzenie do teorii automatów, języków i obliczeń*, PWN, Warszawa 1994).

HORN J., GOLDBERG D., DEB K. (1994), *Implicit Niching in a Learning Classifier System: Nature's Way*, Evolutionary Computation, 2(1), 37–66.

HORNING J.J. (1969), *A study of grammatical inference*, PhD Thesis, Stanford University, USA.

HUANG C., SUN C. (2004), *Parameter Adaptation within Co-adaptive Learning Classifier Systems*, GECCO 2004, LNCS 3103, 774–784.

HUIJSEN W. (1993), *Genetic Grammatical Inference: Induction of Pushdown Automata and Context-Free Grammars from Examples Using Genetic Algorithms*, MSc Thesis, Department of Computer Science, University of Twente, Enschede, The Netherlands.

HWA R. (1999), *Supervised Grammar Induction using Training Data with Limited Constituent Information*, Proc. ACL'99, 73–79.

ISHIGAMI Y., TANI S. (1997), *VC-dimension of finite automata and commutative finite automata with k letters and n states*, Discrete Appl. Math., 74, 123–134.

ISHIZAKA H. (1990), *Polynomial time learnability of sample deterministic languages*, Machine Learning, 2(2), 151–164.

JACKENDOFF R. (2001), *Foundations of Language*, Oxford University Press, Oxford.

JAGOTA A., LYNSØ R.B., PEDERSEN C.N.S. (2001), *Comparing a hidden Markov model and a stochastic context-free grammar*, Proc. WABI'01, LNCS 2149, Springer, Berlin, 69–74.

JAVED F., BRYANT B., CREPNISEK M., MERNIK M., SPRAGUE A. (2004), *Context-free grammar induction using genetic programming*, Proc. 42nd Annual ACM Southeast Regional Conference, Huntsville, 404–405.

JELINEK F. (1998), *Statistical Methods for Speech Recognition*, MIT Press, Cambridge, MA.

JONYER I., HOLDER L.B. COOK D.J. (2004, *MDL-Based Context-Free Graph Grammar Induction and Applications*, Int. J. of Artificial Intelligence Tools, Vol. 13, No. 1, 65–79.

JUDD K.L., TESFATSION L. (2005), *Agent-Based Computational Economics, Handbook of Computational Economics*, Vol. 2, Elsevier, North-Holland.

JURAFSKY D., MARTIN J.H. (2000), *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Prentice Hall series in Artificial Intelligence, Prentice Hall, Upper Saddle River, NJ.

KAELBLING L., LITTMAN M., MOORE A. (1996), *Reinforcement Learning: a Survey*, J. of Artificial Intelligence Research, 4, 237–285.

KAMMEYER T.E., BELEW R.K. (1996), *Stochastic context-free grammar induction with a genetic algorithm using local search*, Technical Report CS96-476, Cognitive Computer Science Research Group, University of California, San Diego.

KANAZAWA M. (1995), *Learnable classes of categorial grammars*, PhD Thesis, Stanford University.

KASAMI T. (1965), *An efficient recognition and syntax algorithm for context-free languages*, Scientific Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

KATAGAMI D., YAMADA S. (2000), *Interactive Classifier System for Real Robot Learning*, IEEE International Workshop on Robot-Human Interaction (ROMAN-2000), Osaka, Japan, 2000, 258–263.

KELLER B., LUTZ R. (1997), *Evolving stochastic context-free grammars from examples using a minimum description length principle*, Workshop on Automata Induction, Grammatical Inference and Language Acquisition ICML-97.

KELLER B., LUTZ R. (2005), *Evolutionary induction of stochastic context-free grammars*, Pattern Recognition, 38, 1393–1406.

KHARBAT F., BULL L., ODEH M. (2005), *Revisiting Genetic Selection in the XCS Learning Classifier System*, Learning Classifier Systems Group Technical Report UWELSCG05-003, University of the West England, Bristol, UK.

KINNEAR K.E. Jr. (1993), *Evolving a sort: Lessons in genetic programming*, Proc. 1993 Int. Conf. on Neural Networks, Vol. 2, IEEE Press.

KIRBY S. (2002), *Natural Language from Artificial Life*, Artificial Life, 8(2), 185–215.
KLEIN D., MANNING C. (2001), *Distributional phrase structure induction*, Proc. CoNLL 2001, 113–121.
KLEIN D., MANNING C. (2003), *Fast exact inference with a factored model for natural language parsing*, Advances in Neural Information Processing Systems, 15, MIT Press.
KLEIN D., MANNING C. (2005), *Natural language grammar induction with a generative constituent--context model*, Pattern Recognition, 38, 1407–1419.
KNOBE B., KNOBE K. (1976), *A method for inferring context-free grammars*, Inf. Contr., 2(2), 129–146.
KOFTA W. (1999), *Podstawy inżynierii genetycznej*, Prószyński i S-ka, Warszawa.
KÖPPEN M., FRANKE K., UNOLD O. (2001), *A survey on fuzzy morphology*, Pattern Recognit. Image Anal., Vol. 11, No. 1, 195–197.
KORKMAZ E.E., UCOLUK G. (2001), *Genetic Programming for Grammar Induction*, Proc. Genetic and Evolutionary Computation Conference GECCO-2001, Morgan Kaufmann, San Francisco, CA, 180.
KOSALA R., VAN DEN BUSSCHE J., BRUYNOOGHE M., BLOCKEEL H. (2002), *Information extraction in structured documents using tree automata induction*, Proc. PKDD 2002, 299–310.
KOSHIBA T., MÄKINEN E., TAKADA Y. (1997), *Learning deterministic even linear languages from positive examples*, Theoret. Comput. Sci., 185(1), 63–79.
KOVACS T.M. (2000), *Strength or Accuracy? Fitness Calculation in Learning Classifier Systems*, w: Learning Classifier Systems: From Foundations to Applications, red. P.L. Lanzi i in., LNCS 1813, Springer, Berlin, 143–160.
KOVACS T.M. (2002a), *A Comparison of Strength and Accuracy-Based Fitness in Learning Classifier Systems*, PhD Thesis, University of Birmingham.
KOVACS T.M. (2002b), *Two Views of Classifier Systems*, w: Advances in Learning Classifier Systems, red. P.L. Lanzi i in., LNCS 2321, Springer, Berlin, Heidelberg, 74–87.
KOZA J. (1992), *Genetic Programming*, MIT Press, Cambridge, MA.
KUNGAS P. (2001), *Learning State Machines in the Robot Moving Context*, Proc. NBR'2000, Trondheim.
KWAŚNICKA H. (1999), *Obliczenia ewolucyjne w sztucznej inteligencji*, Oficyna Wydawnicza PWr., Wrocław.
KWAŚNICKA H., SPIRYDOWICZ A. (2004), *Uczący się komputer. Programowanie gier logicznych*, Oficyna Wydawnicza PWr., Wrocław.
LANG K., PEARLMUTTER B., PRICE R. (1998), *Results of the Abbadingo One DFA Learning Competition and a New Evidence Driven State Merging Algorithm*, Proc. Int. Colloquium on Grammatical Inference ICGA-98, LNAI 1433, Springer, Berlin, Heidelberg, 1–12.
LANGLEY P., STROMSTEN S. (2000), *Learning context-free grammars with simplicity bias*, Proc. ECML 2000, 11th European Conf. on Machine Learning, LNCS 1810, Springer, Berlin, 220–228.
LANKHORST M.M. (1994), *Grammatical Inference with a Genetic Algorithm*, w: Proc. 1994 EUROSIM Conf. on Massively Parallel Processing Applications and Development, red. L. Dekke i in., Elsevier, Amsterdam, 423–430.
LANKHORST M.M. (1995), *A Genetic Algorithm for the Induction of Nondeterministic Pushdown Automata*, Computing Science Reports CS-R 9502, Department of Computing Science, University of Groningen.
LANZI P.L. (1998), *Adding Memory to XCS*, Proc. IEEE Conf. on Evolutionary Computation ICEC98, IEEE Press, Piscataway, NJ.
LANZI P.L. (1999a), *Extending the Representation of Classifier Conditions*. Part I: *From Binary to Messy Coding*, w: Proc. Genetic and Evolutionary Computation Conf. GECCO-99, red. W. Banzhaf i in., Morgan Kaufmann, Orlando, FL, 337–344.
LANZI P.L. (1999b), *Extending the Representation of Classifier Conditions*. Part II: *From Messy Coding to S-Expressions*, w: Proc. Genetic and Evolutionary Computation Conf. GECCO-99, red. W. Banzhaf i in., Morgan Kaufmann, Orlando, FL, 345–352.

LANZI P.L., RIOLO R.L. (2000), *A Roadmap to the Last Decade of Learning Classifier System Research*, w: Learning Classifier Systems. From Foundations to Applications, red. P.L. Lanzi, i in., LNAI 1813, Springer, Berlin, Heidelberg, 33–62.

LANZI P.L., WILSON S.W. (2000), *Toward Optimal Classifier System Performance in Non-Markov Environments*, Evolutionary Computation, 8(4), 2000, 293–418.

LARI K., YOUNG S.J. (1990), *The estimation of stochastic context free grammars using the inside–outside algorithm*, Comput. Speech Lang., 4, 35–56.

LEBARON B., ARTHUR W.B., PALMER R. (1999), *The Time Series Properties of Artificial Stock Market*, J. of Economic Dynamics and Control, 23, 1487–1516.

LEE L. (2002), *Fast Context-Free Grammar Parsing Requires Fast Boolean Matrix Multiplication*, J. of the ACM, 49(1), 1–15.

LEE S. (1996), *Learning of context-free languages: A survey of the literature*, Technical Report, TR-12-96, Center for Research in Computing Technology, Harvard University, Cambridge, MA.

LEUNG S.W., MELLISH C., ROBERTSON D. (2001), *Basic gene grammars and DNA-chart parser for language processing of Escherichia coli promoter DNA sequences*, Bioinformatics, 17, 226–236.

LEVY L.S, JOSHI A.K. (1978), *Skeletal structural description*, Inf. Contr., 2(2), 192–211.

LI M., VITANYI P.M.B. (1993), *An Introduction to Kolmogorov Complexity and its Applications*, Springer, Heidelberg.

LI M., VITANYI P.M.B. (1995), *Computational Machine Learning in Theory and Praxis*, NeuroCOLT Technical Report Series, NC-TR-95-052, University of London.

LIN D. (1995), *A dependency-based method for evaluating broad-coverage parsers*, Proc. IJCAI-95, 1420–1425.

LIN L. (1993), *Reinforcement learning for robots using neural networks*, Technical Report, CMU-CS-93-103, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

LITTMAN M.L. (1994), *Memoryless policies: Theoretical limitations and practical results*, w: From Animals to Animats 3, Proc. 3rd Int. Conf. Simulation of Adaptive Behavior, red. D. Cliff i in., MIT Press, Cambridge, MA, 238–245.

LLORA X. (2002), *Genetic Based Machine Learning using Fine-grained Parallelism for Data Mining*, PhD Thesis, Ramon Llull University, Barcelona.

LOSEE R.M. (1996), *Learning syntactic rules and tags with genetic algorithms for information retrieval and filtering: an empirical basis for grammatical rules*, Information Processing and Management, 32(2), Elsevier Science Ltd., 185–197.

LUCAS S. (1993), *Biased chromosomes for grammatical inference*, Proc. Natural Algorithms in Signal Processing, IEE Workshop, Danbury Park.

LUCAS S. (1994), *Structuring Chromosomes for Context-Free Grammar Evolution*, Proc. 1st IEEE Int. Conf. on Evolutionary Computation, 130–135.

LUCAS S., REYNOLDS T.J. (2003), *Learning DFA: Evolution versus Evidence Driven State Merging*, Proc. Congress Evolutionary Computation, 351–358.

LUCAS S., REYNOLDS T.J. (2005), *Learning Deterministic Finite Automata with a Smart State labeling Evolutionary Algorithm*, IEEE Trans. on Pattern Analysis and Machine Intelligence, 27 (7), 1–12.

LUCAS S., VIDAL E., AMIRI A., HANLON S., AMENGUAL J.C. (1994), *A comparison of syntactic and statistical techniques for off-line ocr*, Proc. ICGI-94, LNAI 862, Springer, Berlin, Heidelberg, 168–179.

LUKE S., HAMAHASHI S., KITANO H. (1999), *"Genetic" Programming*, w: Proc. Genetic and Evolutionary Computation Conference GECCO-99, red. W. Banzhaf in., Morgan Kaufmann, 1098–1105.

LUKE S., SPECTOR L. (1996), *Evolving Graphs and Networks with Edge Encoding: Preliminary Report*, w: Late-breaking Papers of Genetic Programming 96, red. J. Koza, Stanford Bookstore, San Francisco, CA, 117–124.

LUZEAUX D. (1996), *Machine learning applied to the control of complex systems*, Proc. 8th Int. Conf. on Artificial Intelligence and Expert Systems Application, Paris, France.

LYNGSØ R.B., PEDERSEN C.N.C. (2001), *Complexity of comparing hidden Markov models*, Proc. ISAAC'01.

LYNGSØ R.B., PEDERSEN C.N.C., NIELSEN H. (1999), *Metrics and similarity measures for hidden Markov models*, Proc. ISMB'99, AAA I Press, Menlo Park, USA, 178–186.

MAGERMAN D. (1995), *Statistical decision-tree models for parsing*, Proc. ACL 33, 276–283.

MÄKINEN E. (1990), *The grammatical inference problem for the Szilard languages of linear grammars*, IPL, 2(4), 203–206.

MÄKINEN E. (1996), *A note on the grammatical inference problem for even linear languages*, Fundamenta Informaticae, 25 (2), 175–182.

MARCUS M.P., SANTORINI B., MARCINKIEWICZ M.A. (1993), *Building a large annotated corpus of English: The Penn treebank*, Comput. Linguist., 19, 313–330.

MARIMON R., MCGRATTAN E., SARGENT T.J. (1990), *Money as a Medium of Exchange in an Economy with Artificially Intelligent Agents*, J. of Economic Dynamics and Control, 14, 329–373.

MCCALLUM R.A. (1996), *Hidden state and reinforcement learning with instance-based state identification*, IEEE Trans. on Systems, Man and Cybernetics – Part B (Special issue on Learning Autonomous Robots), 26(3), 464–473.

MCMAHON A., SCOTT D., BROWNE W. (2005), *An Autonomous Explore/Exploit Strategy*, GECCO-05, Proc. 2005 Workshop on Genetic and Evolutionary Computation, Washington, D.C., 103–108.

MCNAUGHTON R. (1967), *Parenthesis grammars*, JACM 2(3), 490–500.

MEIJS W. (1993), *Inferring grammar from lexis: machine-readable dictionaries as sources of wholesale syntactic and semantic information*, w: Grammatical Inference – theory, applications and alternatives, red. S. Lucas, Colloquium Proceedings, No. 1993/092, London, UK, 1–5.

MERNIK M., GERLIČ G., ŽUMER V., BRYANT B. (2003), *Can a Parser be Generated from Examples?* Proc. ACM Symposium on Applied Computing, 1063–1067.

MERNIK M., LENIČ M., AVDIČAUŠEVIĆ E., ŽUMER V. (2002), *LISA: An Interactive Environment for Programming Language Development*, 11th Int. Conf. on Compiler Construction, CC'2002, LNCS 2304, 1–4.

MICHALEWICZ Z. (1996), *Genetic algorithms + Data structures = Evolution Programs*, Springer, Berlin, Heidelberg (*Algorytmy genetyczne + struktury danych = programy ewolucyjne*, WNT, Warszawa 1996).

MIKOŁAJACZK B. (1985), *Algebraiczna i strukturalna teoria automatów*, PWN, Warszawa.

MILLER J.H., HOLLAND J. (1991), *Artificial adaptive agents in economic theory*, American Economic Review, 81(2), 365–370.

MITLÖHNER J. (1996), *Classifier systems and economic modeling*, Proc. APL 96 Conf. on Design Future, Vol. 26(4), 77–86, Lancaster, UK.

MOHRI M. (1997), *Finite-state transducers in language and speech processing*, Computational Linguistics, 23(3), 269–311.

MOHRI M. (2000), *Minimization algorithms for sequential transducers*, Theoretical Computer Science, 234, 177–201.

MORGAN N., BOURLARD H. (1995), *Continuous speech recognition: An introduction to the hybrid HMM/connectionist approach*, IEEE Signal Processing Magazine, 12.

MUGGLETON S. (1999), *Inductive Logic Programming*, w: The MIT Encyclopedia of the Cognitive Science MITESC, MIT Press.

MÜHLENBEIN H., SCHLIERKAMP-VOOSEN D. (1993), *Predictive models for the breeder genetic algorithm I: Continuous Parameter Optimization*, Evolutionary Computation, 1(1), 25–49.

MÜHLENBEIN H., SCHLIERKAMP-VOOSEN D. (1995), *Analysis of Selection, Mutation and Recombination in Genetic Algorithms*, w: Evolution as a Computational Process, red. W. Banzhaf, F.H. Eeckman, LNCS 899, Springer, Berlin, Heidelberg, 188–214.

NAGASAKA I., TAURA T. (1997), *3D Geometric Representation for Shape Generation using Classifier System*, Proc. 2nd Annual Conf. on Genetic Programming, Morgan Kaufmann, 515–520.

NEVILL-MANNING C., WITTEN I. (1997), *Identifying hierarchical structure in sequences: A linear-time algorithm*, J. of Artificial Intelligence Research, (7), 67–82.

NEY H. (1992), *Stochastic grammars and pattern recognition*, w: Proc. NATO Advanced Study Institute, red. P. Laface, R.D. Mori, Springer, Berlin, Heidelberg, 313–344.

NIX A.E., VOSE M.D. (1992), *Modeling genetic algorithms with Markov chains*, Annals of Mathematics and Artificial Intelligence, 5, 79–88

NOWAK M.A., KOMAROVA N.L., NIYOGI P. (2002), *Computational and evolutionary aspects of language*, Nature, 417, 611–617.

O'NEILL M. (1989), *Escherichia coli promoters*: II. *A spacing class-dependent promoter search protocol*, J. of Biological Chemistry, 264, 5531–5534.

OFLAZER K. (1996), *Error-tolerant finite-state recognition with application to morphological analysis and spelling correction*, Computational Linguistics, 22(1), 73–89.

OHLER U., NIEMANN H. (2001), *Identification and analysis of eukaryotic promoters: recent computational approaches*, Trends in Genetics, 17(2), 56–60.

ONCINA J., GARCIÀ P. (1992), *Inferring regular languages in polynomial update time*, w: Pattern recognition and image analysis, red. N. Perez i in., Singapore, World Scientific, 49–61.

ONCINA J., GARCIA P., VIDAL E. (1993), *Learning subsequential transducers for pattern recognition interpretation tasks*, IEEE Trans. on Pattern Analysis and Machine Intelligence, 15(5), 448–458.

OSBORNE M. (1999), *MDL-based DCG induction for NP identification*, w: Proc. CoNLL-99, red. M. Osborne i in., EACL, Bergen, Norway, 61–68.

PAO T., CARR J. (1978), *A solution of the syntactic induction-inference problem for regular languages*, Computer Languages, 3, 53–64.

PAREKH R.G., HONAVAR V.G. (1996), *An incremental interactive approach for regular grammar inference*, Proc. 3rd ICGI-96, LNAI 1147, Springer, Berlin, Heidelberg, 238–250.

PARTEE B., TER MEULEN A., WALL R.E. (1993), *Mathematical methods in linguistics*, Kluwer.

PASKIN M.A. (2001), *Grammatical Bigrams*, w: Advances in Neural Information Processing Systems, red. Dietterich T. i in., 14, MIT Press, Cambridge.

PAWLAK M. (1999), *Algorytmy ewolucyjne jako narzędzie harmonogramowania produkcji*, PWN, Warszawa.

PEDERSEN A.G., BALDI P., CHAUVIN Y., BRUNAK S. (1999), *The biology of eukaryotic promoter prediction – a review*, Comput. Chem., 23, 191–207.

PEREIRA F., SCHABES Y. (1992), *Inside–outside reestimation from partially bracketed corpora*, Proc. ACL, 30, 128–135.

PICONE J. (1990), *Continuous speech recognition using hidden Markov models*, IEEE ASSP Magazine, 7(3), 26–41.

PIECH H. (2003), *Wykorzystanie narzędzi stochastycznych do realizacji algorytmów genetycznych*, Wydawnictwo Politechniki Częstochowskiej, Częstochowa.

PINKER S. (1979), *Formal Models of Language Learning*, Cognition, 7, 217–283.

PITT L. (1989), *Inductive inference, DFAs and computational complexity*, Proc. Int. Workshop on Analogical and Inductive Inference, LNAI 397, Springer, London, UK, 18–44.

PITT L., WARMUTH M. (1988), *Reductions among prediction problems: On the difficulty of predicting automata*, Proc. 3rd Conf. on Structure in Complexity Theory, 60–69.

PORAT F., FELDMAN J. (1991), *Learning automata from ordered examples*, Machine Learning, 7, 109–138.

POWERS D., (1997), *Machine learning of natural language*, w: Tutorial Notes, ACL/EACL 97, Madrid, Spain.

PRZEPIÓRKOWSKI A., BAŃSKI P., DĘBOWSKI Ł., HAJNICZ E., WOLIŃSKI M. (2003), *Konstrukcja korpusu IPI*, PAN, Polonica XXII–XXIII, 33–38.

PULLUM G.K., GAZDAR G. (1982), *Natural languages and context-free languages*, Linguist. Phil., 4, 471–504.

QUIRIN A. (2002), *Découverte de règles de classification: classifieurs évolutifs*, Mémoire DEA d'Informatique, Université Louis Pasteur, LSIIT UMR-7005 CNRS, Strasbourg.

RA D.Y., STOCKMAN G.C. (1999), *A new one pass algorithm for estimating stochastic context-free grammars*, Inf. Process. Lett., 72, 37–45.

RECHENBERG I. (1973), *Evolutionsstrategie: Optimierung Technischer System nach Prinzipien der biologischen Evolution*, Fromman-Holzboog, Stuttgart.

RICHARDS R.A., SHEPPARD S.D. (1992), *Two-dimensional Component Shape Improvement via Classifier System*, Artificial Intelligence in Design'92, Kluwer, Dordrecht.

RICHARDS R.A., SHEPPARD S.D. (1996), *Three-dimensional Shape Optimization Utilizing a Learning Classifier System*, w: Genetic Programming 1996, Proc. 1st Annual Conf. Stanford University, CA, red. J. Koza i in., 539–546.

RICO-JUAN J.R., CALERA-RUBIO J., CARRASCO R.C. (2002), *Stochastic k-testable tree languages and applications*, w: Grammatical Inference: Algorithms and Applications, Proc. ICGI'00, red. P. Adriaans i in., LNAI 2484, Springer, Berlin, Heidelberg, 199–212.

RIOLO R.L. (1987), *Bucket Brigade Performance:* I. *Long Sequences of Classifiers*, Proc. 2nd Int. Conf. on Genetic Algorithms ICGA-87, Lawrence Erlbaum Associates, 184–195.

RIOLO R.L. (1988), *CFS-C: A Package of Domain-Independent Subroutines for Implementing Classifier Systems in Arbitrary User-Defined Environments*, Technical Report, University of Michigan.

RIOLO R.L. (1989), *The Emergency of coupled sequences of Classifiers*, w: Proc. 3rd Int. Conf. on Genetic Algorithms ICGA-89, red. J.D. Schaffer, George Mason University, Morgan Kaufmann, 256–264.

RIOLO R.L. (1990), *Lookahead Planning and Latent Learning in a Classifier System*, w: From Animals to Animats 1, Proc. 1st Int. Conf. on Simulation of Adaptive Behavior SAB90, red. J.A. Meyer, S.W. Wilson, MIT Press, Cambridge, MA, 316–326.

RISSANEN J. (1989), *Stochastic Complexity in Statistical Inquiry*, Series in Computer Science, Vol. 15, World Scientific.

RIVEST R.L., SCHAPIRE R.E. (1993), *Inference of finite automata using homing sequences*, Information and Computation, 103, 299–347.

ROBERTS A. (2002), *Automatic acquisition of word classification using distributional analysis of content words with respect to function words*, Technical Report, School of Computing, University of Leeds.

ROBERTSON G.G., RIOLO R.L. (1988), *A tale of two classifier systems*, Machine Learning, 3, 139–159.

ROCHE E., SCHABES Y. (1995), *Deterministic part-of-speech tagging with finite-state transducers*, Computational Linguistics, 21(2), 227–253.

RON D., RUBINFELD R. (1995), *Exactly learning automata with small cover time*, Machine Learning, 27, 69–96.

RON D., SINGER Y., TISHBY N. (1994), *Learning probabilistic automata with variable memory length*, Proc. 7th Annual ACM Conf. on Computational Learning Theory, ACM Press, NJ, 35–46.

RON D., SINGER Y., TISHBY N. (1995), *On the learnability and usage of acyclic probabilistic finite automata*, Proc. COLT 1995, 31–40.

RUDOLPH G. (1998), *Finite Markov chain results in evolutionary computation: A tour d'horizon*, Fundamenta Informaticae, 34, 1–22.

RUSSEL S., NORVIG P. (1995), *Artificial Intelligence. A Modern Approach*, Prentice Hall International.

RUTKOWSKA D. (1997), *Inteligentne systemy obliczeniowe. Algorytm genetyczne i sieci neuronowe w systemach rozmytych*, Akademicka Oficyna Wydawnicza PLJ, Warszawa.

RYAN C., COLLINS J.J., O'NEILL M. (1998), *Grammatical Evolution: Evolving Programs for an Arbitrary Language*, Proc. 1st European Workshop on Genetic Programming, LNCS 1391, Springer, Berlin, Heidelberg, 83–95.

SAKAKIBARA Y. (1987), *Inferring parsers of context-free languages from structural examples*, Technical Report, 81, Fujitsu Limited, Int. Inst. for Advanced Study of Social Information Science, Numazu, Japan.

SAKAKIBARA Y. (1990), *Learning Context-Free Grammars from Structural Data in Polynomial Time*, Theo. Comp. Sci., 76, 223–242.

SAKAKIBARA Y. (1992), *Efficient learning of context-free grammars from positive structural examples*, Inf. Comput., 2(1), 23–60.

SAKAKIBARA Y. (1997), *Recent advances in grammatical inference*, Theoretical Computer Science, 185, 15–45.

SAKAKIBARA Y. (2005), *Learning context-free grammars using tabular representations*, Pattern Recognition, 35, 1372–1383.

SAKAKIBARA Y., BROWN M., HUGHEY R., MIAN I.S., SJOLANDER K., UNDERWOOD R., HAUSSLER D. (1994), *Stochastic context-free grammars for tRNA modeling*, IEEE Trans. Comput., C-22 (5), 442–450.

SAKAKIBARA Y., BROWN M., HUGHLEY R., MIAN I., SJOLANDER K., UNDERWOOD R., HAUSSLER D, (1994), *Stochastic context-free grammars for tRNA modeling*, Nucleic Acids Research, 22, 5112–5120.

SAKAKIBARA Y., KONDO M. (1999), *GA-based learning of context-free grammars using tabular representations*, Proc. 16th Int. Conf. on Machine Learning ICML-99, 354–360.

SAKAKIBARA Y., MURAMATSU H. (2000), *Learning context-free grammars from partially structured examples*, Proc. ICGI'00, LNAI 1891, 229–240.

SALOMAA A. (1969), *Probabilistic and weighted grammars*, Inf. Control., 15, 529–544.

SALVADOR I., BENEDI J.M. (2002), *RNA modeling by combining stochastic context-free grammars and n-gram models*, Int. J. of Pattern Recognition and Artificial Intelligence, 16(3), 309–316.

SÀNCHEZ J.A., BENEDI J.M. (1997), *Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformation*, IEEE Trans. Pattern. Anal. Mach. Intell., 19(9), 1052–1055.

SANZA C., DESTRUEL C., DUTHEN Y. (1998), *A learning method for adaptation and evolution in virtual environments*, Proc. 3rd Int. Conf. on Computer Graphics and Artificial Intelligence, Limoges, France.

SAOUDI A., YOKOMORI T. (1993), *Learning local and recognizable ω-languages and monadic logic programs*, Proc. EUROCOLT, Oxford University Press, 157–169.

SATTERFIELD T. (1999), *Bilingual Selection of Syntactic Knowledge: Extending the Principles and Parameters Approach*, Kluwer, Amsterdam.

SAXON S., BARRY A. (1999), *XCS and the Monk's Problems*, w: Learning Classifier Systems: From Foundation to Applications, LNAI 1813, red. P.L. Lanzi i in., Springer, Berlin, Heidelberg, 223–242.

SCHAFFER J.D., WHITLEY D., ESCHELMAN L.J. (1992), *Combinations of genetic algorithms and neural networks: a survey of the state of the art*, w: COGANN, red. D. Whitley, J.D. Schaffer, IEEE Press, 1–37.

SCHAPIRE R.E. (1990), *Pattern languages are not learnable*, w: Proc. 3rd Annual ACM Workshop on Computational Learning Theory, red. M.A. Fulk, J. Case, Morgan Kaufmann, San Mateo, CA, 122–129.

SCHWEFEL H.P. (1977), *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, Vol. 26 of Interdisciplinary Systems Research, Birkhauser, Basel.

SEARLS D.B. (1993), *The computational linguistics of biological sequences*, Artificial Intelligence and Molecular Biology, 47–120.

SEARLS D.B. (2002), *The language of genes*, Nature, 420, 211–217.

SEGINER Y. (2003), *Learning context free grammars in the limit aided by the sample distribution*, Proc. ECML, Dubrovnik, Croatia, 77–88.

SEMEPERE J.M. GARCIA P., (1994), *A characterisation of even linear languages and its application to the learning problem*, w: Grammatical Inference and Applications, Proc. ICGI'94, red. R.C. Carrasco, J. Oncina, LNAI 862, Springer, Berlin, Heidelberg, 38–44.

SEMEPERE J.M., NAGARAJA G. (1998), *Learning a subclass of linear languages from positive structural information*, w: Grammatical Inference, Proc. ICGI'98, red. V. Honovar, G. Slutski, LNAI 1433, Springer, Berlin, Heidelberg, 162–174.

SEN S. (1993), *Improving classification accuracy through performance history*, w: Proc. 5th Int. Conf. on Genetic Algorithms ICGA93, red. S. Forrest, Morgan Kaufmann, 652–652.

SEN S. (1996), *Modeling human categorization by a simple classifier system*, 1st Online Workshop on Soft Computing WSC1, August 19–30.

SEN S., JANAKIRAMAN J. (1992), *Learning to construct pushdown automata for accepting deterministic context-free languages*, w: Applications of Artificial Intelligence, red. G. Biswas, SPIE, Vol. 1707, X: Knowledge-Based Systems, 207–213.

SHIEBER S.M. (1985), *Evidence against the context-freeness of natural language*, Linguist. Phil., 8, 333–343.

SHU D., SCHAEFFER J. (1989), *VCS: Variable Classifier System*, w: Proc. 3rd Int. Conf. on Genetic Algorithms, red. J.D. Schaffer, 334–339.

SHU D., SCHAEFFER J. (1991), *HCS: Adding Hierarchies to Classifier Systems*, Proc. 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, USA, 1991, 339–345.

SMITH R.E. (1994), *Memory exploitation in learning classifier systems*, Evolutionary Computation, 2(3), 1994, 199–220.

SMITH R.E., DIKE B.A., MEHRA R.K., RAVICHANDRAN B., EL-FALLAH A. (1999), *Classifier Systems in Combat: Two-sided Learning of Maneuvers for Advanced Fighter Aircraft*, Computer Methods in Applied Mechanics and Engineering, Elsevier.

SMITH S.F. (1980), *A Learning System Based on Genetic Algorithm*, PhD Dissertation, University of Pittsburgh.

SMITH S.F. (1983), *Flexible Learning of Problem Solving Heuristics through Adaptive Search*, Proc. 8th Int. Conf. on Artificial Intelligence, Morgan Kaufmann, San Mateo, CA.

SMITH T.C., WITTEN I.H. (1995), *A genetic algorithm for the induction of natural language grammars*, Proc. IJCAI-95 Workshop on New Approaches to Learning for Natural Language Processing, Montreal, Canada, 17–24.

SMITH T.C., WITTEN I.H. (1996), *Learning Language Using Genetic Algorithms*, w: Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing, red. S. Wermter i in., LNAI 1040, 133–145.

SOLAN Z., HORN D., RUPPIN E., EDELMAN S. (2005), *Unsupervised learning of natural languages*, PNAS, 102(33), 11629–11634.

SOLOMONOFF R. (1959), *A new method for discovering the grammars of phrase structure languages*, Proc. Int. Conf. on Information Processing, Paris, UNESCO, 285–290.

SOLOMONOFF R. (1964), *A formal theory of inductive inference*, Information and Control, 7, 224–254.

STOLCKE A. (1994), *Bayesian Learning of Probabilistic Language Models*, PhD Thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley.

STOLCKE A. (1995), *An efficient probabilistic context-free parsing algorithm that computes prefix probabilities*, Comput. Linguist., 21(2), 165–201.

STOLCKE A., OMOHUNDRO S.M. (1994), *Inducing probabilistic grammars by Bayesian model merging*, Proc. 2nd Int. Colloquium on Grammatical Inference, Springer, Berlin, Heidelberg.

STOLZMANN W. (1997), *Anticipative Classifier Systems*, PhD Thesis, Fachbereich Mathematik/ Informatik, University of Osnabrueck.

STOLZMANN W. (1998), *Anticipatory classifier systems*, Proc. 3rd Annual Genetic Programming Conf., Morgan Kaufmann, San Francisco, CA, 658–664.

STOLZMANN W. (1999), *Latent learning in Khepera robots with anticipatory classifier systems*, w: Proc. 1999 Genetic and Evolutionary Computation Conf. Workshop Program, red. A.S. Wu, Morgan Kaufmann, San Francisco, CA, 290–297.

STOLZMANN W. (2000), *An Introduction to Anticipatory Classifier Systems*, w: Learning Classifier Systems. From Foundations to Applications, red. P.L. Lanzi i in., LNAI 1813, Springer, Berlin, Heidelberg, 175–194.

SUTTON R. (1988), *Learning to predict by the methods of temporal differences*, Machine Learning, 3, 9–44.

TAKADA Y. (1987), *A constructive method for grammatical inference of linear languages based on control sets*, Research Report 78, Int. Inst. for Advanced Study in the Soc. Info. Sci.

TAKADA Y. (1988), *Grammatical inference for even linear languages*, IPL 2(4), 193–199.

TAKADA Y. (1994), *A hierachy of language families learnable by regular language learners*, w: Grammatical Inference and Application, Proc. ICGI'94, red. R.C. Carrasco, J. Oncina, LNAI 862, Springer, Berlin, Heidelberg, 16–24.

TANATSUGU K. (1987), *A grammatical inference for context-free languages based on self-embedding*, Bull. Informatics and Cybernetics, 2(3–4), 149–163.

TANOMARU J. (1997), *Evolving Turing Machines from Examples*, w: Artificial Evolution 3rd Conf. on AE'97, red. J.K. Hao i in., LNCS 1363, Springer, Berlin, Heidelberg.

THANARUK T., OKUMARU M. (1995), *Grammar Acquisition and Statistical Parsing*, J. of Natural Language Processing, 2(3).

THIERENS D. (1996), *Dimensional analysis of allele-wise mixing revisited*, w: Parallel Problem Solving From Nature, PPSN IV, red. H.M. Voigt i in., Springer, Berlin, Heidelberg, 225–265.

THOLLARD F. (2001), *Improving probabilistic grammatical inference core algorithms with post-processing techniques*, Proc. 8th Int. Conf. on Machine Learning, Williams, Morgan Kaufman, San Francisco, CA, 561–568.

THOLLARD F., DUPONT P., DE LA HIGUERA C. (2000), *Probabilistic DFA inference using Kullback-Leiber divergence and minimality*, Proc. 17th Int. Conf. on Machine Learning, Morgan Kaufman, San Francisco, CA, 975–982.

TOMITA M. (1982), *Dynamic construction of finite automata from examples using hill climbing*, w: Proc. 4th Annual Cognitive Science Conf., USA, 105–108.

TOMLINSON A., BULL L. (1998), *A Corporate Classifier System*, w: Proc. 5th Int. Conf. on Parallel Problem Solving From Nature PPSN V, red. A.E. Eiben i in., LNCS 1498, Springer, Berlin, Heidelberg, 550–559.

TOMLINSON A., BULL L. (1999), *A corporate XCS*, w: Proc. 1999 Genetic and Evolutionary Computation Conf. Workshop Program, red. A.S. Wu, Morgan Kaufmann, San Francisco, CA, 298–305.

TOWELL G.G., SHAVLIK J.W., NOORDEWIER M.O. (1990), *Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks*, Proc. 8th Conf. on AI (AAAI-90), 861–866.

TRAKHTENBROT B., BARZDIN Ya. (1973), *Finite Automata: Behavior and Synthesis*, Amsterdam, North Holland.

TSOULOS I.G, LAGARIS I.E. (2005), *Grammar inference with grammatical evolution*, Department of Computer Science, University of Ioannina, Greece, preprint.

UNOLD O. (1992), *Model formalny automatycznej dekompozycji zadań języka naturalnego na elementy bazy wiedzy dla systemu z bazowaniem wiedzy*, w: Sztuczna inteligencja (cybernetyka – inteligencja – rozwój), II Ogólnopolska Konferencja CIR'92. Cybernetics'92. PTC Zarząd Główny, WSRP w Siedlcach, Warszawa–Siedlce, 55–63.

UNOLD O. (1994), *Koncepcja lingwistycznego mechanizmu wnioskującego w systemie z dostępem w języku naturalnym*, w: Mat. VII Ogólnopolskie Konwersatorium CIR'94 (cybernetyka – inteligencja – rozwój), Warszawa–Siedlce, 256–263.

UNOLD O. (1995a), *Automatowa dekompozycja zdania języka naturalnego na elementy wiedzy w systemie dialogowym*, w: Głosowa Komunikacja Człowiek–Komputer, I Krajowa Konferencja, Wrocław, 111–116.

UNOLD O. (1996a), *Wnioskowanie na symbolicznej reprezentacji wiedzy nabywanej ze zdań języka naturalnego*, w: IX Ogólnopolskie konwersatorium nt. Sztuczna inteligencja i systemy hybrydowe CIR-11 '96 (cybernetyka – inteligencja – rozwój), ZG PTC Centrum Obsługi Badań Naukowych i Dydaktyki WSRP w Siedlcach, Instytut Badań Systemowych PAN, Siedlce–Warszawa, 37–45.

UNOLD O. (1996b), *Stratyfikacyjny system reprezentacji wiedzy*, w: Język i Technologia, red. Z. Vetulani, W. Abramowicz, Akademicka Oficyna Wydawnicza, PLJ, Warszawa, 177–182.

UNOLD O. (1997a), *Koncepcja komponentu strukturalno-semantycznego jako typ modelu redukcyjnego*, w: Inżynieria wiedzy i systemy ekspertowe, T.1., red. Z. Bubnicki, A. Grzech, Oficyna Wydawnicza PWr., Wrocław, 334–340.

UNOLD O. (1997b), *Automatic Analysis of Natural Language Texts in Man–Machine Communication*, w: Systems Development Methods for the Next Century, Plenum Publishing Corp., red. G. Wojtkowski i in., New York, 185–194.

UNOLD O. (1997c), *Zastosowanie automatu rozmytego w przetwarzaniu języka naturalnego*, w: Mat. IV KK KOWBAN'97, Wrocław–Świeradów Zdrój, 523–528.

UNOLD O. (1998a), *The automaton approach to systems gathering knowledge expressed in natural language*, Proc. Conf. on Intelligent Decision Support Systems in Management, Katowice–Wisła, University Publ. Academy of Economics, 47–58.

UNOLD O. (1998b), *A Fuzzy Automaton Approach to Dialog Systems*, Proc. IASTED International Conference ASC'98, Acta Press, Cancun, Mexico, 215–218.

UNOLD O. (1998c), *Application of Fuzzy Sets in Natural Language Processing*, Proc. 6th Congress on Intelligent Techniques and Soft Computing EUFIT'98, Aachen, Germany, 1262–1266.

UNOLD O. (1998d), *Dialog system with self-learning analyzer of natural texts*, Proc. Conf. Intelligent Decision Support Systems in Management, Katowice–Wisła, Wydawnictwo Akademii Ekonomicznej, 115–121.

UNOLD O. (1999a), *Toward fuzziness in natural language processing*, w: Advances in Soft Computing – Engineering Design and Manufacturing, red. R. Roy i in., Springer, London, 554–567.

UNOLD O. (1999b), *A genetically optimized fuzzy parser of natural language*, Proc. 2nd Int. Conf. on Intelligent Processing and Manufacturing of Materials IPMM'99, Vol. 1, Big Island, Hawaii, Piscataway, NJ: IEEE Operations Center, 277–280.

UNOLD O. (1999c), *Evolving analyzer of natural language using evolutionary computation*, Proc. Workshop Intelligent Information Systems, Ustroń–Warszawa, IPI PAN, 278–287.

UNOLD O. (1999d), *A brief sketch of an evolutionary method for developing architecture of natural language parser*, Proc. International Workshop on Soft Computing in Industry'99, IWSCI'99, Muroran, Hokkaido, Japan, 411–415.

UNOLD O. (1999e), *Toward a mechanics of self-learning parser of natural texts*, Proc. IASTED International Conference on Artificial Intelligence and Soft Computing, Acta Press, Honolulu, Hawaii, USA, 453–456.

UNOLD O. (2000), *An evolutionary approach for the design of natural language parser*, w: Soft Computing in Industrial Applications, red. Y. Suzuki i in., Springer, London, 293–297.

UNOLD O. (2003), *Context–free grammar induction using evolutionary methods*, WSEAS Trans. on Circuits and Systems, 3(2), 632–637.

UNOLD O. (2005a), *GCS – nowy model uczącego się systemu klasyfikującego*, w: Inżynieria Komputerowa, red. W. Zamojski, WKŁ, Warszawa, 60–72.

UNOLD O. (2005b), *Learning context-free language using Grammar-based Classifier System*, w: Human language technologies as a challenge for computer science and linguistics, 2nd Language & Technology Conference, red. Z. Vetulani, Poznań, Wydaw. Poznańskie, 423–426.

UNOLD O. (2005c), *Playing a toy-grammar with GCS*, w: IWINAC 2005, red. J. Mira, J.R. Álvarez, LNCS 3562, 300–309.

UNOLD O. (2005d), *Analiza wpływu wybranych parametrów na efektywność systemu GCS*, Informatyka Teoretyczna i Stosowana, 8(5), Częstochowa, 177–190.

UNOLD O. (2005e), *Context-free grammar induction with grammar-based classifier system*, Archives of Control Science, Vol. 15(LI), 4, 681–690.

UNOLD O., CIELECKI Ł. (2005a), *Grammar-based Classifier System*, w: Issues in Intelligent Systems: Paradigms, red. O. Hryniewicz i in., EXIT, Warszawa, 273–286.

UNOLD O., CIELECKI Ł. (2005b), *How to use crowding selection in Grammar-based Classifier System*, w: Proc. 5th Int. Conf. on Intelligent Systems Design and Applications, red. H. Kwaśnicka, M. Paprzycki, Los Alamitos, IEEE Computer Society Press, 126–129.

UNOLD O., DĄBROWSKI G. (2003a), *Use of learning classifier system for inferring natural language grammar*, w: Design and application of hybrid intelligent, red. A. Abraham i in., Amsterdam, IOS Press, 272–278.

UNOLD O., DĄBROWSKI G. (2003b), *Zastosowanie systemu klasyfikującego w uczeniu gramatyki języka naturalnego*, w: Inżynieria wiedzy i systemy ekspertowe, T.1., red. Z. Bubnicki, A. Grzech, Oficyna Wydawnicza PWr., Wrocław, 288–295.

UNOLD O., DULEWICZ G. (2002), *Use of edge encoding for induction of natural language parser*, Proc. 4th Int. Conf. Recent Advances in Soft Computing RASC 2002, Nottingham, UK, 93–94.

UNOLD O., TROĆ M. (2003), *Restriction Enzyme Computation*, 7th Int. Work-Conference on Artificial and Natural Neural Networks, IWANN 2003, Minorca, Balearic Islands, Spain, June 3–6, 2003, LNCS 2686, 686–693.

UNOLD O., TROĆ M. (2005), *Biomolekularne modele obliczeniowe*, w: Inżynieria Komputerowa, red. W. Zamojski, WKŁ, Warszawa, 73–85.

UNOLD O., TROĆ M., DOBOSZ T., TRUSEWICZ A. (2003), *Finite-State Molecular Computing*, 8th Int. Conf. on Implementation and Application of Automata CIAA 2003, July 16–18, 2003, Santa Barbara, CA, USA, LNCS 2759, 309–310.

UNOLD O., TROĆ M., DOBOSZ T., TRUSEWICZ A. (2004), *Extended molecular computing model*, WSEAS Trans. Biol. Biomed., Vol. 1, iss. 1, 15–19.

VALENZUELA-RENDON M. (1991), *The Fuzzy Classifier System: a Classifier System for Continuously Varying Variables*, w: Proc. 4th Int. Conf. on Genetic Algorithms ICGA91, red. L.B. Booker, R.K. Belew, Morgan Kaufmann, San Mateo, CA, 346–353.

VALIANT L. (1975), *General context-free recognition in less than cubic time*, J. of Computer and Systems Science, 10(20), 308–315.

VALIANT L. (1984), *A theory of learnable*, Communications of the ACM, 27, 1134–1142.

VALLEJO E.E., RAMOS F. (2001), *Evolving Turing Machines for Biosequence Recognition and Analysis*, w: EuroGP, red. J. Miller, LNCS 2038, 192–203.

VAN DEN BOSCH A. (1999), *Careful abstraction from instance families in memory-based language learning*, J. for Experimental and Theoretical Artificial Intelligence, 11(3), 339–368.

VAN LEHN K., BALL W. (1987), *A version space approach to learning context-free grammars*, Machine Learning, 2, 39–74.

VAN ZAANEN M. (2002), *Bootstrapping Structure into Language: Alignment-based Learning*, PhD Thesis, School of Computing, University of Leeds.

VAPNIK V.N., CHERVONENKIS A.Y. (1971), *On the uniform convergence of relative frequencies of events to their probabilities*, Theory of Probability and its Applications, 17, 264–280.

VARDI M.Y., WOLPER P. (1986), *Automata-theoretic techniques in modal logics of programs*, J. of Computer and Systems Science, 32, 183–221.

VENTURINI G. (1994), *Apprentissage Adaptatif et Apprentissage Supervisé par Algorithme Génétique*, PhD Dissertation, Universite de Paris-Sud.

VIDAL E. (1997), *Finite-state speech-to-speech translation*, Proc. Int. Conf. on Acoustic, Speech and Signal Processing, 1, Munich, Germany, 111–114.

VOSE M.D., LIEPINS G.E. (1991), *Punctuated equilibria in genetic search*, Complex Systems, 5(1), 31–44.

VRIEND N.J. (2000), *An Illustration of the Essential Difference between Individual and Social Learning and its Consequences for Computational Analyses*, J. of Economic Dynamics and Control, 24, 1–19.

WALTROUS R., KUHN G. (1992), *Induction of finite state automata using second-order recurrent networks*, w: Advances in Neural Information Processing 4, red. J. Moody i in., Morgan Kaufmann, San Francisco, CA, 309–316.

WANG J.T.L., ROZEN S., SHAPIRO B.A., SHASHA D., WANG Z., YIN M. (1999), *New techniques for DNA sequence classification*, J. of Computational Biology, 6(2), 209–218.

WATKINSON S., MANANDHAR S. (2001), *A Psychologically Plausible and Computationally Effective Approach to Learning Syntax*, the Workshop on Computational Natural Language Learning CoNLL'01, ACL/EACL 2001.

WEXLER K., CULICOVER P. (1980), *Formal Principles of Language Acquisition*, MIT Press, Cambridge, MA.

WHITLEY L.D., RANA S., HECKENDORN R.B. (1997), *Island model genetic algorithms and linearly separable problems*, Proc. AISB Workshop on Evolutionary Computation, Springer, New York, 109–125.

WICOX J.R. (1995), *Organisational Learning within a Learning Classifier System*, MSc Thesis, University of Illinois.

WIDROW B., HOFF M. (1960), *Adaptive switching circuits*, Western Electronic Show and Convention, 4(4).

WIERZCHOŃ S.T. (2001), *Sztuczne systemy immunologiczne*, Akademicka Oficyna Wydawnicza EXIT, Warszawa.

WILES J., ELMAN J. (1995), *Landscapes in recurrent networks*, w: Proc. 17th Annual Cognitive Science Conf., red. J.D. Moore, J.F. Lehman, Lawrance Erlbaum Associates.

WILSON S.W. (1985), *Knowledge Growth in an Artificial Animal*, w: Proc. Int. Conf. on Genetic Algorithms and their Applications, red. J.J. Grefenstette, Pittsburgh, Lawrence Erlbaum Associates, 16–23.

WILSON S.W. (1987), *Classifier Systems and the Animat Problem*, Machine Learning, 2, 199–228.

WILSON S.W. (1994), *ZCS: A Zeroth Level Classifier System*, Evolutionary Computation, 2(1), 1–18.

WILSON S.W. (1995), *Classifier Fitness Based on Accuracy*, Evolutionary Computation, 3(2), 147–175.

WILSON S.W. (1998), *Generalisation in the XCS classifier system*, w: Genetic Programming 1998, Proc. 3rd Annual Genetic Programming Conf. (GP98), red. J.R. Koza i in., Morgan Kaufmann, San Francisco, CA.

WILSON S.W. (2000), *Get real! XCS with Continuous-Valued Inputs*, w: Learning Classifier Systems. From Foundations to Applications, red. P.L. Lanzi i in., LNAI 1813, Springer, Berlin, Heidelberg, 209–219.

WILSON S.W. (2001), *Mining oblique data with XCS*, w: Advances in Learning Classifier Systems, red. P.L. Lanzi i in., LNAI 1996, Springer, Berlin, Heidelberg, 158–174.

WILSON S.W., GOLDBERG D.E. (1989), *A critical review of classifier systems*, w: Proc. 3rd Int. Conf. on Genetic Algorithms, red. J.D. Shaffer, Morgan Kaufmann, San Francisco, CA, 244–255.

WOLFF J.G. (2003), *Information Compression by Multiple Alignment, Unification and Search as a Unifying Principle in Computing and Cognition*, Artificial Intelligence Review, 19, 193–230.

WYARD P. (1991), *Context-Free Grammar Induction Using Genetic Algorithms*, w: Proc. 4th Int. Conf. on Genetic Algorithms, red. R.K. Belew, L.B. Booker, Morgan Kaufmann, San Diego, CA, 514–518.

WYARD P. (1994), *Representational Issues for Context-Free Grammar Induction Using Genetic Algorithm*, Proc. 2nd Int. Colloquium on Grammatical Inference and Applications, LNAI 862, Springer, London, 222–235.

WYATT D. (2004), *Applying the XCS Learning Classifier System to Continuous-Valued Data-Mining Problems*, Technical Report UWELCSG05-001, Learning Classifier Systems Group, University of the West England, Bristol.

YOKOMORI T. (1988a), *Learning simple languages in polynomial time*, Technical Report SIG-FAI, Japanese Society for AI.

YOKOMORI T. (1988b), *Inductive inference of context-free languages based on context-free expressions*, Int. J. Computer Math., 24, 115–140.

YOKOMORI T. (1991), *Polynomial-time learning of very simple grammars from positive data*, Proc. 4th COLT, 213–227.

YOUNGER D.H. (1967), *Recognition and parsing of context-free languages in time n^3*, Information and Control., 10(2), 189–208.

YOUNG-LAI M. (1996), *Application of a Stochastic Grammatical Inference Method to Text Structure*, MSc Thesis, Computer Science Department, University of Waterloo.

YOUNG-LAI M., TOMPA F.W. (2000), *Stochastic grammatical inference of text database structure*, Machine Learning, 40, 111–137.

ZHOU H. (1990), *CSM: A Computational Model of Cumulative Learning*, Machine Learning, 5(4), 383–406.

ZHOU H., GREFENSTETTE J.J. (1986), *Induction of finite automata by genetic algorithms*, Proc. 1986 IEEE Int. Conf. on Systems Man and Cybernetics, Atlanta.

ZOMORODIAN A. (1995), *Context-Free Language Induction by Evolution of Deterministic Push-Down Automata Using Genetic Programming*, w: Working Notes for the AAAI Symposium on Genetic Programming, red. E.V. Siegel, J.R. Koza, MIT Press, Cambridge, MA, USA, 127–133.

Skorowidz

Spis rysunków

Spis tabel

Evolutionary grammatical inference

The monograph takes up an important and prolific, both theoretically and practically, subject of grammatical inference (grammar induction). A new model of evolutionary grammatical inference has been proposed, and its main purpose is context-free grammar induction. The structure of the new evolutionary model utilizes a learning mechanism applied in learning classifier systems. Here the classifiers are productions of context-free grammar presented in the Chomsky normal form, and the environment to which the system adapts is a learning set composed of the exemplary sentences. These sentences are labeled to distinguish a collocation or lack thereof to the searched language. The purpose of learning is a correct classification of learning sentences. Since the set of classifiers constitutes a grammar production unit, the correct classification of labeled sentences denotes inducing a grammar. The model monitors the productions used during the learning set analysis and after the analysis calculates a fitness function of each production. New grammar productions are discovered during the induction by a covering and a genetic algorithm.

The thesis is divided into two parts. The first part introduces us into the area of grammatical inference, evolutionary processing, and learning classifier systems. In particular, the state of the art in the research in context-free grammar induction has been presented, a new categorization method of learning classifier systems has been proposed, and their generic models have been introduced in a uniform depiction.

The second part proposes an original model of evolutionary grammatical inference, dedicated to context-free grammar induction. The architecture and operation of the new model have been described with the use of the categories of a learning classifier system. So called “production fertility mechanism” has been introduced, which together with a crowding mechanism and inversion operator is supposed to counteract the high epistasis in production population. New covering operators adapted to the applied parsing method and estimators of induction accuracy and cost have been defined. The induction conducted includes regular languages from the Tomita set, chosen formal context-free languages, and large natural language corpora. These experiments show4ed that for each class of the language examined the model obtains results which are comparable with, and in some cases even better than, the best known in the literature of the subject, and not only among the evolutionary methods. Computer simulations have been conducted to experimentally identify qualities of the proposed evolutionary model. In addition to detailed conclusions, interesting results concerning general evolutionary mechanisms have been obtained, including an influence of tournament selection and of crowding on selective pressure, or the role of a new full covering operator in the evolution process of the population of a learning classifier system. Except for the area of linguistic engineering examined in this thesis, one of the potential applications of the model has been pointed to, which is computational genomics. The issues of an identification of human telomer sequence and of searching for a *E. coli* promoter region have also been investigated. The model in its current implementation can be applied at a high level of the specificity estimator to recognize regions not belonging to promoter sequences.

Spis treści

Contents