Halina Kwaśnicka

Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania
Politechnika Wrocławska, Wyb. Wyspiańskiego 27, 50-370 Wrocław
tel: (71) 3202397, fax: (71) 211018, E-mail: kwasnicka@ci.pwr.wroc.pl

ALGORYTMY GENETYCZNE W UCZENIU SIĘ MASZYN

1. Wstęp

W niniejszej pracy podjęto próbę pokazania możliwości wykorzystania algorytmów genetycznych do automatycznego uczenia się maszyn. Dziedzina automatycznego uczenia się maszyn jest intensywnie rozwijana w wielu ośrodkach naukowych na świecie, wiele aplikacji wykorzystuje algorytmy genetyczne.

Zaprogramowanie komputerów w taki sposób, aby mogły się one uczyć stanowiło wyzwanie dla wielu specjalistów z różnych dziedzin, nie tylko informatyki, ale również teorii poznania (ang. *cognitive science*), filozofii i in. W ramach *informatyki* wyodrębniła się nowa dziedzina: *automatyczne uczenie się maszyn* (ang. *Machine Learning*). Zajmuje się ona szukaniem odpowiedzi na pytanie: w jaki sposób konstruować programy komputerowe, które będą potrafiły automatycznie poprawiać swoje działanie w miarę zdobywania 'doświadczenia'? Satysfakcjonującej odpowiedzi na powyższe pytanie nadal nie ma.

W obecnym, skomputeryzowanym świecie istnieją olbrzymie bazy danych. Oczywiste jest, że są one swego rodzaju skarbnicą wiedzy. Kryją w sobie wiedzę na temat powiązań pomiędzy różnymi wielkościami, najczęściej trudną do wychwycenia przez człowieka. Pomocy komputerów w automatycznym pozyskiwaniu wiedzy z baz danych nie można przecenić. Zaprojektowanie odpowiedniego programu, satysfakcjonującego pod względem jakości i czasu działania jest nadal wyzwaniem dla informatyków.

Zakres zastosowań algorytmów genetycznych w szeroko rozumianym automatycznym uczeniu się jest tak duży, że niemożliwe wydaje się wykonanie pełnego przeglądu istniejących prac i autorka nie stawia sobie takiego celu.

Przedstawiono prace wybrane spośród dostępnych i, zdaniem autorki, bardziej interesujących. Starano się tak dobrać omawiane prace, by pokazać różne podejścia.

2. Automatyczne uczenie się maszyn i algorytmy genetyczne

Za [16] możemy przyjąć następującą definicję uczenia się maszyn:

> Mówimy, że program komputerowy jest zdolny do automatycznego uczenia się zadań z określonej klasy *T*, względem miary jego działania *P*, wykorzystując dany zbiór doświadczeń *D*, jeżeli wskutek kolejnych doświadczeń z *D* lepiej rozwiązuje zadania z klasy *T* względem miary *P*.

Przykład definicji zadania automatycznego uczenia się:
Klasa zadań *T*: gra w szachy; Miara działania *P*: procent wygranych gier; Doświadczenia uczące *D*: zestaw wykonanych gier z przeciwnikami.

Napisanie programu, który umożliwi komputerowi uczenie się rozwiązywania zadanego problemu jest zadaniem bardzo pożądanym, ale jednocześnie bardzo trudnym. Obecnie wydaje się, że nie ma programu, który umożliwiałby komputerowi uczenie się zbliżone do możliwości człowieka. Wśród wielu opracowanych systemów i metod uczących są też systemy z algorytmami genetycznymi. Kluczowy problem dla automatycznego uczenia się to indukcja ogólnych funkcji ze zbioru specyficznych przykładów. Wśród popularnych podejść można wyróżnić 16]:

☞ Metody polegające na uczeniu jawnego opisu funkcji docelowej (uczenie indukcyjne)

Przetwarzanie dostępnych danych (przykładów uczących) w celu znalezienia pewnych ogólnych, jawnych opisów funkcji, wykorzystywanych gdy zachodzi konieczność przetworzenia nowego zdarzenia (ang. *instance*).

- Uczenie się pojęć (ang. *Concept Learning*)

Pozyskanie definicji ogólnych kategorii mając do dyspozycji próbkę pozytywnych i negatywnych przykładów (przeszukiwanie przestrzeni predefiniowanych hipotez w celu znalezienia hipotez najbardziej ‘pasujących’ do przykładów uczących).

- Uczenie drzew decyzyjnych (ang. *Decision Tree Learning*)

Uczona funkcja jest reprezentowana jako drzewo decyzyjne, liście odpowiadają decyzjom (akcjom) a węzły atrybutom. Łuki wychodzące z węzła odpowiadają ograniczeniom na wartości atrybutu reprezentowanego przez ten węzeł.

- Uczenie Bayesowskie

Wiedza aprioryczna jest łączona z obserwowanymi danymi (przykładami uczącymi) do oszacowywania końcowego prawdopodobieństwa hipotez. Wiedza dostarczana systemowi może być w postaci prawdopodobieństw a priori dla każdej

kandydującej hipotezy lub w postaci rozkładów prawdopodobieństw. Konieczność dostarczenia systemowi wielu prawdopodobieństw jest utrudnieniem dla stosowania tej metody.

- Uczenie wykorzystujące sztuczne sieci neuronowe

Mając zadany ciąg przykładów uczących można nauczyć sieć neuronową generowania odpowiednich sygnałów wyjściowych dla zadanych sygnałów wejściowych.

☞ Metody polegające na zapamiętywaniu przykładów (Instance-Based Learning)

Wszystkie przykłady z ciągu uczącego są wprost zapamiętywane. Są one wykorzystywane dopiero wtedy, gdy system otrzyma zapytanie dotyczące nowego zdarzenia (ang. *instance*). Najprostsza metoda to proste zapamiętanie wszystkich przykładów i odszukiwanie odpowiedniego dla nowego zadania. W bardziej zaawansowanych metodach badane są powiązania aktualnego zapytania z zapamiętanymi przykładami w celu ustalenia odpowiedzi, np. wnioskowanie na bazie przykładów (ang. *Case-Based Reasoning*), *k*-najbliższych sąsiadów (*k-Nearest Neighbor*), czy lokalnie ważona regresja (*Locally Weighted Regression*).

☞ Uczenie analityczne

Stosuje się wiedzę aprioryczną, dotyczącą danej dziedziny oraz wnioskowanie dedukcyjne w celu powiększenia informacji dostarczonej przez przykłady uczące. Wiedza aprioryczna musi być systemowi dostarczona. Przykłady są uogólniane bardziej w oparciu o logiczne wnioskowanie niż o statystyczne, jak miało to miejsce w metodach indukcyjnych. Szuka się tu hipotezy najbardziej pasującej do apriorycznej wiedzy i obejmującej dane uczące. Bywa stosowane z sukcesem w planowaniu (ang. *planning*) i tworzeniu harmonogramów (ang. *scheduling*).

☞ Połączone uczenie indukcyjne i analityczne

Połączenie takie daje korzyści w postaci większej poprawności uogólniania przy dostępnej wiedzy apriorycznej oraz umiejętności wyszukania pewnych zależności w obserwowanych danych uczących dla wypracowania pewnej wiedzy apriorycznej.

☞ Uczenie wzmocnione (Reinforcement Learning)

Jest to uczenie autonomicznego agenta, odbierającego bodźce i działającego w określonym środowisku, wyboru najlepszych akcji aby osiągnął swój cel (uczenie sterowania autonomicznym robotem, optymalizacja sekwencji operacji w fabrykach, gry). W każdej jednostce czasu agent wykonuje akcję w swoim środowisku, trener może dostarczać wypłatę (nagrodę lub karę) aby wskazać pożądany lub docelowy stan. Zadaniem agenta jest nauczyć się wybrać sekwencję akcji dającą największą sumę wypłat.

W naturze istnieje wiele przykładów systemów uczących się. Człowiek od dawna próbował naśladować naturę, jedna z takich prób, to imitacja mózgu. Jej celem było uczynienie komputera „zdolnym" jak człowiek, prace związane z tą tematyką doprowadziły do rozwoju sztucznych sieci neuronowych. Natomiast w oparciu o analogie do procesów obserwowanych w ewolucji biologicznej powstała teoria *Algorytmów Genetecznych* (**GA** – *Genetics Algorithms*). Stosowane są one jako algorytmy przeszukujące (*Search*), optymalizujące (*Optimization*) i uczące się (*Learning*). Powstało wiele różnych podejść, jak strategie ewolucyjne, programowanie genetyczne i ewolucyjne. Podejścia te możemy nazywać algorytmami ewolucyjnymi. Algorytmy genetyczne, czy też szerzej, ewolucyjne, znajdują różne zastosowania, od komputerowego wspomagania projektowania (np. samolotów, układu gazociągu), poprzez systemy finansowe, kryminalistyka, do paradygmatu ekonomii ewolucyjnej [4,12].

Ewolucja populacji jest procesem przeszukiwania przestrzeni potencjalnych rozwiązań. W procesach takich jest ważne zachowanie równowagi pomiędzy przekazywaniem najlepszych cech do następnego pokolenia, czyli wykorzystaniem dotychczas znalezionych „obiecujących" rozwiązań, a szerokim przeszukiwaniem. Algorytm genetyczny umożliwia zachowanie takiej równowagi. Aby zastosować **GA** do rozwiązania danego problemu należy wykonać następujące czynności:

☞ Dokładne sprecyzowanie problemu (co jest optymalizowane, zakres zmienności, ograniczenia, dokładność reprezentacji argumentów funkcji celu, itp.).

W algorytmach genetycznych stosuje się zakodowane rozwiązania, zwykle, choć niekoniecznie, za pomocą binarnego łańcucha. Zakodowane wartości umieszczane są najczęściej na jednym łańcuchu, nazywanym *chromosomem*, stanowiącym pojedynczego *osobnika* w ewoluującej populacji. Wartość *funkcji celu* obliczona dla zdekodowanego chromosomu niesie informacje o *przystosowaniu* (*fitness*) danego osobnika, od przystosowania osobnika zależy liczba jego potomków.

☞ Zdefiniowanie problemu w terminach algorytmów genetycznych.

Definiowanie *chromosomów* (osobników), *funkcji przystosowania* (np. funkcja celu może być przekształcona aby zapewnić dodatnie wartości lub uwzględnić kary nakładane na niedopuszczalne rozwiązania), ustalenie parametrów **AG** (np. liczebność populacji *N*, prawdopodobieństwo mutacji, krzyżowania, itp.). Wykonanie tego kroku jest zwykle dość trudne.

☞ Teraz możliwe jest wykorzystanie programu komputerowego. Kolejne kroki, które wykonuje program to:

➪ Utworzenie początkowej populacji osobników. Populację początkową można wybrać losowo, jeśli brak jest jakichkolwiek przesłanek odnośnie rozwiązań.

➪ Ocena każdego osobnika w populacji (liczenie wartości funkcji

przystosowania).

- Sprawdzenie warunku zatrzymania programu (np. zadana liczba iteracji lub istnienie zadowalającego rozwiązania) – jeśli warunek zatrzymania nie jest spełniony, to muszą być wykonane następne kroki, jeśli zachodzi, to prezentowane jest najlepsze rozwiązanie (i ewentualnie inne, żądane statystyki) i następuje zatrzymanie programu.
- Tworzenie nowego pokolenia populacji:
  - ✓ Selekcja osobników do reprodukcji (zgodnie z założona metodą selekcji),
  - ✓ Reprodukcja osobników – tworzenie potomków z zastosowaniem operatorów krzyżowania i mutacji.
- Powrót do kroku „ocena każdego osobnika w populacji".

Populację początkową tworzy najczęściej zbiór *N* losowo wygenerowanych chromosomów. Populacja ta ewoluując znajduje nowe rozwiązania. Każdy osobnik w bieżącej populacji jest oceniany, następnie zachodzi proces reprodukcji. Można stosować różne metody selekcji osobników do reprodukcji, niemniej wszystkie one mają wspólną cechę: lepszy osobnik musi mieć większe szanse na posiadanie potomstwa, zgodnie z uproszczoną zasadą naturalnej selekcji '*przeżywa najlepszy*'. Reprodukowany osobnik może być krzyżowany z innym osobnikiem (*crosssover*). Powstały po krzyżowaniu potomek podlega *mutacji*. Mutacja polega na losowej zmianie wartości genu, np. na zamianie bitu z *zera* na *jedynkę* i odwrotnie. Po zakończeniu procesu reprodukcji powstaje *N* nowych osobników, tworzących następne pokolenie populacji – populację potomków. Ocena potomków kończy jeden cykl „życia" populacji. Cykle takie są powtarzane tak długo, dopóki satysfakcjonujące rozwiązanie nie zostanie znalezione.

3. Reprezentacja reguł z zastosowaniem GA

Algorytmy genetyczne znajdują coraz szersze zastosowanie w automatycznym generowaniu reguł decyzyjnych, w oparciu o zadany ciąg uczący. Oliver w 1994 roku zaproponował zastosowanie algorytmów genetycznych do optymalizacji sparametryzowanych reguł decyzyjnych [14]. W podejściu takim mamy sformułowaną ogólną regułę, z dokładnością do parametrów. Zadaniem **GA** jest dostrojenie parametrów reguły tak, by najbardziej odpowiadała ona ciągowi uczącemu. Zatem kolejność czynności jest następująca:

- Konstrukcja ogólnej, sparametryzowanej reguły (dokonuje się tego dysponując ciągiem uczącym w postaci przykładów sytuacji i podjętych w nich decyzji)

Załóżmy, że dysponujemy ciągiem n zestawów uczących, jeden zestaw uczący zawiera opis sytuacji za pomocą dwóch zmiennych v_1 i v_2 i podjętej decyzji d_i. Niech v_1 i v_2 przyjmują wartości liczbowe z zadanych przedziałów. Zestaw uczący będzie miał postać trójek: ($v_{1,i}$,v_{2,i_i} , d_i). Musimy zdecydować:

– jakie operatory zastosujemy do związania zmiennych z ich wartościami w przesłankach reguły, oraz
– jak podzielić dopuszczalne wartości zmiennych v_1 i v_2 na pewną liczbę przedziałów, oraz jakie ustalić granice tych przedziałów (oznaczmy je odpowiednio jako $v_{1,b1}$, $v_{1,b2}$, ..., $v_{2,b1}$, $v_{2,b2}$, ... gdzie $v_{j,bi}$ oznacza granicę i-tego przedziału j-tej zmiennej).

Przyjmijmy, że wybieramy operatory *większy-niż*, *mniejszy-niż*, *równy*, oznaczone odpowiednio przez o_1, o_2, o_3. Ogólna postać reguły jest zatem następująca:

$$\texttt{IF } v_1 \; o_S \; v_{1,bj} \texttt{ AND } v_2 \; o_P \; v_{2,bk} \texttt{ THEN } d=d_i \qquad (1)$$

☞ Następna czynność to zakodowanie tej reguły

Aby to wykonać należy zakodować wszystkie możliwe operatory i ustalone granice przedziałów dla obu zmiennych v_1 i v_2, oraz wszystkie możliwe decyzje. Reguła może być sekwencją:

`[operator,granica_`v_1`,operator,granica_`v_2`,decyzja]`, czyli [o_S, $v_{1,bj}$, o_P, $v_{2,bk}$, d_i],

którą następnie należy zakodować. Jeśli podejmowana decyzja jest prostą decyzją binarną (typu *tak* lub *nie*), możemy z ogólnej postaci reguły usunąć konkluzję i ograniczyć się tylko do zbioru reguł mających w konkluzji *tak*. Inaczej mówiąc, będziemy uczyć system reguł podejmujących decyzję *tak*. Jeśli dane nie będą pasować do wyuczonych w systemie reguł, wtedy odpowiedź systemu będzie *nie*. W tym prostym przykładzie, reguła składa się z czterech zakodowanych wartości: operator i granica dla obu zmiennych występujących w przesłankach reguły.

☞ Zastosowanie algorytmu genetycznego do generowania reguł

Funkcją przystosowania może być miara poprawności generowanej przez regułę konkluzji. Na zakodowanych regułach można wykonywać operatory genetyczne, np. krzyżowanie i mutację.

Przykład:

Załóżmy ciąg uczący postaci:

v_1	300	400	500	600	700	800	900
v_2	11	28	10	19	56	18	13
d	F	F	T	T	F	T	T

Przyjmijmy następujące operatory: o_1 oznacza ≥ oraz o_2 oznacza < .

Ustalamy granice przedziałów:

dla zmiennej v_1: $v_{1,b1} = 500$, $v_{1,b2} = 750$

dla zmiennej v_2: $v_{2,b1} = 10$, $v_{2,b2} = 20$, $v_{2,b3} = 30$

Jedna z reguł wygenerowanych przez algorytm genetyczny może mieć postać:

$$\textit{reguła\_i}\text{: IF } v_1 \geq 500 \text{ AND } v_2 < 20 \text{ THEN } d = \text{T} \tag{2}$$

Kodujemy nasze zadanie. Operatory za pomocą jednego bitu:

≥ jako *1* (jeden),
< jako *0* (*zero*);

Granice zmiennych za pomocą jednego i dwóch bitów (odpowiednio dla v_1 i v_2):

$v_{1,b1}$ = 500 jako *0*, $v_{1,b2}$ = 750 jako *1*,
$v_{2,b1}$ = 10 jako *01*, $v_{2,b2}$ = 20 jako *10*, $v_{2,b3}$ = 30 jako *11*.

Chromosom reprezentujący przykładową regułę *reguła_i* jest sekwencją bitów:

$$[1\ \ 0\ \ 0\ \ 10] \tag{3}$$

Każdy osobnik w populacji (chromosom) będzie ciągiem pięciu bitów, z których każdy ma swoje znaczenie. Pierwszy bit koduje operator dla związania pierwszej zmiennej, drugi bit – wartość (granicę przedziału) pierwszej zmiennej, trzeci bit to operator wiążący druga zmienną i dwa ostatnie bity to zakodowana wartość drugiej zmiennej. Początkową populację może stanowić zbiór losowo wybranych ciągów binarnych.

4. Genetyczne systemy uczące się (GBML)

W [4] D. Goldberg umieścił dość obszerne zestawienie wczesnych zastosowań genetycznych systemów uczących się. W niniejszej pracy omówione zostaną krótko trzy aplikacje, powstałe mniej więcej równolegle w różnych ośrodkach akademickich. Skrót *GBML* pochodzi od angielskiego określenia ***G**enetic **B**ased **M**achine **L**earning **S**ystems*. Jedne z pierwszych podejść do genetycznego uczenia się maszyn to systemy klasyfikujące, najczęściej cytowane są trzy podejścia: (1) Podejście Michigan (prace J. Hollanda), (2) Podejście Pitt (zainicjowane przez De Jong'a, nazwa pochodzi od miasta Pittsburg), (3) Indukcyjne uczenie wykorzystujące programowanie ewolucyjne (*GIL* – Genetic Inductive Learning, prace Janikov'a), za 15]).

4.1. Podejście Michigan

Najistotniejszą cechą algorytmu z Michigan jest to, iż osobnik reprezentuje pojedynczą regułę klasyfikującą. Populacja jest więc zbiorem reguł klasyfikujących. System działa w zamodelowanym środowisku, populacja reguł ewoluuje poprzez sporadyczne podawanie pobudzenia (bodźców) i wzmocnienia (wypłat) z otoczenia. Dzięki temu populacja ewoluuje a system „uczy się", które reguły (odpowiedzi) są odpowiednie dla zadanego pobudzenia. Każda reguła (osobnik) zakodowana jest w postaci łańcucha bitów. Klasyfikator ma następującą

postać:

$$(a_1, a_2, ..., a_n) : c_i, \quad (3)$$

gdzie a_i oznacza wartość *i*-tego atrybutu, natomiast c_i jest klasą.
Bodźcem przychodzącym z zewnątrz do systemu klasyfikującego jest komunikat wysyłany przez otoczenie. Taki komunikat może być dekodowany na kilka komunikatów wewnętrznych. Powodują one aktywację reguł klasyfikujących. Te reguły, które zostaną uaktywnione produkują odpowiedzi, które mogą spowodować uaktywnienie kolejnych reguł, lub stanowić komunikaty wyjściowe, wysyłane do otoczenia jako odpowiedź systemu. Otoczenie systemu dokonuje oceny odpowiedzi systemu, a zaimplementowany algorytm 'brygady porządkującej' (ang. *bucket brigade*) dokonuje uaktualnienia oceny reguł klasyfikujących zwiększając lub zmniejszając ich moc w zależności od tego, czy wysłana odpowiedź została przez otoczenie uznana jako poprawna. Moc reguł jest istotna przy licytacji reguł o prawo wysłania komunikatu (komunikaty silniejszych reguł są preferowane) oraz przy selekcji reguł do reprodukcji przez **GA**. W systemach takich **GA** działa po zadanej liczbie cykli (rozumianych jako podanie komunikatu wejściowego i wysłanie odpowiedzi systemu). W zadaniach klasyfikujących celem jest znalezienie pewnej liczby różnych reguł klasyfikujących, tak by pokryły one cały obszar znanych przykładów. Stanowi to trudność dla **GA**, które z definicji są zbieżne do jednego optimum, zatem wygenerowane reguły będą podobne do siebie. Aby temu zapobiec stosuje się specjalizowane metody selekcji, umożliwiające „zasiedlanie" różnych nisz ekologicznych, tzn. znajdowanie różnych optimów lokalnych. Jedną z takich metod jest metoda selekcji ze współczynnikiem ścisku, która zapewnia, że nowa reguła zastępuje najbardziej podobną do niej.

4.2. Podejście Pitt

Zasadnicza różnica pomiędzy tym podejściem, a omówionym wyżej tkwi w tym, że tu jeden osobnik (chromosom) reprezentuje nie pojedynczą regułę, a zestaw reguł. Ponieważ cały zestaw reguł klasyfikujących decyduje o sprawności klasyfikatora, zatem podejście, w którym ewoluują i konkurują między sobą zestawy reguł klasyfikujących wydaje się bardziej naturalne. Zestaw reguł może być oceniany biorąc pod uwagę poprawność klasyfikacji znanych przykładów. Odpada istotny w poprzednim podejściu problem heurystyk związanych z ustaleniem wypłat, mogą być też stosowane tradycyjne metody selekcji. Pojawia się problem z reprezentacją chromosomów, które w tym przypadku będą łańcuchami o zmiennej długości.

4.3. System *GIL* (Gentic Inductive Learning)

Pojedynczy chromosom (osobnik) reprezentuje tu opis pojęcia, którego system powinien się nauczyć. Każde zdarzenie pasujące do tego opisu jest pozytywnym przykładem uczonego pojęcia, nie pasujące jest negatywnym przykładem. Długość chromosomu w systemie *GIL* jest zmienna. Jest on zbiorem (dysjunkcją) pewnej liczby kompleksów (ich liczba może być różna na poszczególnych chromosomach), przy czym kompleks jest tu rozumiany jako koniunkcja pewnej liczby selektorów odpowiadających różnym atrybutom. Liczba selektorów w poszczególnych kompleksach też może być różna. Selektory (a zatem całe chromosomy również) są kodowane binarnie. Wartość jedynki na *i*-tej pozycji w selektorze oznacza, że w tym selektorze występuje *i*-ta wartość tego selektora. Każdy selektor ma tyle znaków binarnych, ile może mieć różnych wartości. Same jedynki w danym selektorze odpowiadają zakodowanej informacji „obojętne co". Najbardziej ogólna reguła składa się więc z samych jedynek.

W systemie zaimplementowano specjalizowane operatory genetyczne, działające na różnych poziomach.

Na poziomie chromosomu:

- *RuleExchange*, wymienia kompleksy pomiędzy dwoma chromosomami,
- *RuleCopy,* kopiuje losowy kompleksu z jednego chromosomu do drugiego,
- *NewPEvent*, dołącza opis pozytywnego przykładu do wybranego chromosomu,
- *NewNEvent*, dołącza opis negatywnego przykładu do wybranego chromosomu,
- *RuleGeneralizator,* uogólnia losowy podzbiór kompleksu, tzn. zamienia kilka kompleksów jednym, stanowiącym sumę logiczną tych usuwanych,
- *RuleDrop,* usuwa losowy podzbiór kompleksów z chromosomu,
- *RuleSpecjalization*, specjalizuje losowy podzbiór kompleksów (zastępuje wybrane kompleksy kompleksem będącym ich iloczynem logicznym).

Na poziomie kompleksu:

- *RuleSplit*, z jednego kompleksu tworzy kilka, każdy inną wartością wybranego selektora,
- *SelectorDrop,* w wybranym kompleksie ustawia wszystkie bity na 1,
- *IntroSelector*, selektor składający się z samych jedynek modyfikuje, zamieniając część z tych jedynek na zera.

Na poziomie selektora:

- *ReferenceChange*, dodaje lub usuwa jedną jedynkę w wybranym selektorze,
- *ReferenceExtension*, rozszerza dziedzinę selektora zezwalając na jego dodatkowe wartości,
- *ReferenceRestriction*, zawężenie dziedziny selektora.

Osobniki oceniane są ze względu na ich spójność i poprawność działania.

System *GIL* był porównywany z innymi systemami klasyfikującymi, w tym znanymi, nie wykorzystującymi algorytmów genetycznych, takimi jak C4.5 i AQ15. Wyniki wykazują, że jest to dość dobry system, lepszy od pozostałych.

4.4. Uczenie się strategii decyzyjnych w systemie SAMUEL

SAMUEL [5] jest systemem, w którym uczenie konkurencyjne jest stosowane do wygenerowania strategii dla agentów wyposażonych w podstawowe sensory, mogące dostarczać zaszumionych danych o środowisku. Agent podejmując decyzje ustala wartości zmiennych sterujących jego ruchem. System opisany jest za pomocą reguł. Osobnik reprezentuje całą strategię, to znaczy zestaw reguł. Konkluzje reguł dotyczą zmiennych sterujących. Każda reguła ma skojarzoną z nią *siłę* szacującą jej użyteczność w środowisku. Reguła o wysokich wypłatach ma dużą *siłę*. W SAMUEL'u reguły konkurują na dwóch poziomach: poziom strategii – każda strategia jest oceniana na podstawie wykonywania 20 zadań, za każde rozwiązane zadanie przyznawana jest wypłata. Na podstawie tej oceny lepsze strategie są wybierane do reprodukcji i krzyżowania. Na poziomie reguł – każda z reguł konkuruje z innymi o uaktywnienie się. Uwzględniana jest tu *siła* reguł, odzwierciedlająca historię jej własnych wypłat. Najsłabsze reguły mogą być usuwane ze strategii by zrobić miejsce na nowe reguły. Modyfikacja reguł obejmuje uogólnianie, specjalizację, łączenie i usuwanie. SAMUEL był testowany dla gry w *kotka-i-myszkę*: jeden agent miał nauczyć się łapać mysz, był wyposażony w sześć sensorów (w tym pozycję swoją i w określonym zasięgu myszy), drugi – modeluje mysz, porusza się w losowym kierunku i z losową prędkością, ma informację o aktualnej pozycji kota. System potrafił nauczyć się odpowiednich strategii. SAMUEL był wykorzystywany dla różnych zadań, w tym do przydzielania kredytu (ang. *credit assignment*), sterowania pojazdami. Zdaniem autorów, jego wyniki są zachęcające.

4.5. Tworzenie prototypów

Analizując sposób, w jaki człowiek dokonuje rozróżniania pojęć, dochodzimy do wniosku, że nie analizuje on wszystkich możliwych atrybutów, lecz korzysta z pewnych prototypów. Prototypy są wygodniejsze, pozwalają na krótszy opis klas. Niestety, nie wiadomo w jaki sposób człowiek tworzy prototypy, natomiast automatyczne tworzenie prototypów, które mogłyby być dalej wykorzystywane w systemach uczących się jest zadaniem bardzo trudnym. System PLEASE [8] wykorzystuje **GA** do tworzenia prototypów na podstawie wcześniej dokonanych klasyfikacji. Podobieństwo nowego przykładu określa się stosując tzw. *zasadę kontrastu*: jest to różnica pomiędzy ważoną sumą atrybutów równych ich prototypowym odpowiednikom, a ważoną sumą atrybutów różniących się od ich

prototypów. Dla każdej klasy system może tworzyć wiele prototypów. W ogólności prototypy mogą się różnić od widzianych wcześniej przykładów. Założono, że atrybuty mają wartości rzeczywiste i zrezygnowano z kodowania binarnego (co jest zgodne z obecną tendencją w aplikacjach z **GA**). Osobnik jest kompletnym opisem docelowego pojęcia. Każdy prototyp to zbiór wartości atrybutów, prototypy należące do tej samej klasy umieszczone są obok siebie na chromosomach, długość chromosomów jest zmienna. *k*-tego osobnika można przedstawić jako:

$$P^k_{1,1}P^k_{1,2}\ P^k_{2,1}P^k_{2,2}\|P^k_{3,1}P^k_{3,2}\ P^k_{4,1}P^k_{4,2}\ P^k_{5,1}P^k_{5,2}\ ,$$

gdzie $P^k_{n,j}$ oznacza wartość *j*-tego atrybutu w *n*-tym prototypie, *k* oznacza numer osobnika w populacji, symbol || oznacza granicę pomiędzy klasami. W powyższym przykładzie klasa pierwsza ma dwa prototypy, klasa druga trzy.

Mutacja zmienia wartość danego atrybutu na inną, wylosowaną z dziedziny atrybutu. Krzyżowanie musi zapewniać poprawność prototypu: u pierwszego rodzica losowany jest punkt krzyżowania. W strukturze drugiego osobnika dobiera się punkt krzyżowania zgodny semantycznie z wylosowanym u pierwszego (jeśli u pierwszego osobnika wylosowano punkt pomiędzy prototypami należącymi do *l*-tej klasy, to u drugiego również musi to być punkt pomiędzy prototypami w klasie *l*, podobnie, jeśli punkt wypadnie pomiędzy *i*-tym a *i+1*-szym atrybutem, to u drugiego też musi wystąpić pomiędzy tymi samymi atrybutami). Jako funkcję przystosowania przyjęto liczbę błędnych klasyfikacji. Uzyskane rezultaty, zdaniem autorów [8] pokazują, że system uczy się wolniej niż metodą najbliższego sąsiada, lecz odpowiedzi są szybsze ze względu na krótsze opisy klas. PLEASE jest wciąż rozwijany, planowane są prace nad dodaniem nowych operatorów oraz rozszerzenie reprezentacji atrybutów.

5. Genetyczne systemy uczące się w środowisku równoległym

Wiele praktycznych aplikacji z algorytmami genetycznymi wymaga długich obliczeń. Czas uzyskania wyników jest rzeczą ważną, dlatego też poszukuje się metod ich szybszego przetwarzania. Jedną z takich możliwości daje równoległe przetwarzanie. W rozdziale tym przedstawione zostaną dwa systemy. Jeden do indukcji pojęć z przykładów, z możliwością uczenia się dysjunkcji, przy czym pożądaną cechą jest, by system mógł uczyć się wszystkich dysjunkcji równocześnie, oraz by mógł znajdować dysjunktory nielicznie reprezentowane w przykładach pozytywnych. Drugi system, to hierarchiczny **GA** mający za zadanie znalezienie pewnych prawidłowości (reguł) ukrytych w bazach danych.

Historia równoległych algorytmów genetycznych (**PGA** – ang. *Parallel Genetic Algotithms*) nie jest zbyt długa, sięga początku lat osiemdziesiątych. Mimo, iż w literaturze można znaleźć omówienia wielu różnych podejść do **PGA**, to opisywane eksperymenty dotyczą różnych zadań, stosują różne operatory genetyczne, wykonywane są w różnych środowiskach, korzystają z różnej architektury, co powoduje, że jakiekolwiek porównanie ich staje się praktycznie niemożliwe. Pewien sposób usystematyzowania znanych podejść zaproponował E. Cantú-Paz [2]:

- ***równoległość globalna*** (ang. *global parallelization*)

Zarówno operatory genetyczne jak i liczenie przystosowania wykonywane są równolegle. Osobniki w całej populacji konkurują między sobą i mogą dowolnie krzyżować się między sobą (jak w szeregowym **GA**). Równoległość ta nie wymaga określonej architektury.

- ***gruboziarniste PGA*** (ang. *coarse grained*)

Gruboziarniste **PGA** oznacza, że czas przeznaczony na obliczenia jest dużo większy od czasu potrzebnego na komunikację pomiędzy procesami. Czasami ten rodzaj **PGA** nazywany jest *„rozproszonym **GA***", ponieważ najczęściej implementowany jest na maszynach MIMD z rozproszoną pamięcią. Cała populacja podzielona jest na kilka stosunkowo licznych podpopulacji, nazywanych w analogii do biologii, *demami*. Każdy dem ewoluuje prawie niezależnie od siebie, jednak od czasu do czasu, osobniki z jednego demu mogą przechodzić do innego i wnosić do niego swój materiał genetyczny. Należy zdefiniować dodatkowy operator, odpowiedzialny za wymianę materiału genetycznego pomiędzy demami. Najczęściej, jest to ***migracja***: z zadanym prawdopodobieństwem, wybrane osobniki są przesuwane z jednego demu do innego (*model wyspowy,* ang. *island model* – osobnik może migrować do dowolnego demu, oraz *model okrężny,* ang. *stepping stone model* – osobnik może migrować tylko do demu położonego w jego sąsiedztwie).

- ***drobnoziarniste zrównoleglanie*** (***ang. fine grained***)

Cała ewoluująca populacja podzielona jest na dużą liczbę stosunkowo mało licznych demów, najlepiej, kiedy dysponuje się oddzielnym procesorem dla każdego osobnika populacji. Wymaga to komputera o dużej liczbie procesorów (ang. *massively parallel computers*).

- ***zrównoleglanie mieszane*** (***ang. hybrid paralelization***)

Można stosować mieszane sposoby zrównoleglania algorytmów genetycznych.

5.1. REGAL – przykład równoległego, genetycznego systemu uczącego się

System REGAL (ang. **RE**LATIONAL **G**ENETIC **A**LGORITHM **L**EARNER) jest ogólnie

dostępny (można go znaleźć pod adresem: http://www.di.unito.it lub jako anonimowy użytkownik ftp: pianeta.di.unito.it, kartoteka pub/MLprog). Nadaje się do zadań indukcji pojęć na podstawie przykładów: dane są dwa zbiory, $E(h)$ i $C(h)$ odpowiednio pozytywnych i negatywnych przykładów h, należy znaleźć regułę, która obejmuje cały zbiór $E(h)$ i jednocześnie nie obejmuje zbioru $C(h)$ (przy zakłóceniach ostatni warunek może być luźniejszy) [17].

REGAL jest to system wykorzystujący **GA**, działający w środowisku rozproszonym, węzły połączone są w hiperkostkę. Jego język zawiera formuły w logice pierwszego rzędu, zbudowane z koniunkcji, dysjunkcji, negacji i XOR, zawiera kwantyfikator szczegółowy i ogólny.

W REGAL'u zaimplementowano pięć operatorów genetycznych: dwu-punktowe i pojedyncze krzyżowanie, zaprojektowane specjalnie w tym celu krzyżowanie specjalizujące i uogólniające, oraz operator zarodkowy (ang. *seeding operator*), pierwotnie stosowany do tworzenia populacji początkowej w sposób zapewniający rozpoczynanie pracy ze zbiorem formuł, z których każda obejmuje przynajmniej jedno pojęcie. **GA** ma włączony mechanizm *funkcji współudziału*[1] (*sharing functions*) [4], aby umożliwić tworzenie się (i utrzymywanie) demów skupionych wokół różnych optimów lokalnych. System umożliwia uczenie się alternatywnych pojęć, to jest reguł, w których przesłanki mogą być alternatywnymi opisami tych samych pojęć, obejmującymi różne przestrzenie przykładów. Reguła (opis pojęcia) może mieć zatem postać:

$$\varphi_1 \cup \varphi_2 \cup \ldots \cup \varphi_m \Rightarrow h \ , \qquad (3)$$

gdzie φ_i jest formułą w języku logiki pierwszego rzędu, opisującą pewną podprzestrzeń przykładów. Wszystkie obejmowane przez regułę (3) przykłady powinny należeć do podzbioru pozytywnych przykładów z ciągu uczącego, reguła nie powinna pokrywać żadnego negatywnego przykładu.

Generowanie takich reguł jest zadaniem dość trudnym, zwłaszcza, gdy niektóre składniki są nielicznie reprezentowane w ciągu uczącym. Twórcy takich algorytmów jak ID3, C4.5, AQ15 uważają, że można uczyć jednej alternatywy w danym momencie czasu. Wiadomo jednak, że **GA** stosując nisze[2] potrafi uczyć wszystkich alternatyw jednocześnie. Wykonano wiele prób z zastosowaniem selekcji metodą ścisku (ang. *crowding factor*) jak i funkcji współudziału. Wyniki, choć niewystarczające, są na tyle obiecujące, że prace te są nadal rozwijane.

W [17] zaproponowano nową postać funkcji współudziału, która w zamierzeniu

---

[1] Funkcja współudziału polega na tym, by osobniki zajmujące dany szczyt przystosowawczy dzieliły między siebie zysk przystosowawczy. Pozwala to na ukształtowanie się populacji na lokalnych szczytach z zachowaniem proporcji liczby osobników na poszczególnych szczytach do wysokości tych szczytów [4].

[2] „Nisza" jest pojęciem zaczerpniętym z biologii: nisza ekologiczna jest pozycją populacji danego gatunku w biocenozie. Problem nisz w **GA** polega na stosowaniu takich operatorów genetycznych i metod selekcji, które umożliwią utrzymywanie się osobników populacji na lokalnych optimach [4].

ma zmniejszyć złożoność obliczeniową operatora selekcji. Zaproponowany operator, nazwany *uniwersalnym głosowaniem* (ang. *universal suffrage*), ma za zadanie umożliwiać populacji zasiedlanie w stanie równowagi wszystkich szczytów lokalnych. Jest wtedy szansa, że lokalne optima, oznaczające formuły opisujące nielicznie reprezentowane przykłady zostaną znalezione. Nowa metoda selekcji w dużym stopniu uwzględnia to, ile przykładów pokrywa dana formuła. Załóżmy, że w populacji **P**(t) jest r formuł φ_i (i=1, ..., r) obejmujących dany przykład ξ. Przykładowi ξ można przypisać koło ruletki podzielone na r sektorów, z których każdy odpowiada formule φ_i i ma powierzchnię proporcjonalną do wartości p_i:

$$p_i = p(\varphi_i) = \frac{f(\varphi_i)}{\sum_{\varphi_k \in R(\xi)} f(\varphi_k)} \tag{4}$$

gdzie $f(\varphi_j)$ jest przystosowaniem reguły j, a $R(\xi)$ jest zbiorem wszystkich formuł obejmujących przykład ξ. W [17] stosowano następującą funkcję oceny formuł:

$$f(\varphi) = (1 + Az)e^{-\alpha w^{\beta}} + D \tag{5}$$

gdzie w i z są miarami spójności (*consistency*) i prostoty (simplicity) formuły φ, natomiast α, β, A i D są parametrami zdefiniowanymi przez użytkownika (mają wartości z przedziału [0,1], zwykle $D<<1$).
Selekcji osobników (formuł) dokonuje się w następujący sposób:

- Powtórz n razy (n – liczba formuł, które chcemy wybrać, musi być większe od rozmiaru populacji):
- Ze zbioru N pozytywnych przykładów wybierz losowo przykład (losowanie z powtórzeniem, rozkład równomierny),
- Zbuduj dla wybranego przykładu koło ruletki (z formułami, które go pokrywają),
- Uruchom koło ruletki,
- Wybierz do reprodukcji formułę odpowiadającą wylosowanemu sektorowi na kole.

Każdy pozytywny przykład może być wybrany więcej niż jeden raz, podobnie każda formuła może być wybrana do reprodukcji kilka razy. Załóżmy, że formuła φ obejmuje s przykładów. Wtedy występuje ona w s ruletkach, w każdej może wystąpić z innym prawdopodobieństwem. Zakładając, że istnieje N przykładów pozytywnych i należy wybrać n przykładów, prawdopodobieństwo wyboru do reprodukcji formuły φ wynosi:

$$P_{sel} = P\{\varphi \; jest \; wybrany \; do \; reprodukcji\} = f(\varphi) \cdot \frac{n}{N} \cdot \frac{1}{\sum_{\varphi_l \in R(\xi)} f(\varphi_l)} \tag{6}$$

Powyższy wzór pokazuje, że zaproponowana metoda jest równoważna klasycznej

selekcji. Udało się zredukować złożoność obliczeniową do rzędu O(M)+O($N \cdot M$), gdzie M – liczebność populacji, N – liczba pozytywnych przykładów. Wcześniejsze metody dawały złożoność obliczeniową rzędu O($N^2 \cdot M$).

System REGAL [17] testowano dla sztucznie zdefiniowanego zadania. Celem eksperymentów było sprawdzenie, czy zaproponowana metoda pozwoli nauczyć się nielicznie reprezentowanych dysjunktorów (gdy inne są licznie reprezentowane). Zastosowano 500 przykładów pozytywnych i 500 negatywnych, ich poprawny opis winien składać się z czterech dysjunktorów: φ_1 reprezentowany przez 312 przykładów, φ_2 przez 167, φ_3 przez 56 i φ_4 przez 35. W podanym ciągu uczącym istniał też inny dysjunktor w miejsce czwartego, φ_4', który nieźle pokrywał φ_4 i był reprezentowany przez 105 przykładów. System czasami znajdował to drugie rozwiązanie (z φ_4') zamiast zamierzonego pierwszego (z φ_4).

W REGAL'u zastosowano **PGA**, wszystkie węzły połączone są w hiperkostkę. Osobniki mogą migrować pomiędzy węzłami. Cechą charakterystyczną systemu jest to, że pozwala on na wybór pomiędzy dwiema skrajnymi sytuacjami: jednoczesne uczenie tylko jednego dysjunktora, a jednoczesne uczenie wszystkich dysjunktorów.

W tym celu zaimplementowana jest strategia długoterminowego dostrajania, nazywana 'Torysi-i-Wigowie' (ang. *Tories-and-Whigs*). Węzeł typu *Tory* zawsze czeka na powstanie stabilnej i dobrej populacji, obejmującej możliwie wszystkie pozytywne przykłady docelowego pojęcia. Natomiast węzeł typu *Whig* działa w inny sposób: jeśli przez zadaną liczbę generacji najlepszy osobnik na danym węźle nie zmieni się, to zakłada się, że węzeł znalazł częściowe rozwiązanie φ. To rozwiązanie jest zapamiętywane, po czym następuje ponowne zainicjowanie **GA** na tym węźle. Jednakże przykłady obejmowane przez φ są usuwane ze zbioru przykładów uczących. Generowana jest nowa populacja formuł, rozpoczyna się nowa ewolucja, ale dotyczy ona tylko przykładów nie objętych przez znalezione częściowe rozwiązanie. Pierwsze wyniki uzyskane dla sztucznych zadań są zachęcające, ale system wymaga dalszych prac, zwłaszcza nad nowymi strategiami ewolucyjnymi.

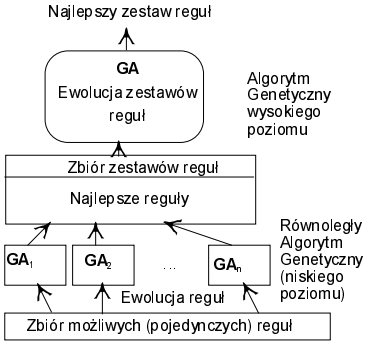
5.2. Pozyskiwanie wiedzy z baz danych: Hierarchiczny GA

Istniejące bazy danych zawierają mnóstwo danych. Analiza danych zawartych w dużej bazie wiedzy pozwoliłaby na sformułowanie pewnych regularności czy prawidłowości występujących pomiędzy zapamiętanymi danymi. Wykorzystanie pełnej informacji zawartej w tych bazach oraz sformułowanie ukrytej w nich wiedzy zwykle przekracza możliwości percepcji człowieka. Celem popularnej ostatnio dziedziny, nazywanej ***Data Mining*** (*pozyskiwanie wiedzy z baz danych* czy też *odkrywanie wiedzy*) jest opracowanie metod automatycznego,

indukcyjnego wnioskowania z istniejących baz danych. Inaczej mówiąc, chodzi o wykrywanie interesujących i użytecznych wzorców istniejących w bazie danych [18]. Najbardziej pożądaną postacią prezentacji odkrytej wiedzy są reguły *if x then y*, ponieważ są one najbliższe ludzkiej percepcji wiedzy, nadają się dobrze do stosowania w systemach ekspertowych.

W uczeniu się na podstawie przykładów indukowana reguła (hipoteza) była na ogół dobra, jeśli pokrywała możliwie dużą część przykładów. W *odkrywaniu wiedzy* należy wygenerować reguły, które będą użyteczne, tzn. będą one wskazywać pewne ogólne trendy i korelacje, ale nie muszą być prawdziwe dla wszystkich danych zawartych w bazie. Gdybyśmy generowali reguły prawdziwe dla jak największej części bazy, otrzymywalibyśmy regułę ogólną, ale nieprzydatną (np. cena wszystkich produktów jest dodatnia).

Odkrywanie reguł można sformułować jako proces przeszukiwania (ang. *search*) zatem można pokusić się o stosowanie **GA** do rozwiązywania tych problemów. Dużym problemem jest tu sformułowanie funkcji przystosowania, oceniającej pojedyncze reguły lub też zestawy reguł. Ciekawe podejście, polegające na łączeniu generowania pojedynczych „dobrych" reguł z generowaniem „dobrych" zestawów reguł omówione jest w pracy [18]. Zastosowany jest tu hierarchiczny **GA**, pokazany na Rysunku 2. Składa się on z dwóch poziomów. Na poziomie niższym uruchamiany jest równoległy algorytm genetyczny (**PGA**), który ma za zadanie wybrać „dobre" pojedyncze reguły, spośród wszystkich możliwych. Dopiero z nich tworzone są zestawy reguł, które ewoluują na poziomie wyższym. Poziom niższy jest zrealizowany jako drobnoziarnisty **PGA**, zatem duża liczba stosunkowo mało licznych demów ewoluuje równolegle. Dla **GA** wyższego poziomu założono stałą liczbę pojedynczych reguł w jednym zestawie.



Rysunek 1. Schemat działania hierarchicznego **GA**

Przykład [18]:

Niech baza danych zawiera tygodniowe dane o sprzedaży 156 produktów, w ciągu 55 tygodni, zawierające takie informacje jak cena, wielkość sprzedaży, dostawca, promocja, itp., oraz informacje o pogodzie (średnia tygodniowa wilgotność, minimalna i maksymalna temperatura, liczba słonecznych godzin). Każde pole w bazie może być postrzegane jako funkcja produktu i czasu, z wyjątkiem danych o pogodzie – te są funkcją tylko czasu. Wartości pól mogą być liczbowe (cena) lub typu wyliczeniowego (promocja). Wartości liczbowe są normalizowane tak, by ich wartość średnia wynosiła zero a wariancja jeden. Pola, których wartości

mogą być w zasadzie funkcją innych pól są polami zależnymi (np. ogólna wielkość sprzedanego produktu). W przykładzie przyjęto, że reguła będzie postaci *if ... then ...* , i tworzą ją trzy części:

Specyfikacja (ang. *specificity*) (*S*) – informuje, czego (jakiego produktu) dotyczy dana reguła, jest to przedmiot odniesienia,

Przesłanka reguły (ang. *conditional clauses*) (W_1, W_2, ..., W_n) – są to klauzule warunków, zakładamy, że mogą być połączone operatorem logicznym AND,

Konkluzja reguły (ang. *predictive clause*) (*K*) – klauzula konkluzji.

Ogólna postać reguły to:

$$S \text{ IF } W_1 \text{ AND } W_2 \text{ AND } \ldots \text{ AND } W_n \text{ THEN K} \tag{7}$$

Specyfikacja *S* może odnosić się do jednego produktu, bądź do wszystkich produktów; w konkretnym tygodniu, lub w przedziale czasu, itp. (np. dla produktu *x* w przedziale czasu [40÷50] zachodzi ...). Klauzule warunków W_i odnoszą się do pól danych i ich znormalizowanych wartości, z uwzględnieniem czasu, np.

$$\text{cena(jabłka,dzisiaj)} < 1{,}2 \cdot \text{cena(pomarańcze,ostatni\_tydzień)}$$

Konkluzja *K* jest pojedynczą klauzulą o takiej samej postaci jak klauzule warunków, ale musi spełniać jeden warunek: jedno z pól w klauzuli musi być kategorią zależną w bazie danych.

☞ Algorytm genetyczny niższego poziomu

Pozostaje problem z oceną przystosowania reguły. Można powiedzieć, że idealna reguła powinna być interesująca, stosowalna, dokładna, itp. [18].

Niech *dokładność reguły* α (*accuracy*) będzie zdefiniowana jako:

$$\alpha = \frac{|C \cap P|}{|C|} \tag{8}$$

gdzie *C* oznacza te punkty w bazie, dla których warunki reguły są prawdziwe, *P* zbiór punktów w bazie, dla których konkluzja jest prawdziwa.

Natomiast *zasięg reguły* γ (*coverage*) zdefiniowany jest następująco:

$$\gamma = \frac{|C \cap P|}{|P|} \tag{9}$$

Dla oceny reguły ważny powinien być jej zasięg i dokładność, ale również reguła będzie tym lepsza, im większe będzie miała statystyczne znaczenie[3]. Miarą statystycznego znaczenia reguły może być jej poprawność (*correction*): $a = |P|/|D|$, gdzie *D* jest dziedziną reguły ($|D|$ jest liczbą wszystkich konkluzji). Współczynnik ten mówi, na ile bardziej prawdopodobne jest, że konkluzja reguły jest prawdziwa, kiedy warunki są prawdziwe. Ogólnie, reguły stosujące się do większej części bazy

[3] Każda reguła, która jest prawdziwa dla całej bazy danych ma $\alpha = \gamma = 1$, ale zazwyczaj jest ona nieużyteczna.

danych są użyteczne, ale często bardziej interesujące mogą okazać się reguły, które stosują się do poszczególnego produktu. Zwłaszcza będzie tak w odniesieniu do reguł predykcyjnych. Zatem funkcja przystosowania powinna premiować reguły zarówno ze wzrostem $|C \cap P|$ jak i $|C' \cap P'|$, przy czym C' jest zbiorem punktów nie należących do C, P' – zbiorem punktów nie należących do P. Przystosowanie pojedynczej reguły zdefiniowane jest jako kombinacja miary jej jakości i ogólności. Stosowana w pracy [18] funkcja jest sparametryzowana, co pozwala na uwzględnianie w różnym stopniu dwóch czynników. Najlepszy model, jaki udało się autorom znaleźć to:

$$f = (\log(1+|C \cap P|) + \log(1+|C' \cap P'|))\left(\alpha - \frac{|C' \cap P'|}{|C'|}\right) \tag{10}$$

W **GA** niskiego poziomu stosowane są następujące operatory genetyczne:

Generowanie reguł: Losowany jest przedmiot odniesienia (*specificity*), pewna liczba klauzul warunków oraz klauzula konkluzji (wymuszane jest, by konkluzja zawierała jedno pole zależne). Operatory są (parametrycznie) ukierunkowane na pewne specjalne wartości. I tak, przedmiot odniesienia "*for all time*" jest losowany z zadanym prawdopodobieństwem, i tylko w przypadku nie wylosowania go, będzie generowany losowy przedział czasu. Podobnie, najbardziej prawdopodobne wartości stałych mnożenia i czasu opóźnienia w klauzulach wynoszą zero.

Mutacja: jest ukierunkowana, by częściej zmieniała składniki reguły na pewne specjalne wartości. Dla zmiany wartości numerycznych (nie ma kodowania binarnego) stosowana jest mutacja pełzająca (ang. *creep*). Zmiana wartości wyliczeniowych następuje poprzez zastąpienie starej wartości wylosowaną nową. Mutacja przedmiotu odniesienia (*specificity*) reguły obejmuje zmianę przedziału czasu lub wyspecyfikowanego produktu. Mutacja klauzul to zmiana pola danych, do których się odnosi dana klauzula, zmiana nierówności, itp. Dodatkowy typ mutacji obejmuje usunięcie oraz dodanie losowo wybranej klauzuli warunku. Powstała reguła jest analizowana i wszystkie podwójne klauzule są z niej usuwane.

Crossover: operator krzyżowania produkuje pojedynczego potomka z dwóch rodziców. Stosuje się ukierunkowane jego działanie tak, aby preferowany był wybór materiału genetycznego od jednego rodzica. Przedmiot odniesienia jest brany od jednego partnera, każda klauzula warunku rodziców jest rozpatrywana w celu dodania jej do potomka. Konkluzja brana jest od jednego z rodziców lub jest łączeniem klauzul konkluzji rodziców. Podwójne klauzule są usuwane z reguł.

Reprodukcja: winna zapewnić znalezienie lokalnych optimów. Zastosowano drobnoziarnistą strukturę populacji. Osobniki do reprodukcji i krzyżowania

wybierane są metodą turniejową[4] (binarną). Potomek jest mutowany i oceniany, po czym zastępuje rodzica jeśli jest lepszy. Po zadanej liczbie generacji, najlepsza reguła jest zachowywana dla **GA** wyższego poziomu i proces jest powtarzany.

☞ Algorytm genetyczny wyższego poziomu

Jest to etap generowania zestawu reguł. Postać reguły jest ta sama co na niskim poziomie, ale należy zdefiniować nową funkcję przystosowania (odpowiednią dla zestawu reguł, choć, nadal jakość pojedynczej reguły jest bardzo ważna). Przyjęta w omawianej pracy funkcja w zamierzeniu ma łączyć w sobie jakość pojedynczych reguł i jakość całych zestawów reguł. W tym celu zdefiniowano różnicę pomiędzy regułami (wynosi 1 dla reguł całkowicie innych i 0 dla reguł identycznych). Zestaw reguł jest oceniany poprzez mnożenie przystosowania każdej reguły z tego zestawu przez sumę różnic tej reguły z każdą inną regułą w tym zestawie.

Operatory genetyczne

Populacja początkowa jest generowana losowo. Pozostałe operatory to:

Mutacja – zrealizowana jako wybór wg rozkładu dwumianowego pewnej liczby reguł i zastąpienie ich losowo wybranymi regułami,

Krzyżowanie – kombinacja reguł z dwóch zbiorów rodzicielskich,

Selekcja – metoda turniejowa.

Ewolucja trwa zadaną liczbę iteracji. W omawianym przykładzie, zadaniem **GA** na tym poziomie było znalezienie dobrego podzbioru pięciu reguł spośród podstawowego zbioru 200 reguł. Przykład wyprodukowanej reguły to:

> Dla jabłek złote delicje, w ciągu ostatnich 44 tygodni, jeśli średnie nasłonecznienie (w godzinach) plus 5,4 razy cena detaliczna, jest większe niż 9,5, to ogólna wartość sprzedaży jabłek jest większa od 632 funtów.

W przedstawionym podejściu, poziom wyższy **GA** nie był w sprzężony z poziomem niższym. Wydaje się, że przekazywanie informacji zwrotnej z poziomu wysokiego do poziomu niższego mogłoby poprawić funkcjonowanie systemu.

6. GA w zadaniu doboru odpowiedniego zestawu cech

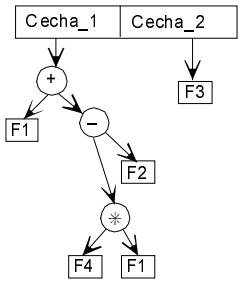
Niezależnie od wybranej metody automatycznego uczenia się (NN, ID3, itp.), przygotowanie odpowiedniego zestawu cech jest zadaniem istotnym. System powinien wykorzystywać możliwie najmniejszą liczbę cech, pozwalającą na poprawne uczenie się. Uwzględnianie cech, które nie mają własności rozróżniających, wydłuża proces obliczeniowy nie zwiększając poprawności uczenia. Z drugiej strony, dążąc do minimalizacji liczby cech, nie można

[4] W metodzie turniejowej binarnej, z dwóch wylosowanych osobników wybierany jest ten, którego wartość przystosowawcza jest lepsza.

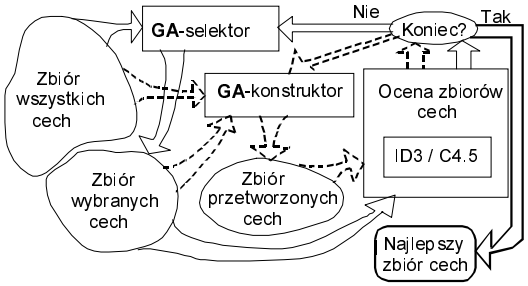
zrezygnować z cech istotnych dla właściwego rozpoznania.

Do wyboru odpowiedniego zestawu cech można z powodzeniem stosować algorytm genetyczny. **GA** może zarówno przeprowadzić selekcję cech jak i skonstruować nowe cechy, będące kombinacją tych wybranych, co w efekcie jeszcze zmniejsza liczbę koniecznych cech do analizowania przez system uczący się. W [18] można znaleźć przykład zastosowania **GA** do przygotowania danych dla takich algorytmów jak ID3 czy C4.5. Dane wejściowe dla systemu stanowi zbiór możliwych cech. Zaimplementowano oddzielnie dwa niezależne moduły stosujące **GA**, jeden, nazwijmy go **GA**–*selektor* ma za zadanie wybrać te cechy, które są istotne z punktu widzenia zadania rozpoznawania, drugi to **GA**–*konstruktor*, jego zadanie to stworzenie nowych cech będących kombinacją cech wejściowych. Na wejście **GA**–*konstruktora*, można podać zarówno surowy zestaw cech (będzie działał sam, bez **GA**–*selektora*) jak i ten wybrany przez **GA**–*selektor* (gdy włączone są oba moduły). Moduły te stosują inną reprezentację chromosomów. **GA**–*selektor* operuje na osobnikach będących podzbiorem możliwych cech. Znana jest liczba wszystkich cech, zatem efektywne będzie kodowanie osobnika na tylu bitach, ile jest cech. Jedynka na danej pozycji oznacza, że dana cecha jest włączona do zestawu wybranych cech, zero – eliminację cechy. Proste operatory mutacji i krzyżowania mogą być tu łatwo stosowane. **GA**–*konstruktor* musi wykorzystywać inną reprezentację. Nowe cechy są tworzone jako kombinacja istniejących. Cechy mające wartości rzeczywiste można łączyć poprzez stosowanie podstawowych operatorów arytmetycznych. Chromosom ma zmienną długość. Naturalną reprezentacją dla takich zadań jest struktura drzewiasta (Rysunek 2).

Zastosowano tu krzyżowanie jak w tzw. *programowaniu genetycznym*, zaproponowanym przez Kozę [10], oraz – dla zachowania koniecznej różnorodności – mutację, jako losową zmianę operatora lub cechy na inny operator lub cechę z dopuszczalnego zbioru. Oba moduły wykorzystują tę samą procedurę oceny przystosowania osobników. Po wybraniu podzbioru cech następuje modyfikacja danych treningowych poprzez usuwanie i/lub dodanie z przykładów wartości usuniętych i/lub skonstruowanych atrybutów. Następnie uruchamiany jest algorytm ID3 (lub C4.5) do tworzenia drzewa decyzyjnego w oparciu o zmodyfikowane przykłady uczące. Utworzone drzewo jest oceniane biorąc pod uwagę jego zdolności klasyfikujące na danych testowych (Rysunek 3).



Rysunek 2. Reprezentacja osobnika (składa się z 2 cech: {(F1+(F4∗F1–F2)); F3}



Rysunek 3. Schemat systemu uczącego się z wstępnym doborem cech przez **GA**

Algorytm testowano dla zadania rozpoznawaniu obrazów: wybrano losowo 200 wektorów po 8 cech z wybranego obszaru obrazka (30x30 pikseli), które podzielono na równe części, treningową i testową (67% danych wykorzystano do indukcji drzew, 33% do wyboru podzbioru cech). Stosowano najpierw moduł selekcji, później moduł konstrukcji. **GA**-*selektor* zredukował liczbę cech do 4 (o 50%), **GA**-*konstruktor* dalej zredukował do 3 cech: dwie pojedyncze cechy oraz trzecia będąca kombinacją wszystkich czterech. Interesujące jest, że wydajność rozpoznawania poprawiła się w stosunku do uzyskiwanej bez wstępnego przetworzenia cech.

7. Uwagi końcowe

Niniejsza praca miała pokazać użyteczność algorytmów genetycznych w szeroko rozumianym automatycznym uczeniu się maszyn. Uzyskiwane dotychczas wyniki nie są rewelacyjne, niemniej często przewyższają te, uzyskiwane technikami bez **GA**. Widoczne jest to zwłaszcza w systemach kombinowanych, gdzie **GA** są wykorzystywane na etapie wstępnego przetwarzania danych lub dopasowywania znajdowanych rozwiązań.

Interesujące zastosowania **GA** do szeroko rozumianego automatycznego uczenia się, to dostrajanie lub projektowanie całych systemów inteligentnych. Dotyczy to projektowania systemów opartych na sieciach neuronowych, logice rozmytej czy też popularnych gier menedżerskich, w których zadaniem **GA** jest szukanie optymalnej strategii [9, 11, 13]. Będzie to przedmiotem odrębnego opracowania.

LITERATURA

[1] *Building Classification Models: ID3 and C4.5* in http://yoda.cis.temple.edu:8080/UGAIWWW/lectures95/ learn/C45/.

[2] Cantú-Paz E., *A Survey of Parallel Genetic Algorithms*, IlliGAL Report No. 97003, May 1997. (http://GAL4.GE.UIUC.EDU).

[3] Davis L., *Handbook of Genetic Algorithm*, Van Nostrand Reinhold, New York.

[4] Goldberg D.E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Inc., 1989.

[5] Grefenstette J.J., *Learning Decision Strategies with Genetic Algorithms*, Proceedings of the Workshop on Analogical and Inductive Inference, Lecture Notes in Artificial Intelligence 642, Springer Verlag, 35-50, http://www.aic.nrl.navy.mil/ ~gref/papers.html.

[6] Heider H., Drabe T., *Fuzzy System Design with a cascaded Genetic Algorithm*, IEEE International Conference on Evolutionary Computation (IEC-97), 1997.

[7] Herrera F., Lozano M., *Generating Fuzzy Rules From Examples Using Genetic Algorithms*, praca dostępna w sieci http://decsai.ugr.es/~lozano/public.html.

[8] Knight L., Sen S., *PLEASE: A Prototype Learning System Using Genetic Algorithms*, Proceedings of the Sixth International Conference on Genetic Algorithms (pp. 429-435), Pittsburgh, Pennsylvania USA, 1995, (http://euler.mcs.utulsa.edu /~sandip/PLEASE.ps).

[9] Kot W., *Konkurencja firm na rynku – gra symulacyjna*, Praca magisterska, WZI, Wydział Informatyki i Zarządzania, Politechnika Wrocławska, Wrocław, 1997.

[10] Koza J.,R., *Genetic Programming*, Cambridge, MA: MIT Press, 1992.

[11] Kozieja R., *Zastosowanie Algorytmów genetycznych w modelowaniu ekonomicznym. Poszukiwanie optymalnej strategii gry kierowniczej*, Praca magisterska, WZI, Wydział Informatyki i Zarządzania, Politechnika Wrocławska, Wrocław, 1997.

[12] Kwaśnicka H. *Genetic and Evolutionary Algorithms – an Overview*, praca zgłoszona do: *INFORMATICA – An International Journal of Computing and Informatics*, seria 'An overview paper'., 1997.

[13] Kwaśnicka H., *Evolutionary Approach to Artificial Neural Network Design – Good Way or Blind Alley*, MENDEL'97 *III International Mendel Conference on Genetic, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets*. Technical University of Brno, Brno, Czech Republic,str. 342-347, June 25-27, 1997.

[14] Medsker L.,R., *Hybrid Intelligent Systems*, Kluwer Akademic Publishers, Boston/Dordrecht/London, 1995.

[15] Michalewicz Z., *Genetic Algorithms + Data structure = Evolution Programs*, Springers Series Artificial Intelligence., 1992.

[16] Mitchell T., M., *Machine Learning*, McGraw-Hill Companies, Inc., 1997.

[17] Neri F., Zini F. ,*Learning Disjunctive Concepts with Distributed Genetic Algorithms*, Dipartamento di Informatica, Univ. di Torino, Italy http://www.dai.ed.ac.uk/groups/evalg/eag_local_copies_of_papers.body. html.

[18] Radcliffe N.J., Surry P.D., *Co-operation through Hierarchical Competition in Genetic Data Mining*, Edinburg Parallel Computing Centre, Scotland. www.dai.ed.ac.uk/groups/evalg/eag_local_copies_of_papers.body.html

[19] Vafaie H., De Jong K., *Genetic Algorrithms as a Tool for Restructuring Feature Space Representations*, Computer Science Department, George Mason University, 1995.www.cs.gmu.edu:80/research/gag/pubs.html.