Politechnika Wrocławska

Wydziałowy Zakład Informatyki

Wydział Informatyki i Zarządzania

Praca Magisterska

Projektowanie struktury sieci neuronowej za pomocą wyspowego równoległego algorytmu genetycznego

Michał Owsiak

Promotor: dr Urszula Markowska-Kaczmar
Ocena:

Wrocław, 2001

Streszczenie

W niniejszej pracy przedstawiona została metoda projektowania struktury sieci neuronowej za pomocą wyspowego równoległego algorytmu genetycznego. W trakcie pracy określono optymalną architektórę aplikacji równoległej, która następnie została zaimplementowana. Posłużyła ona do późniejszych testów numerycznych na podstawie których został oceniony zarówno model teoretyczny jak i sama implementacja. W trakcie obliczeń poszukiwane były optymalne struktury sieci neuronowej dla problemów rozpoznawania irysów, rozpoznawania parzystości liczb zapisanych binarnie, konwersji liczb binarnych na skalę termoetrową oraz realizacji funkcji XOR. Testy zostały przeprowadzone w heterogenicznym środowisku równoległym.

Spis treści

1 Wstęp

Sieci neuronowe są niewątpliwie mocnym narzędziem obliczeniowym, jeżeli chcemy dokonywać klasyfikacji obiektów na podstawie ich cech. Współczesne algorytmy uczenia sieci neuronowych pozwalają na sprawne trenowanie struktur neuronowych, tak aby rozwiązywały zadany problem. Oczywiście musimy mieć świadomość, że rozwiązania wszystkich możliwych problemów nie możemy upatrywać tylko i wyłącznie w samym algorytmie. Znakomitą większość czasu potrzebnego na znalezienie optymalnego rozwiązania zajmuje poprawny dobór wzorców uczących oraz znalezienie najlepszej z możliwych struktury sieci neuronowej (często traktowane jako element mało istotny). Opracowanie wzorców uczących zwykle nie nastręcza problemu natury obliczeniowej. Zbiór danych przeznaczonych do trenowania sieci przygotowywany jest przez eksperta, który dokonuje wstępnej klasyfikacji. Ten sam proces obejmuje oczywiście wzorce testujące (pozwalające na ocenę ogólności rozwiązań sieci). Odrębnym problemem jest dobór struktury sieci tak, aby była strukturą optymalną. Niestety nie dysponujemy jeszcze prostą, a zarazem szybką metodą oceny takich struktur. Nie jest to proces łatwy. Na strukturę sieci składają się neurony (jednostki aktywacji), połączenia wiążące je w całość oraz przypisane im wagi. Dodatkowo ważnym jest, by sieć była w ogóle w stanie rozwiązać zadany problem. Jak można się domyślać przestrzeń poszukiwań jest ogromna.

Jak już wspomniano struktura sieci neuronowej jest zwykle traktowana po macoszemu, dlatego jej dobór najczęściej sprowadza się do przyjęcia, struktury sieci warstwowej o połączeniach pełnych pomiędzy warstwami. Jest to oczywiście rozwiązanie najprostsze. Jednak najczęściej prowadzące do nadmiarowości zarówno w liczbie neuronów, jak i połączeń między nimi. Jeżeli takie rozwiązanie nie jest dla nas zadowalające i zdecydujemy się na poszukiwanie optymalnej struktury, to najczęściej stosowaną metodą jest tak zwana metoda „pruningu". W krótkim zarysie polega ona na uczeniu zadanej struktury sieci neuronowej, a następnie usunięciu z niej części połączeń lub neuronów. W ten sposób otrzymana struktura jest ponownie oceniana. Jeżeli cechuje się lepszymi parametrami, to poszukiwania są kontynuowane, a za bazową przyjmowana jest mniejsza sieć. Może jednak stać się tak, że uzyskany efekt daje gorsze rezultaty. Wtedy algorytm musi postąpić krok w tył i rozpocząć „ucinanie" neuronów w innym miejscu. Należy zwrócić uwagę na fakt, że proces ten jest niczym innym jak znajdowaniem maksimum pewnej funkcji celu, określającej jakość sieci neuronowej. Dodatkowo algorytm ten zaliczyć należy do tak zwanych metod „brute force". Ciekawszym, do rozwiązania tego zadania, wydaje się być zastosowanie algorytmu genetycznego. Powszechnie uważa się, że algorytmy genetyczne są znakomitym narzędziem do poszukiwania optimum funkcji w sytuacji, gdy przestrzeń poszukiwań jest znacząca. Ich zaletą jest fakt, że poszukiwania przebiegają niejako w kilku miejscach przestrzeni jednocześnie. Dodatkowym atutem algorytmów genetycznych jest ich równoległość działania, co pozwala na łatwe i zarazem naturalne dokonanie modyfikacji w algorytmie tak, aby działał na maszynach równoległych.

Celem niniejszej pracy było stwierdzenie, czy wyspowe równoległe algorytmy genetyczne mogą znaleźć zastosowanie w procesie projektowania struktury sie neuronowej. Dodatkowo, ocenie podlegał wydajność takiego algorytmu poszukiwania struktury sieci neuronowej. Jak

już zostało wspomniane jest to proces szalenie kosztowny obliczeniowo. Znalezienie metody szybkiego i niezawodnego projektowania sieci neuronowej byłoby zatem znacznym krokiem naprzód.

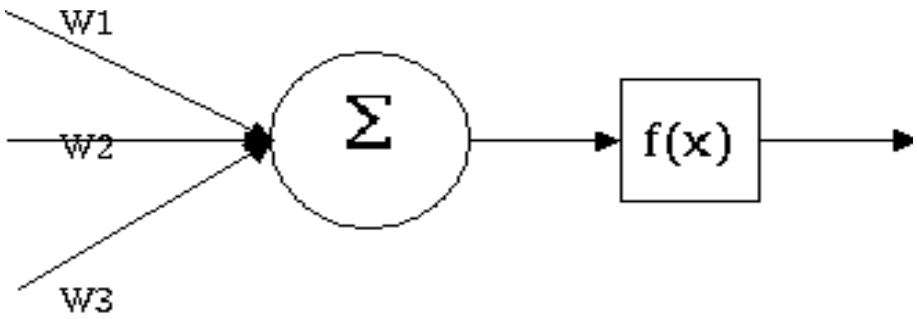
W kolejnych rozdziałach pracy zostały przedstawione elementy składające się na całość problemu. Rozdział drugi traktuje o sieciach neuronowych, podstawowych strukturach sieci oraz sposobie ich uczenia. W rozdziale trzecim zostały zaprezentowane podstawowe pojęcia związane z przetwarzaniem wykorzystującym algorytmy genetyczne. Rozdział ten traktuje o algorytmach genetycznych jako takich. Rozdział czwarty zawiera opis metod pozwalających na stosowanie algorytmów genetycznych do reprezentacji truktur sieci neuronowej. Zawarte w tym rozdziale informacje stanowią przegląd stosowanych powszechnie metod. Rozdział piąty traktuje o metodach zrównoleglania algorytmów genetycznych. W rozdziale tym zostały opisane trzy najczęściej stosowane metody zrównoleglania algorytmów genetycznych. W rozdziale szóstym przedstawiony został opis aplikacji, a dokładniej jej modelu teoretycznego. W rozdziale tym przedstawiono rozwiązania na, które zdecydował się autor pracy. W rozdziale tym zawarty został również model teoretyczny zrównoleglenia aplikacji. Rozdział siódmy traktuje o testach przeprowadzonych z wykorzystaniem wcześniej przedstawionej aplikacji.Testy opierają się o podstawowe problemy klasyfikacji. W rozdziale ósmym znajduje się krótkie podsumowanie wyników uzyskanych w testach. Dokładniejszy opis aplikacji można znaleźć w rozdziale dziewiątym. Rozdział ten prezentuje zależności między poszczególnymi elementami składającymi się na całość aplikacji. Opisane zostały również metody komunikacji aplikacji zarówno z użytkownikiem jak i systemem. W dodatku zostały zawarte wszystkie parametry pozwalające na sterowanie przebiegiem testów.

2 Sieci neuronowe

Sztuczne sieci neuronowe mają za zadanie symulowanie działań zachodzących w ludzkim mózgu. Za pomocą sztucznych neuronów buduje się struktury podobne do tych z jakimi do czynienia mamy właśnie w ludzkim mózgu [1]. Zasadnicza różnica między elementami sztucznymi a ich pierwowzorem leży w ilości neuronów (w mózgu znajduje się około 10^{11} komórek nerwowych) oraz ich zdolności do równoległej pracy. W niniejszej pracy wykorzystane zostały wprawdzie klasyczne sieci neuronowe jednokierunkowe niemniej jednak próbowano doszukać się możliwości zrównoleglenia procesu ich poszukiwania. W zamyśle metoda miała doprowadzić do skrócenia czasu potrzebnego do znalezienia optymalnej struktury sieci neuronowej.

2.1 Pojedynczy neuron

Uproszczone działanie pojedynczego elementu sieci neuronowej jesteśmy w stanie z powodzeniem symulować. W konsekwencji jesteśmy również w stanie symulować całą sieć neuronową. Początków sztucznych sieci neuronowych upatrywać możemy w pracach McCullocha i Pittsa. W roku 1943 jako pierwsi zaproponowali model neuronu wzorując się na strukturach, z jakimi spotykamy się w mózgu ludzkim [3]. Doszli oni do wniosku, że w dalekim uproszczeniu, działanie neuronu możemy symulować poprzez sumowanie ważonych sygnałów wejściowych. Idąc tym tropem otrzymali model przypominający oryginał. Zaproponowany przez nich sztuczny neuron składał się z czterech elementów: wejść, elementu sumującego, funkcji aktywacji oraz wyjścia



Rysunek 2.1: Schemat neuronu McCullocha i Pittsa

Matematyczna interpretacja rysunku 9.6 ma postać wyrażoną wzorem 2.1.

$$f(x) = f(\sum_i u_i w_i) \tag{2.1}$$

Zapis ten oznacza tyle, że wyjście każdego neuronu to wartość pewnej funkcji aktywacji f w punkcie będącym sumą ważonych wejść neuronu. Sumę ważonych wejść przyjęło się określać

mianem *wartości net*. W pierwotnym modelu funkcja aktywacji była funkcją jednostkową lub funkcją Heaviside'a opisaną wyrażeniem 2.2.

$$f(x) = \begin{cases} 1 & dla\, x \geq 0 \\ 0 & dla\, x < 0 \end{cases} \tag{2.2}$$

Oczywiście model tutaj przedstawiony dalece odbiega od prawdziwego neuronu. Rzeczywiste neurony są dużo bardziej złożone i charakteryzują się następującymi cechami:

- Rzeczywiste neurony często nie są nawet w przybliżeniu elementami progowymi, a wręcz przeciwnie, reagują na wejście w sposób ciągły.
- Potrafią dokonywać operacji logicznych (i, lub, nie) w obszarze drzewa dendrytowego.
- Generują ciąg impulsów, a nie sygnał wyjściowy. Oddawanie pojedynczego impulsu pozbawia informację wielu aspektów, takich jak czas trwania sygnału czy jego faza.
- Posiadają różną skalę opóźnienia. Nie występuje element taktowania przez zegar centralny.

Mimo tych wszystkich wad model McCullocha i Pittsa nadal przedstawia wiele ciekawych własności. Jego autorzy wykazali, że synchroniczny zespół neuronów jest w zasadzie zdolny pracować jako uniwersalna maszyna licząca przy odpowiednio dobranych wagach na połączeniach między neuronami [3]. Badania te spowodowały znaczny wzrost zainteresowanią tego rodzaju metodami obliczeniowymi. Kolejne prace doprowadziły między innymi do określenia nowych funkcji aktywacji neuronu.

Twórcy pierwszego modelu neuronu zakładali, że funkcja aktywacji będzie realizowała funkcję Heaviside'a. Niemniej jednak w miarę postępów badań nad pierwotnym modelem okazało się, że doskonale do tego celu nadają się również inne funkcje. Dodatkowym atutem stosowania innych rozwiązań są nowe właściwości neuronów. Bezpośrednim rozszerzeniem funkcji jednostkowej jest funkcja bipolarna postaci [6] opisana wyrażeniem 2.3.

$$f(x) = \begin{cases} 1 & dla\, x \geq 0 \\ -1 & dla\, x < 0 \end{cases} \tag{2.3}$$

lub

$$f(x) = \begin{cases} 1 & x > 1 \\ -1 & x < 1 \\ x & |x| \leq 1 \end{cases} \tag{2.4}$$

Początkowo tylko te właśnie, progowe, funkcje znajdowały swoje zastosowanie. Obecnie najczęściej stosowana jest funkcja sigmoidalna unipolarna wyrażona wzorem 2.5.

$$f_u(x) = \frac{1}{1 + e^{-\beta x}} \tag{2.5}$$

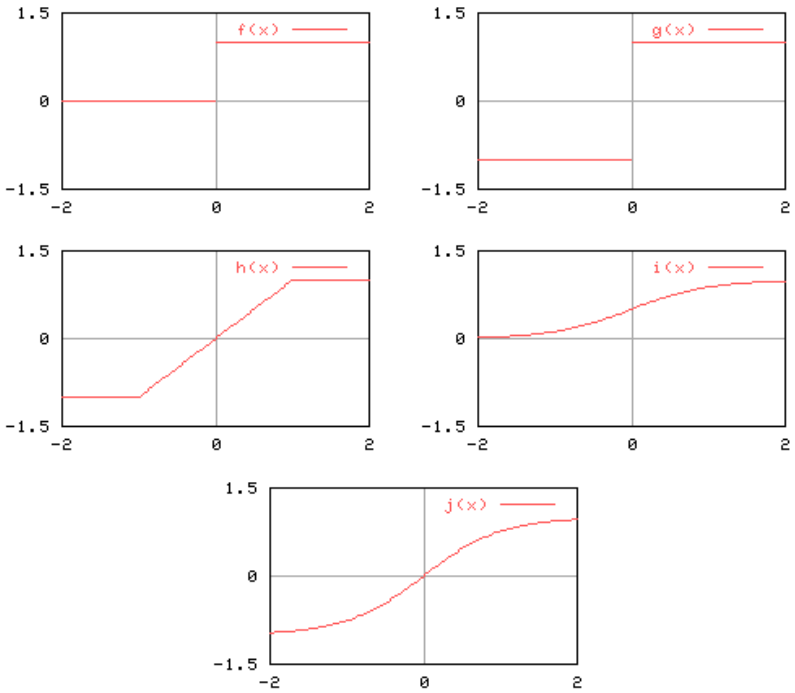
lub bipolarna

$$f_b(x) = tgh(\beta x) \tag{2.6}$$

lub inna jej postać

$$f_b(x) = 2f_u(x) - 1. \tag{2.7}$$

W przeciwieństwie do funkcji (2.2), (2.3), oraz (2.4) posiadają one charakterystyki ciągłe [4]. Wspólną cechą wszystkich funkcji jest natomiast ograniczenie ich wartości. W przypadku funkcji unipolarnych do wartości z przedziału $< 0, 1 >$, zaś dla funkcji bipolarnych do wartości mieszczących się w przedziale $< -1, 1 >$. O znakomitym powodzeniu funkcji sigmoidalnych przesądza znaczne rozszerzenie możliwości neuronu. Szkice wszystkich wymienionych funkcji możemy znaleźć na rys. 9.6.



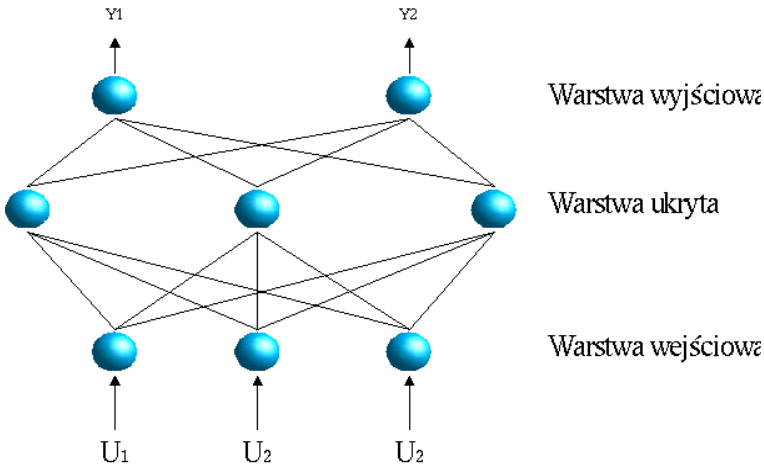
Rysunek 2.2: Przykłady funkcji aktywacji. Funkcje przedstawione na rysunku to odpowiednio f(x) - funkcja unipolarna, g(x) - funkcja bipolarna, h(x) - funkcja jednostkowa, i(x)-funkcja sigmoidalna unipolarna, j(x) - funkcja sigmoidalna bipolarna

2.2 Wielowarstwowa sieć neuronowa

Jak już było wspomniane wyżej, perceptrony proste posiadają znaczne ograniczenia. Mimo swoich dużych możliwości są bezużyteczne, gdy zależności pomiędzy rozpoznawalnymi klasami są bardziej złożone. Aby rozwiązać tego

rodzaju problem musimy sięgnąć po mocniejsze narzędzie. Jest nim sieć warstwowa. Sieć taka cechuje się jasno określonym podziałem na trzy warstwy (rys. 9.6):

- warstwę wejściową - rozprowadzającą sygnał wejściowy do neuronów w warstwie ukrytej,
- warstwę ukrytą - dokonującą przetwarzania sygnałów wejściowych,
- warstwę wyjściową - oddającą sygnał wyjściowy sieci (zwykle wektor), będący opisem klasy do której zostało zaklasyfikowane wejście.



Rysunek 2.3: Przykładowa sieć neuronowa

Wyższość sieci warstwowej nad perceptronem polega na tym, że sieć tego rodzaju posiada znacznie szersze możliwości klasyfikacji wektorów wejściowych. Najprościej rzecz ujmując każda z warstw wprowadza bardziej szczegółowy podział klas przynależności [5]. Pozwala to na coraz lepszą, dokładniejszą analizę danych. W przypadku gdy mamy do czynienia z problemem funkcji XOR wystarczy, że zastosujemy dwie warstwy neuronów, co pozwoli na podział przestrzeni rozwiązań o jakim była mowa w rozdziale (**??**).

Zasada działania takiej sieci jest podobna do działania warstwy perceptronów. W przypadku sieci warstwowej sygnał uznawany za wyjściowy pochodzi z ostatniej warstwy sieci. Dodatkowo wyjścia poprzednich warstw stanowią podstawę do obliczeń dla neuronów w warstwach kolejnych. Proces uczenia również jest podobny. Prezentowane kolejno wzorce stanowią podstawę do oceny jak dobrze zapamiętane w neuronach wagi potrafią odwzorować wektory wejściowe na oczekiwane wektory wyjściowe. Oczywistym jest, że algorytm (**??**) stosowany do modyfikacji wag w perceptronie tutaj nie znajdzie zastosowania. Od razu widać, że różnica między wektorem o, a wektorem y może stanowić podstawę obliczenia błędu tylko dla warstwy wyjściowej.

W przypadku sieci warstwowej musimy w taki sposób powiązać wagi z wyjściem, aby było możliwe określenie ich wpływu na błąd wyjścia. Algorytm określany mianem

propagacji wstecznej błędu dokonuje takiego właśnie powiązania wag i wyjścia sieci [3, 4, 5, 6].

Przez $y_i^{(k)}(n)$, we wzorze 2.8 oznaczmy wyjście i-tej jednostki w k-tej warstwie. W szczególności, dla warstwy ostatniej, będzie to również wyjście sieci.

$$y_i^{(k)}(n) = f(s_i^{(k)}(n)) \tag{2.8}$$

Występujący we wzorze (2.8) czynnik $s_i^{(k)}$, oznacza wartość *net* neuronu i-tego, warstwy k-tej, czyli sumę ważonych sygnałów wejściowych opisaną wzorem 2.9

$$s_i^{(k)}(n) = \sum_{j=0}^{N_{k-1}} w_{ij}^{(k)}(n) \cdot x_j^{(k)}(n). \tag{2.9}$$

W ten sposób możemy określić, jaki wpływ na błąd wyjścia miała warstwa wyjściowa, a następnie, propagując ten błąd w kierunku wejścia sieci ocenić, w jakim stopniu na powstanie błędu wpłynęły wszystkie wagi. Wektor $d = [d_1, ..., d_N]$, oznacza pożądane wyjście z sieci.

$$\varepsilon_i^{(k)} = \begin{cases} d_i^{(L)}(n) - y_i^{(L)}(n) & dla\, k = L \\ \sum_{m=1}^{N_{k+1}} \delta_m^{(k+1)}(n) \cdot w_{mi}^{(k+1)}(n) & dla\, k = 1, ..., L \end{cases} \tag{2.10}$$

Dzięki temu będziemy mogli określić wartość zmiany wag w kolejnym kroku tak, aby wartości na wyjściu lepiej przybliżały wartości oczekiwane. W celu skrócenia formalnego zapisu określającego zmianę wag przedstawiamy go za pomocą δ wyrażonego wzorem 2.11

$$\delta_i^{(k)}(n) = \varepsilon_i^{(k)}(n) \cdot f'(s_i^{(k)}(n)). \tag{2.11}$$

A dalej, korzystając z przyjętej wcześniej wartości współczynnika uczenia η, obliczamy nowe wartości wag (wzór 2.12

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)} + 2\eta\delta_i^{(k)}(n)x_j^{(k)}(n). \tag{2.12}$$

W ten sposób otrzymujemy sieć z zestawem wag lepiej odwzorowującym prezentowane wejścia w oczekiwane wyjścia.

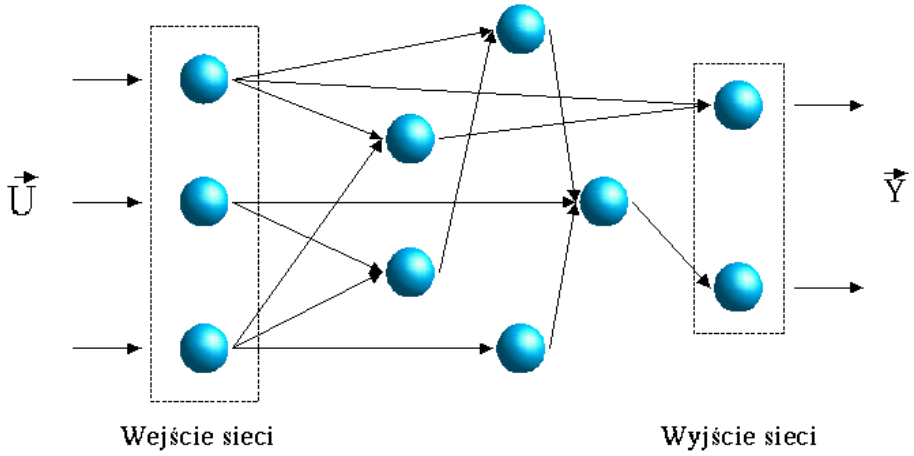
We wzorze (2.11) występuje pochodna funkcji aktywacji. Ze względu na ten fakt koniecznym jest by funkcja aktywacji neuronów była funkcją ciągłą. Pociąga to stosowanie funkcji wyrażonych wzorami (2.5), (2.6) lub (2.7).

Występuje wiele wariacji na temat algorytmu propagacji wstecznej błędu. W tej pracy stosowany jest zarówno klasyczny algorytm, jak i algorytm wzbogacony o tak zwany *człon momentum*. Człon ten pozwala na znaczną poprawę wydajności algorytmu poprzez przyspieszenie jego zbieżności (2.13). Zasada jego działania polega na tym, że kolejna zmiana wag opiera się po części na zmianie poprzedniej. Tym samym zmiany wartości wag są mniejsze niż we wzorze (2.12).

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) + 2\eta\delta_i^{(k)}(n)x_j^{(k)}(n) + \alpha[w_{ij}^{(k)}(n) - w_{ij}^{(k)}(n-1)] \tag{2.13}$$

2.3 Jednokierunkowa sieć neuronowa

Jednokierunkowe sieci neuronowe są niejako kolejnym krokiem w ewolucji sieci neuronowych. Zasada ich działania jest zbliżona do sieci warstwowych. Podstawowa różnica leży w połączeniach między jednostkami aktywacji. W przypadku sieci warstwowej mieliśmy do czynienia z jednoznacznie zarysowanymi warstwami neuronów. Dwie z nich, wejściowa i wyjściowa były wyróżnione. Dodatkowo połączenia między warstwami były pełne. W przypadku sieci jednokierunkowych (ang. feed-forward) całkowicie rezygnuje się z pojęcia warstwy. Na rysunku 9.6 przedstawiona została przykładowa jednokierunkowa sieć neuronowa. Obwiedzione prostokątną ramką jednostki reprezentują zarówno neurony wejściowe, jak i wyjściowe. Nawet wtedy, gdy zrezygnujemy z podziału sieci na warstwy musimy wskazać jednostki, które pozwolą na rozprowadzenie sygnału wejściowego wewnątrz sieci oraz takie, które pozwolą na pobranie wyniku obliczeń jednostek wewnętrz struktury.



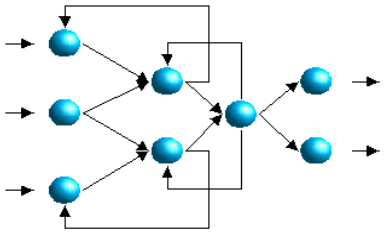
Rysunek 2.4: Sieć jednokierunkowa

Zaletą sieci jednokierunkowych jest fakt, że algorytm propagacji wstecznej (patrz rozdział 2.2) doskonale nadaje się do ich uczenia. Różnice w interpretacji formuły (2.10) sprowadzają się do porzucenia pojęcia warstwy. Obliczając błędy kolejnych jednostek odnosimy się jedynie do jednostek, których sygnały wejściowe pochodzą od aktualnie rozpatrywanego neuronu. Tym samym obliczanie błędu w danej jednostce następuje dopiero wtedy, gdy dla wszystkich jednostek z nią powiązanych błędy zostały już obliczone. Oczywiście człon momentum również doskonale znajduje zastosowanie w przypadku modyfikacji wag sieci jednokierunkowej.

2.4 Rekurencyjne sieci neuronowe

Sieci rekurencyjne to kolejna odmiana sieci warstwowych. Zasadnicza różnica jawi się w możliwości występowania połączeń wstecznych. Oznacza to, że neurony mogą propagować sygnał nie tylko w przód (por. 2.3), ale również wstecz. Jest to wprawdzie rozwiązanie

znacznie bardziej zbliżone do struktur znajdujących się w mózgu ludzkim jednak w tej pracy nie jest rozważane. Schemat sieci rekurencyjnej przedstawia rysunek 9.6.



Rysunek 2.5: Typowa sieć rekurencyjna

Proces uczenia takiej sieci przebiega, w odróżnieniu od poprzednich modeli, w oparciu o dwie jednostki czasu. Mianowicie dla obliczenia wartości sygnałów w aktualnej chwili t , musimy posiadać dokładne informacje o stanie sieci w chwili $(t-1)$. Jest to związane z faktem, że obliczenia jednostek aktywacji mogą opierać się o sygnał wsteczny. W celu uczenia tego typu sieci korzysta się ze zmodyfikowanej wersji algorytmu propagacji wstecznej związanej czasem (ang. Back Propagation through time).

3 Algorytmy genetyczne

Algorytmy genetyczne są niczym innym jak algorytmami poszukiwania opartymi o mechanizm doboru naturalnego i dziedziczności [1]. W dużej mierze ich działanie opiera się o sprawdzony i wydaje się poprawnie funkcjonujący schemat. Algorytmy genetyczne, ze względu na źródło ich inspiracji, posługują się w znacznej mierze słownictwem zaczerpniętym z biologii i genetyki. Jeżeli więc zaczniemy zajmować się tego rodzaju algorytmami, spotkamy się z określeniami *populacja, osobnik,* w przypadku gdy będziemy mówić o zbiorze potencjalnych rozwiązań problemu. Opisując szczegółowo każde rozwiązanie będziemy korzystać z określenia *chromosom*- opisując strukturę rozwiązania. *Genotyp* i *fenotyp* pozwolą nam na odróżnienie zakodowanej reprezentacji *chromosomu* od jego interpretacji dającej w efekcie potencjalne rozwiązanie. Patrząc w głąb *chromosomu* natkniemy się oczywiście na *geny* i towarzyszące im *allele* oraz ich *pozycje*. Te wszystkie określenie nie pozostają bez znaczenia [1]. Bezpośrednia analogia do struktur biologicznych pozwala na ujednolicenie terminologii, a jednocześnie wskazuje, że podstawy działania algorytmów genetycznych istotnie silnie związane są z biologią.

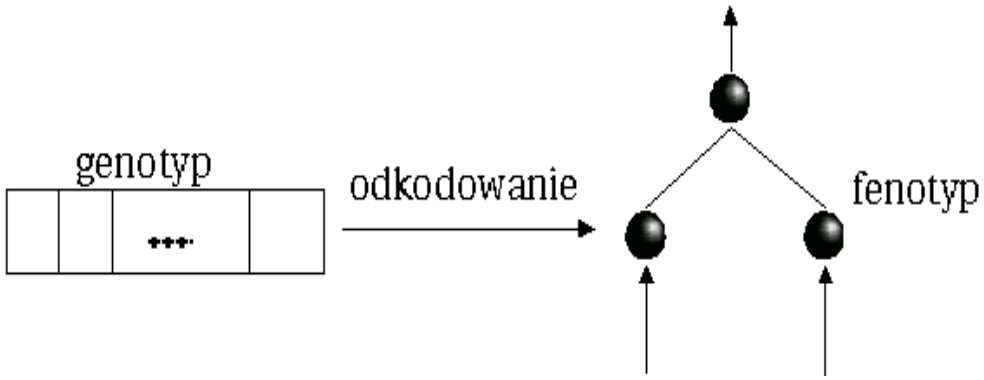
Liczne prace na temat tej metody przeszukiwania przestrzeni, wydawałoby się tak niewiele mającej wspólnego z naukami ścisłymi, wykazują znaczne jej możliwości [1], [2]. Do jej zalet można niewątpliwie zaliczyć przeszukiwanie dużych przestrzeni w kilku miejscach jednocześnie. Szybkie zdobywanie lokalnych maksimów z możliwością ich opuszczenia poprzez odkrywanie nowych obszarów. Jasną i przejrzystą zasadę działania algorytmu. Wszystkie te cechy sprawiają, że algorytmy genetyczne próbuje stosować się w wielu dziedzinach, starając się tym samym zwiększyć wydajność poszukiwania optymalnych rozwiązań.

Stosowanie algorytmów genetycznych do poszukiwań optymalnych sieci neuronowych można w zasadzie podzielić na dwie grupy problemów. Pierwszy, zbieżny z tematem niniejszej pracy, sprowadza się do określenia samej tylko struktury sieci neuronowej [14], [9]. Mówiąc struktury mam na myśli zbiór jednostek aktywacji oraz połączeń między nimi. Rozwiązaniem takiego problemu jest nienauczona, ale optymalna pod względem struktury sieć neuronowa. Druga klasa problemów dotyczy poszukiwania struktury sieci neuronowej wzbogaconej o wartości wag przypisanych do połączeń między jednostkami aktywacji [12], [11]. W niniejszej pracy główny nacisk położony został na rozwiązanie problemu należącego do pierwszej z klas. Niemniej jednak podjęte zostały próby zdefiniowania takiej metody kodowania osobników, żeby ewolucji podlegały nie tylko połączenia pomiędzy jednostkami, lecz również przypisane im wagi.

3.1 Osobnik

Podstawowym elementem algorytmu genetycznego są pojedyncze osobniki. Reprezentują one potencjalne rozwiązania problemu. Określenie osobnik wywodzi się z faktu, że każde potencjalne rozwiązanie reprezentuje zarówno zbiór atrybutów kodujących to rozwiązanie, jak i samo rozwiązanie [1, 2]. Ta analogia do świata zwierzęcego nie jest oczywiście bezcelowa. Zakodowany opis osobnika reprezentuje jego genotyp, gdy tymczasem

jego odkodowana postać jest analogią do fenotypu. Tak naprawdę ciąg kodowy określamy mianem chromosomu, zaś zbiór wszystkich składających się na osobnika chromosomów określamy mianem genotypu. W tej pracy obydwa pojęcia będą wykorzystywane zamiennie. Wynika to z faktu, że genotyp będzie składał się z jednego tylko chromosomu. Widzimy więc, że określenie osobnik przenosi znaczną dawkę informacji. Niedość że wykonując na osobnikach operacje genetyczne potrafimy decydować o ich genotypie, to dodatkowo jesteśmy w stanie porównywać je między sobą poprzez ocenę ich fenotypu. Rysunek 9.6 przedstawia zależności pomiędzy terminami przytoczonymi powyżej. Widzimy wyraźnie, że obraz osobnika (fenotyp) nierozerwalnie związany jest z jego zapisem genetycznym (genotyp). Zaznaczony strzałką na rysunku 9.6 proces odkodowania informacji w genotypie polega na takim jej przetworzeniu, aby uzyskać interesującą nas strukturę. Oczywiście w tej pracy dotyczyło to takiej transformacji genotypu, aby w efekcie uzyskać w pełni funkcjonalną sieć neuronową.



Rysunek 3.1: Zależność między genotypem i fenotypem

Rozwiązanie problemu za pomocą algorytmu genetycznego nierozerwalnie związane jest z opisem każdego potencjalnego rozwiązania za pomocą ciągu kodowego. Rozróżnia się w zasadzie dwie metody opisu osobnika. Kodowanie binarne [2] oraz kodowanie tak zwane mocne [14]. W kodowaniu binarnym każda pozycja chromosomu może przyjąć tylko jedną z dwu możliwych wartości 0 lub 1. Jest to bardzo ubogie, ale jednocześnie oszczędne pod względem obliczeniowym rozwiązanie. Dodatkowo, zawsze mamy pewność, że osobniki są poprawnymi rozwiązaniami problemu. Korzyści płynące z oszczędności w zapisie informacji są czasami niwelowane małą jej wartością. Ograniczona postać genotypu sprawia, że do zapisu bardziej skomplikowanych struktur wymagany jest wiekszy rozmiar chromosomu. Częstokroć pozycje bitowe zastępuje się pozycjami całkowitymi z tym jednak ograniczeniem, że każda z nich może przyjmować wartości ze zbioru $\{0, 1\}$. Nie ma to oczywiście większego wpływu na cały proces obliczeniowy z wyjątkiem uproszczenia obliczeń.

Kodowanie mocne opiera się o złożone struktury danych opisujące każde z rozwiązań. Zmuszeni jesteśmy wtedy do wprowadzenia własnych operatorów genetycznych i częstokroć większej dbałości o poprawność merytoryczną genotypu [9, 10, 11]. Oczywiście procentuje

to w efektywności zapisu, może jednak prowadzić do powstawania osobników nie będących poprawnymi rozwiązaniami problemu. Konieczna jest wtedy taka korekcja genotypu, aby osobnik spełniał określone ograniczenia.

Rozpatrzmy problem poszukiwania maksimum funkcji $f(x) = x^2$ w przedziale domkniętym $< 0; 31 >$ [1]. Natychmiast możemy przyjąć, że genotyp będzie się składał z pięciu pozycji binarnych. Każda z tych piątek będzie opisywała potencjalne rozwiązanie. Fenotypem każdego ciągu kodowego będzie natomiast dziesiętna wartość tego ciągu. Zależność ta została przedstawiona w tabeli 9.6.

Tabela 3.1: Przykład zależności pomiędzy fenotypem a genotypem, dla problemu znalezienia maksimum funkcji $f(x) = x^2$w przedziale $< 0; 31 >$

genotyp	fenotyp
00000	0
01011	11
00011	3
10000	16

W tabeli 9.6 możemy zauważyć kilka różnych pozycji reprezentujących różne rozwiązania. Jest to oczywiście namiastka procesu poszukiwania optymalnego rozwiązania. Algorytm genetyczny nie operuje na pojedynczych rozwiązaniach, a na całych ich zbiorach.

3.2 Populacja

Pod pojęciem populacji rozumiemy zbiór wszystkich możliwych rozwiązań danego problemu. Operacje dokonywane na całej populacji są związane z przetwarzaniem informacji zawartej w genotypie poszczególnych osobników. Wprowadzenie pojęcia populacji ma w zasadzie znaczenie symboliczne, jednak ułatwia znacznie rozumienie i operowanie algorytmami genetycznymi. Populacje pozwalają na przejrzyste wyznaczenie przystosowania każdego z osobników w obrębie danej populacji, wyznaczenie przystosowania średniego, czy minimalnego i maksymalnego. Dodatkowo znając maksymalny rozmiar populacji możemy kontrolować operacje krzyżowania i klonowania osobników.

Z punktu widzenia wykorzystania algorytmu genetycznego istotne są:

- zbiór osobników całej populacji,
- liczebność populacji. Zwykle w trakcie obliczeń niezmienna, ale spotyka się implementacje, w których liczebność osobników może się zmieniać dynamicznie zależnie od spełnionych warunków [10],
- wartości przystosowania osobników: maksymalne, minimalne, średnie. Wartości te pozwalają na precyzyjne określenie jakości osobnika w obrębie populacji oraz późniejszą selekcję materiału genetycznego przeznacznonego do reprodukcji.

Naturalnym jest pytanie, w jaki sposób przebiega proces znajdowania najlepszych rozwiązań wewnątrz populacji. Kluczowymi pojęciami są wartość przystosowania i funkcja przystosowania. Wartości te pozwalają na ocenę jakości osobnika w obrębie pojedynczej populacji. Znając przystosowania poszczególnych osobników jesteśmy w stanie odrzucić z populacji te, których przystosowanie jest mierne (poniżej średniej), pozostawić te osobniki, których przystosowanie jest bliskie średniemu oraz powielić te, których jakość przewyższa przystosowanie średnie [2]. Ważnym elementem jest oczywiście określenie funkcji przystosowania w taki sposób, aby obiektywnie oceniała wszystkie potencjalne rozwiązania. Dodatkowo wartości tej funkcji powinny być porównywalne.

3.3 Funkcja przystosowania

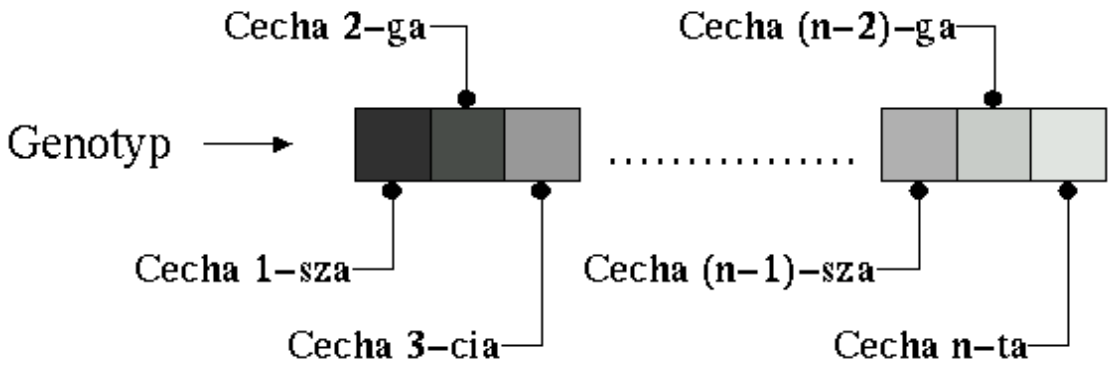
Funkcja przystosowania pozwala na takie wyznaczenie wartości przystosowania każdego z rozwiązań, aby możliwe było ich porównanie. W zasadzie dla każdego problemu musimy zdefiniować na nowo funkcję przystosowania [2]. Jest ona ściśle związana z zakodowaną reprezentacją osobnika (*genotypem*), jak i jego odkodowaną postacią ($fenotypem$). Typowy przebieg tej funkcji sprowadza się do odkodowania genotypu, a następnie ocenie przydatności fenotypu.

Lakoniczne stwierdzenia „dekodowanie genotypu“ oraz „ocena przydatności fenotypu“, mogą często reprezentować bardzo skomplikowane operacje. Związane jest to ze stopniem skomplikowania zarówno genotypu, jak i fenotypu. W przypadku sieci neuronowych operacje te wiążą się zwykle ze zamianą genotypu osobnika na sieć neuronową, a następnie oceną poszczególnych rozwiązań. W skład oceny sieci neuronowej może wchodzić jej rozmiar, liczba połączeń, czas uczenia, zdolność generalizacji.

3.4 Reprezentacja osobnika

Najczęściej stosowaną reprezentacją chromosomu pojedynczego osobnika jest przedstawienie go za pomocą ciągu kodowego, w którym każda pozycja lub ich zbiór definiuje określoną cechę osobnika. Ten sposób zapisu jest oczywiście bardzo wygodny ze względu na fakt, że ten sam obszar genotypu w każdym z osobników odpowiada za te same cechy fenotypu. Pozwala to na proste i eleganckie wykonywanie operacji genetycznych w obrębie tego typu chromosomów. Na rysunku 9.6 możemy zobaczyć typową reprezentację genotypu.

Reprezentowana na rysunku 9.6 przez pojedynczą pozycję cecha może być bardzo złożoną strukturą danych. Oczywiście nie jest to jedyna możliwa reprezentacja postaci osobnika. Spotyka się struktury listowe, drzewiaste, czy nawet gramatyki. Szerzej o różnych metodach reprezentacji genotypu traktuje rozdział poświęcony stosowaniu algorytmów genetycznych w obrębie sieci neuronowych.



Rysunek 3.2: Reprezentacja genotypu

3.5 Operatory genetyczne

Operatory genetyczne są podstawowymi operacjami pozwalającymi na funkcjonowanie algorytmu genetycznego. Ich stosowanie prowadzi do modyfikacji materiału genetycznego przenoszonego przez każdego z osobników. Tym samym możemy pozyskiwać coraz lepsze rozwiązania [1, 2, 6]. Nieodzownym elementem każdego algorytmu genetycznego jest element losowości. Podstawą każdej operacji genetycznej jest generacja wartości losowej, zwykle z przedziału $< 0; 1 >$. Każda z trzech podstawowych operacji genetycznych *selekcja*, *krzyżowanie* oraz *mutacja* w dużej mierze opiera się o wartości losowe. Operacje te mogą zajść tylko wtedy, gdy prawdopodobieństwo ich wystąpienia jest odpowiednio większe od współczynników selekcji, krzyżowania oraz mutacji. Wykorzystywane pojedynczo nie doprowadzą do wyznaczenia rozwiązania optymalnego. Wykorzystanie ich jednocześnie do takiego rozwiązania może doprowadzić.

3.5.1 Selekcja

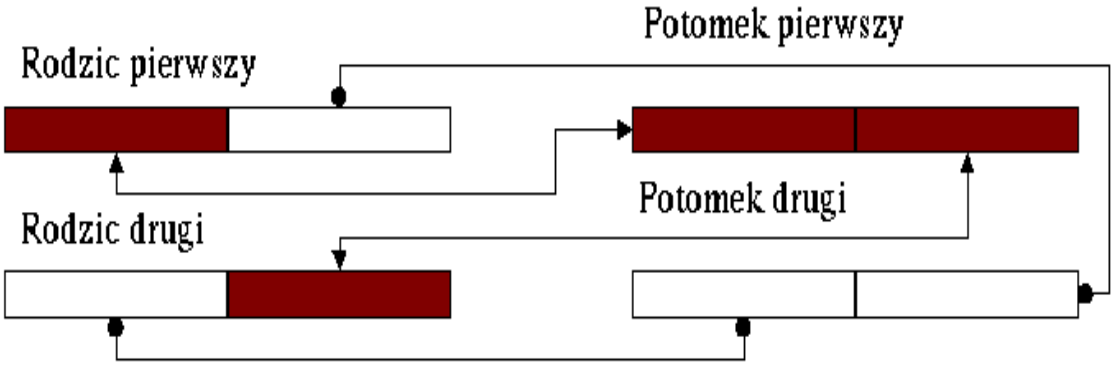
Selekcja jest operacją dokonującą naturalnego wyodrębnienia z populacji osobników lepiej przystosowanych. Co za tym idzie do nowej, kolejnej generacji, znacznie łatwiejszy dostęp będą miały osobniki lepiej przystosowane (cechujące się większą wartością funkcji przystosowania). Dzięki temu lepszy materiał genetyczny będzie ulegał powieleniu oraz krzyżowaniu z innym jemu podobnym pod względem jakościowym. Osobniki przystosowane źle, (to znaczy posiadające przystosowanie mniejszego od średniego) będą miały bardzo małą szansę przekazania swoich genów kolejnym pokoleniom.

Wybór osobnika polega na wykorzystaniu wykalibrowanej ruletki, której powierzchnia została podzielona wedle wartości przystosowań osobników. W oparciu o te wartości dokonywany jest wybór osobnika przeznaczonego do krzyżowania. Oczywistym jest, że czym większe przystosowanie osobnika, tym większy obszar na ruletce dla niego przeznaczony, co za tym idzie również większa szansa wyboru tego właśnie osobnika. Widzimy więc, że jakość osobnika w znacznym stopniu determinuje jego szansę na przekazanie genów, przynajmniej częściowe. Należy bowiem pamiętać, że *współczynnik*

krzyżowania decyduje, w jakim stopniu osobniki ulegają temu procesowi, a w jakim przechodzą do nowej populacji w niezmienionym stanie.

3.5.2 Krzyżowanie

Wybrane, w czasie selekcji, osobniki podlegają późniejszemu procesowi krzyżowania. Oczywiście tylko wtedy, gdy faktycznie krzyżowanie ma zajść. Proces krzyżowania polega na wymianie materiału genetycznego między dwoma osobnikami tak, aby powstały dwa nowe osobniki posiadające cechy wspólne obu rodziców. Tak naprawdę wymianie podlega informacja genetyczna zawarta w chromosomach obu rodziców. Na rysunku 9.6 widzimy przedstawiony w sposób symboliczny proces krzyżowania.



Rysunek 3.3: Proces krzyżowania

Jak pokazano na rysunku 9.6 widzimy, że chromosomy obydwu rodziców są w taki sposób dzielone, że każdy z nowo powstałych osobników ma rozmiar identyczny z rodzicami, ale posiada wymieszany materiał genetyczny. Takie postępowanie pozwala na łączenie cech najlepszych osobników, przez co prowadzi do osiągania coraz lepszych przystosowań przez ich potomstwo.

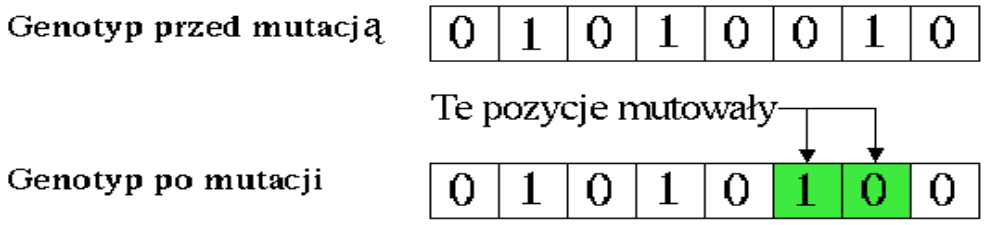
Punkt podziału genotypu następuje w miejscu dowolnym jednak punkt podziału mieści się w obrębie każdego z osobników. Fakt, że na rysunku (9.6) punkt podziału przebiega dokładnie w połowie chromosomu ma znaczenie symboliczne.

3.5.3 Mutacja

Same operacje krzyżowania i selekcji doprowadziłyby w pewnym momencie do stabilizacji genotypu. Okazałoby się, że mamy do czynienia z populacją osobników o bardzo podobnych własnościach. Dodatkowo kolejne operacje genetyczne nie wprowadzałyby istotnych zmian wewnątrz populacji.

Najbardziej niepożądanym zjawiskiem jest to, aby osobniki zdobyły lokalne maksimum, a następnie zaprzestały dalszych poszukiwań. Z takim zjawiskiem mamy właśnie do czynienia wtedy, gdy materiał genetyczny nie ulega żadnym modyfikacjom.

Aby zapobiegać tego rodzaju sytuacjom wprowadza się pojęcie mutacji. Polega ono na losowej zamianie dowolnej pozycji chromosomu. Rysunek 9.6 przedstawia taki właśnie chromosom. Taki, to znaczy, poddany działaniu mutacji.

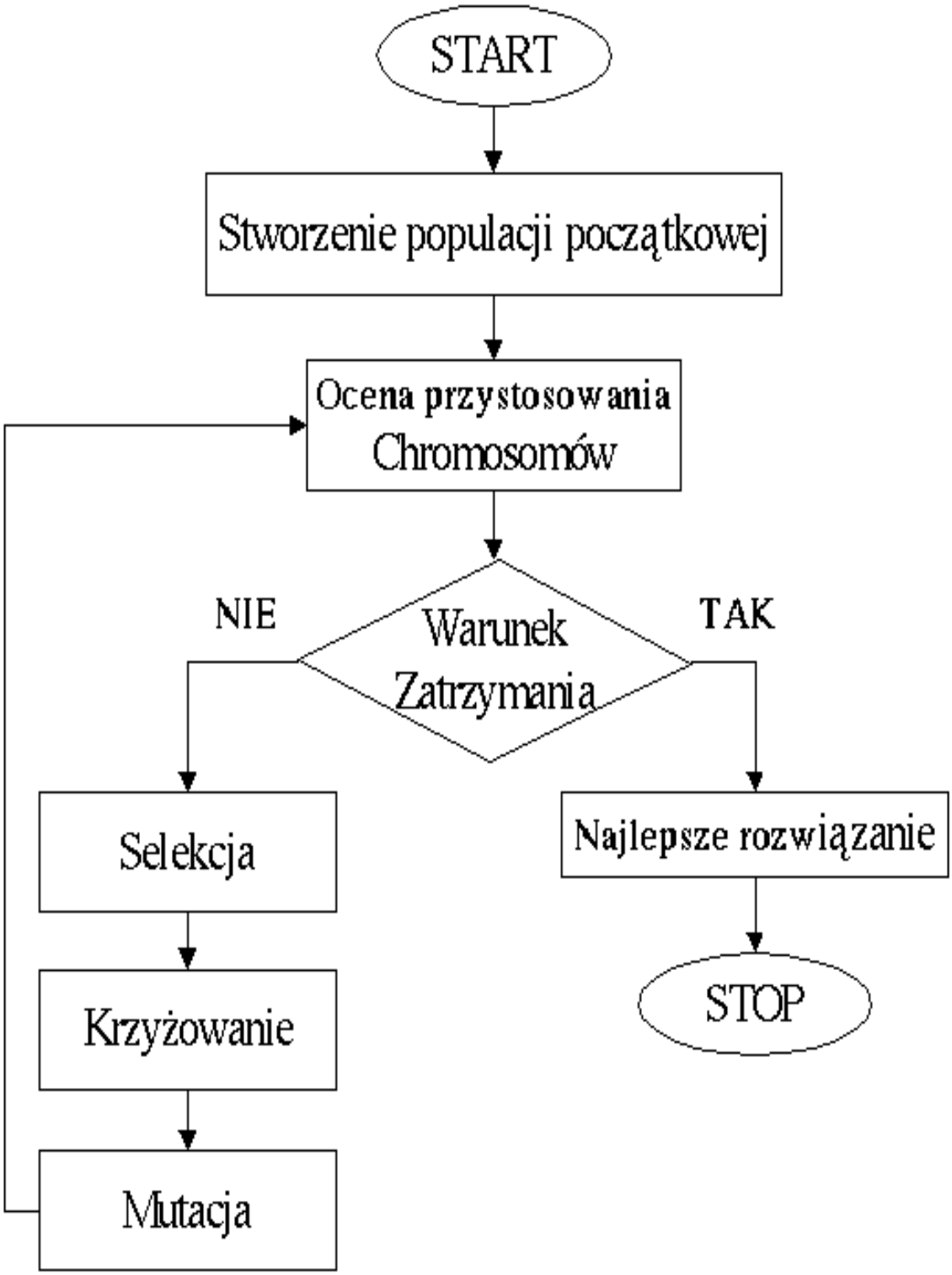


Rysunek 3.4: Mutacja genotypu

Należy pamiętać, że każda pozycja genotypu może podlegać mutacji. To czy zajdzie, czy nie, zależy od *współczynnika mutacji*. Jego wpływ polega na określeniu, jak duża jest szansa pojawienia się mutacji w przypadku każdej z pozycji genotypu. Operacja negacji ma sens jedynie dla genotypu, którego pozycje są wartościami binarnymi. W przeciwnym wypadku zmuszeni jesteśmy opracować własną procedurę modyfikacji pozycji genotypu. Ma to o tyle duże znaczenie, że przypadkowa zmiana wartości na danej pozycji może doprowadzić do powstania osobnika nie spełniającego ograniczeń przed nim stawianych. Jeżeli, przykładowo, każda z pozycji przenosi informację w postaci liczby zmienno-przecinkowej z przedziału $< 0; 1 >$, to będziemy musieli zadbać o to, by po zajściu mutacji wartość na danej pozycji należała do zadanego przedziału.

3.6 Typowy przebieg algorytmu genetycznego

W poprzednich rozdziałach przedstawione zostały wszystkie elementy niezbędne do poprawnej pracy algorytmu genetycznego. Za ich pomocą możemy zapisać typowy przebieg algorytmu genetycznego. Schemat algorytmu przedstawiony na rysunku 9.6 reprezentuje znakomitą większość stosowanych rozwiązań. Oczywiście różnice między nimi tkwią wewnątrz poszczególnych procedur. Spotyka się również inne, rozbudowane modyfikacje pierwotnego wzoru. Mowa tutaj o równoległych, czy rozproszonych algorytmach genetycznych. Niemniej jednak zwykle modyfikacje pierwowzoru sprowadzają się do określenia oryginalnej reprezentacji osobników, nowej funkcji oceny, oraz metod krzyżowania.



Rysunek 3.5: Przebieg typowego algorytmu genetycznego

4 Zastosowanie algorytmów genetycznych do projektowania sieci neuronowych

Poprzednie rozdziały traktowały o sieciach neuronowych oraz algorytmach genetycznych. Niniejsza część pracy poświęcona jest zastosowaniu algorytmów genetycznych do projektowania sieci neuronowej.

Zostaną zaprezentowane rozwiązania opisane w innych pracach, głównie te, na które zostały zaimplementowane i przebadane w niniejszej pracy.

4.1 Reprezentacja sieci neuronowej

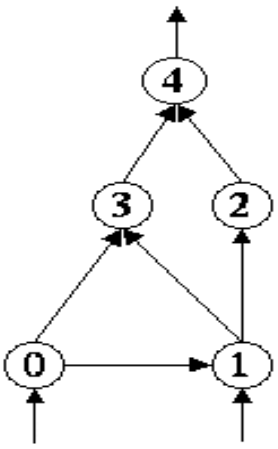
Istotnym elementem algorytmu genetycznego jest reprezentacja poszczególnych rozwiązań. Istnieje wiele rozmaitych metod reprezentacji genotypu. Zależnie od oczekiwanego rozwiązania autorzy decydują się na poszukiwanie coraz nowszych metod reprezentacji sieci neuronowej lub starają się udoskonalić już istniejące. Oba podejścia mają swoje uzasadnienie. Z jednej strony poszukiwanie nowych reprezentacji sieci neuronowej może prowadzić do bardzo ciekawych wyników. Z drugiej strony, kiedy staramy się zbadać własności innych cech algorytmu niż zapis sieci neuronowej, wystarczy wykorzystać jedną ze znanych i sprawdzonych metod. Pozwala to na ocenę samego tylko rozwiązania. Często okazuje się, że samo rozszerzenie znanych metod reprezentacji, przykładowo wprowadzenie zmodyfikowanej funkcji oceny, pozwala na osiągnięcie lepszych wyników.

Głównym celem w opisie sieci neuronowej jest dążenie do prostego oraz efektywnego zapisu połączeń między jednostkami aktywacji. Jeżeli opisujemy tylko i wyłącznie strukturę sieci, to w reprezentacji będziemy starali się zapisać informacje o wszystkich możliwych oraz aktywnych połączeniach między jednostkami. Często w obrębie struktury sieci neuronowej zapisuje się również informacje dotyczące wag połączeń. W takim pzypadku w zapisie genotypu przechowuje się dodatkowo informację na temat wartości tych wag. Zasadnicza różnica między poszczególnymi reprezentacjami tkwi w operacjach genetycznych oraz funkcji oceny.

4.2 Macierzowa reprezentacja struktury sieci neuronowej

Jest to jedna z podstawowych metod reprezentacji sieci neuronowej [8]. Opis połączeń zapisany jest na przecięciach kolumn i rzędów macierzy. Zależnie od tego, czy dopuszczamy rekurencję połączeń czy też nie, wykorzystana jest albo cała macierz, albo tylko jedna jej trójkątna połowa. Rysunek 9.6 przedstawia sieć neuronową oraz równoważny jej opis za pomocą macierzy połączeń. Wystąpienie jedynki na przecięciu i-tej kolumny i j-tego rzędu oznacza, że występuje połączenie między neuronem j-tym oraz i-tym. Połączenie biegnie od neuronu j-tego. Jeżeli w macierzy występuje zero, to połączenie nie istnieje.

Na sieć zostało nałożone tylko jedno ograniczenie, mianowicie takie, że nie mogą występować połączenia idące od jednostek o indeksach większych do jednostek o indeksach mniejszych. Ograniczenie takie eliminuje możliwość powstawania cykli.



neurony	0	1	2	3	4
0	x	1	0	1	0
1	x	x	1	1	0
2	x	x	x	0	1
3	x	x	x	x	1
4	x	x	x	x	x

Rysunek 4.1: Sieć neuronowa jednokierunkowa oraz odpowiadająca jej macierz połączeń

Macierz przedstawioną na rysunku 9.6 zwykle zamienia się na ciąg kodowy w taki sposób, że kolejne wiersze macierzy zapisuje się na kolejnych pozycjach genotypu. Zapis taki prezentuje rysunek 9.6.

1	0	1	0	1	1	0	0	1	1

Rysunek 4.2: Zapis macierzy połączeń za pomocą ciągu

Przedstawiony sposób opisu sieci neuronowej wykorzystany był jako podstawowy podczas przeprowadzania testów obliczeniowych. Dodatkowo został on w końcowych eksperymentach wzbogacony o wartości wag oraz współczynniki mutacji przypisane do każdego połączenia. W ten sposób przechowywane one były razem z informacją o występowaniu połączenia. Macierz przedstawiona na rys. 9.6 przedstawia tę samą strukturę sieci, z którą mieliśmy do czynienia na rysunku 9.6. Różnica między tą reprezentacją a reprezentacją macierzową polega na rozszerzeniu informacji przenoszonej na pojedynczej pozycji. W reprezentacji z rysunku 9.6 dodatkowo przenoszone są informacje dotyczące wartości wagi przypisanej do każdego połączenia. Dodatkowo umieszczona została informacja o tym, w jakim stopniu może się zmienić wartość wagi podczas mutacji. Przyjęto, że wartość zmiany musi należeć do przedziału $< -0.01; 0.01 >$.

Na rysunku 9.6 każda pozycja reprezentowana jest przez trzy atrybuty. Pierwszy atrybut p decyduje o istnieniu połączenia w zależności od tego czy w tym miejscu wystąpi 0 lub jeden odpowiednio połączenia między neuronami nie będzie lub wręcz odwrotnie, będzie

neurony	0			1			2			3			4		
	p	w	m	p	w	m	p	w	m	p	w	m	p	w	m
0	x	x	x	1	0.5	-0.01	0	-0.1	0.02	1	0.1	-0.03	0	0.1	0.01
1	x	x	x	x	x	x	1	0.9	-0.03	1	0.6	0.05	0	-0.21	-0.03
2	x	x	x	x	x	x	x	x	x	0	-0.1	-0.04	1	0.23	-0.07
3	x	x	x	x	x	x	x	x	x	x	x	x	1	0.15	-0.01
4	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Rysunek 4.3: Rozbudowana reprezentacja macierzowa

ono istniało. Drugi atrybut w określa wartość wagi przypisanej do tego właśnie połączenia. Niezależnie od wartości atrybutu określającego istnienie połączenia wartość wagi zawsze istnieje i zawsze podlega mutacji. Trzeci atrybut m decyduje jak bardzo może ulec zmianie wartość wagi w trakcie mutacji. Powinno to zapobiec niekontrolowanej zmienności wartości wag.

Zasadnicza różnica między obiema reprezentacjami tkwi w procesie oceny osobników. Pierwsza reprezentacja wymaga uczenia każdej sieci od początku. Proces ten pochłania bardzo dużo czasu. Niemniej jednak szansa znalezienia poprawnego osobnika jest duża. Koniecznym jest jedynie, by sygnał wejściowy docierał do neuronów wyjściowych.

W drugiej reprezentacji ocenie również podlega sieć zbudowana w oparciu o reprezentujący ją genotyp. Nie podlega ona jednak uczeniu, a tylko ocenie jej zdolności rozpoznawania prezentowanych wzorców. Oczywiście sam proces oceny ulega znacznemu skróceniu, ale dużo trudniej jest znaleźć prawidłową strukturę sieci. W tej reprezentacji nie wystarczy, aby połączenia pozwalały na poprawny przebieg sygnału. Wymagane jest, aby wagi przypisane do odpowiednich połączeń pozwalały na poprawną klasyfikację wzorców.

4.2.1 Czyszczenie genotypu

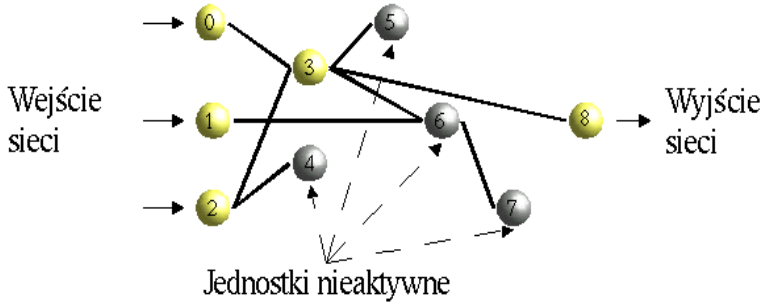
Opisana wyżej reprezentacja sieci może przenosić informacje na temat sieci niepoprawnych lub zawierających zbędne połączenia. W celu uniknięcia zbędnego uczenia sieci niezdolnych do zapamiętania prezentowanych wzorców, genotyp poddawany jest procesowi czyszczenia.

Ocenę jakości genotypu musimy zacząć od sprawdzenia, które z występujących w genotypie połączeń są aktywne. Tabela 9.6 przedstawia macierz, w której część informacji sprawia, że struktura sieci zbudowanej w oparciu o tę właśnie macierz zawiera połączenia nieaktywne. Graficznie sytuacja zaprezentowana została na rysunku 9.6. Wyczyszczenie sieci ze zbędnych połączeń polega na rekurencyjnym przejściu przez całą strukturę i zaznaczeniu, jednostek które powinny w niej pozostać. Będą to te wszystkie jednostki, których sygnał wyjściowy dociera do neuronów wyjściowych sieci.

Ten element oceny pozwoli na pozbycie się nadmiaru połączeń (zresztą bezużytecznych), ale nie pozwala na stwierdzenie czy sieć jest, chociaż potencjalnie, w stanie nauczyć się

	0	1	2	3	4	5	6	7	8
0	x	0	0	1	0	0	0	0	0
1	x	x	0	0	0	0	1	0	0
2	x	x	x	1	1	0	0	0	0
3	x	x	x	x	0	1	1	0	1
4	x	x	x	x	x	0	0	0	0
5	x	x	x	x	x	x	0	0	0
6	x	x	x	x	x	x	x	1	0
7	x	x	x	x	x	x	x	x	0
8	x	x	x	x	x	x	x	x	x

Rysunek 4.4: Genotyp sieci rozpoznającej parzystość liczby zawierający wadliwe połączenia



Rysunek 4.5: Graficzna reprezentacja macierzy przedstawionej na rysunku 9.6

rozpoznawania zadanych wzorców. Do tego, aby istniała szansa poprawnego zakończenia procesu uczenia, koniecznym jest, by wszystkie jednostki wyjściowe sieci były aktywne. Po stwierdzeniu, czy ten warunek zachodzi możemy określić, czy sensownym jest rozpoczynać proces uczenia. W prezentowanej aplikacji genotyp przedstawiony na rysunku 9.6 zostanie oczyszczony w taki sposób, że połączenia biegnące zarówno do, jak i od neuronów 4,5,6,7 zostaną usunięte. W ten sposób otrzymamy w pełni poprawną strukturę sieci neuronowej.

Oczyszczony w ten sposób genotyp podlega dalszemu procesowi oceny. Uzyskana w ten sposób sieć neuronowa zostaje poddana procesowi uczenia, a następnie testowana na zdolność uogólniania.

4.2.2 Ocena osobnika

Ocena poszczególnych osobników polega na zamianie ich reprezentacji na strukturę sieci, a następnie uczeniu ich za pomocą algorytmu propagacji wstecznej. Prezentowane wzorce uczące mają doprowadzić do poprawnego zapamiętania w wagach informacji na temat cech istotnych dla poprawnej klasyfikacji. Po zakończeniu procesu uczenia następuje ocena zdolności uogólniania sieci. Testowanie tej zdolności polega na prezentacji wzorców, którymi sieć neuronowa nie była uczona, tak zwanych wzorców testujących. Ocenę zdolności uogólniania określa się na podstawie stopnia poprawnie sklasyfikowanych

wzorców. Dodatkowo ocenie podlega sama struktura. Składają się na nią aktywne jednostki sieci oraz aktywne połączenia.

Proces oceny rozpoczyna czyszczenie genotypu, po którym następuje próba uczenia osobnika, jeżeli struktura na to pozwala.

Funkcja oceny osobników jest złożoną funkcją. Ze względu na fakt, że nie istnieją matematyczne metody oceny jakości struktury sieci neuronowej proces ten opiera się głównie na ocenie zdolności uczenia się sieci.

Funkcja oceny skonstruowana została w taki sposób, że oceniane są poszczególne atrybuty sieci, mające istotne znaczenie dla procesu poszukiwania optymalnej jej struktury. Wśród nich można wymienić:

- stosunek liczby wszystkich połączeń występujących w sieci do liczby możliwych połączeń,
- stosunek liczby aktywnych połączeń do połączeń występujących w genotypie,
- czas uczenia sieci neuronowej wyrażony liczbą prezentacji wzorców uczących,
- zdolność uczenia się zdefiniowana jako stosunek liczby wykorzystanych cykli uczących do dostępnej liczby cykli uczących,
- zdolność uogólniania sieci neuronowej wyrażona stosunkiem prawidłowo rozpoznanych wzorców testowych do całkowitej ich ilości.

Biorąc pod uwagę, że każda sieć podlega procesowi uczenia ocena poszczególnych rozwiązań jest czasochłonna.

4.2.3 Funkcja oceny

Przyjęta dalej funkcja oceny jest dosyć skomplikowana. Została ona skonstruowana w taki sposób, że reprezentuje kombinację liniową parametrów opisujących sieć. Zestaw wszystkich parametrów zawiera tabela 9.6.

Tabela 4.1: Parametry wykorzystywane do konstrukcji funkcji przystosowania oraz ich objaśnienia

conMax	- maksymalna liczba możliwych połączeń w sieci
conGen	- liczba połączeń w genotypie
aConGen	- liczba aktywnych połączeń w genotypie
learnMax	- maksymalna liczba cykli uczących
learnCnt	- wykorzystana liczba cykli uczących
testCnt	- liczba wzorców testowych
accTestCnt	- liczba rozpoznanych wzorców testowych

Zapis funkcji fitness przedstawia wzrór (4.1).

$$fit = w_0 f_0(x) + w_1 f_1(x) + ... + w_{17} f_{17}(x) \tag{4.1}$$

gdzie funkcje $f_1, f_2, ..., f_{17}$ mają następującą postać:

$$\begin{aligned}
f_0 &= \frac{aConGen}{conGen}, \\
f_1 &= 1 - \frac{aConGen}{conGen}, \\
f_2 &= \frac{conGen}{conMax}, \\
f_3 &= 1 - \frac{conGen}{conMax}, \\
f_4 &= \frac{learnCnt}{learnMax}, \\
f_5 &= 1 - \frac{learnCtn}{learnMax}, \\
f_6 &= \frac{accTestCnt}{testCnt}, \\
f_7 &= 1 - \frac{accTestCnt}{testCnt}, \\
f_8 &= aConGen, \\
f_9 &= \frac{1}{aConGen}, \\
f_{10} &= conGen, \\
f_{11} &= \frac{1}{conGen}, \\
f_{12} &= learnCnt, \\
f_{13} &= \frac{1}{learnCnt}, \\
f_{14} &= accTestCnt, \\
f_{15} &= \frac{1}{accTestCnt}, \\
f_{16} &= conMax.
\end{aligned}$$

Zdefiniowanie funkcji oceny sprowadza się w zasadzie do określenia wektora wag $w = [w_0, ..., w_{17}]$ w taki sposób, aby zadany problem był rozwiązywalny przez algorytm genetyczny.

W przypadku wykorzystania reprezentacji macierzowej z wagami, na wartość przystosowania nie składa się czas potrzebny na nauczenie sieci. Jest to związane z faktem, że sieć taka traktowana jest jako nauczona, a ocenie podlega jej zdolność uogólniania. W tym przypadku proces testowania opiera się o wzorce uczące.

4.2.4 Krzyżowanie

W przyjętej dalej reprezentacji sieci neuronowej operacja krzyżowania polega na podziale genotypu przedstawionego za pomocą ciągu kodowego (rys. 9.6 oraz 9.6). Otrzymane połówki genotypu są następnie łączone naprzemiennie, dzięki czemu otrzymujemy dwa ciągi kodowe, z których każdy zawiera część informacji pochodzącej od każdego z rodziców.

W przypadku reprezentacji macierzowej z uwzględnieniem wartości wag, proces krzyżowania przebiega dokładnie tak samo.

4.2.5 Mutacja

Proces mutacji w reprezentacji macierzowej polega na losowej zamianie każdej z pozycji genotypu. Współczynnik mutacji określa, z jakim prawdopodobieństwem mutacja może wystąpić. W trakcie dokonywania mutacji nie jest oceniana poprawność genotypu. Dopiero podczas oceny poszczególnych rozwiązań wykorzystane jest badanie czy rozwiązanie jest poprawne (posiada wystarczającą liczbę połączeń). Mutacja polega na negacji pozycji genotypu, która ma podlegać mutowaniu.

Inaczej wygląda sytuacja w przypadku reprezentacji macierzowej z wagami. W tym przypadku mutacji podlega każdy parametr pozycji, a zatem współczynnik określający istnienie połączenia (określający czy dane połączenie między jednostkami występuje), waga przypisana do tego połączenia oraz współczynnik mutacji wagi. Mutacja współczynnika połączenia jest oczywista (może on przyjmować wartości 0 lub 1). Mutacja wagi polega na modyfikacji wartości wagi o wartość współczynnika mutacji wagi. Z kolei współczynnik mutacji wagi modyfikowany jest o wartość losową, z przedziału $< -0.01; 0.01 >$.

4.2.6 Selekcja

Selekcja opiera się o wartość funkcji przystosowania. W aplikacji wykorzystano mechanizm ruletki, gdzie każdy z osobników otrzymuje pewną powierzchnię pola wykalibrowanej ruletki. Rozmiar tego pola zależy od wartości przystosowania osobnika. Istotny wpływ na funkcję oceny poszczególnych rozwiązań mają współczynniki przypisane do każdej składowej funkcji. Odpowiednie dobranie wartości współczynników może znacznie zmienić końcowe rozwiązanie problemu.

Po zakończeniu procesu selekcji następuje proces krzyżowania, a dalej mutacja. Przygotowana w ten sposób populacja podlega ponownej ocenie.

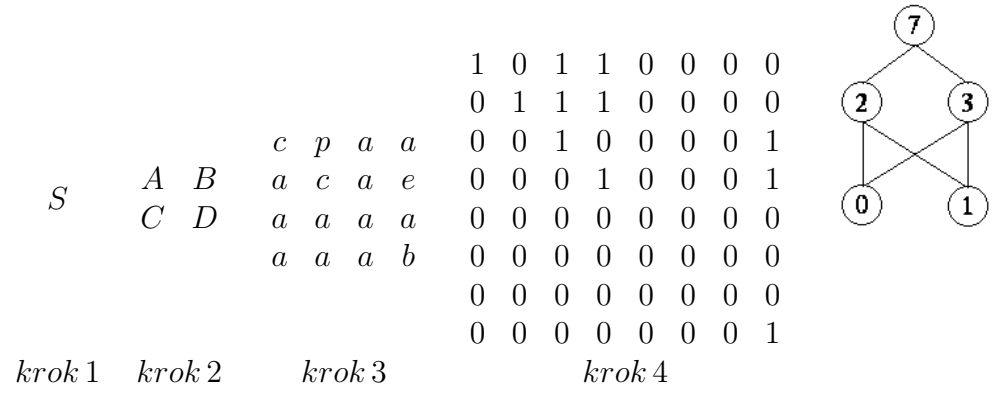
Kryterium zatrzymania algorytmu jest przekroczenie dostępnej liczby cykli. Końcowy wynik uznawany jest za najlepszy z możliwych w danej serii testów.

4.3 Gramatyki

Innym sposobem reprezentacji sieci neuronowej jest przedstawienie jej struktury za pomocą gramatyki [8]. Taka reprezentacja pozwala na znaczne zmniejszenie struktury genotypu opisującego sieć o znacznej wielkości. W zapisie tym wykorzystuje się gramatykę, której reguły produkcji generują macierz incydencji (por. 4.2).

Przedstawione na rysunku 9.6 kolejne kroki produkcji wynikają z reguł produkcji przedstawionych na rysunku 9.6. Symbole S, A, B, C, D oraz a, c, e, p są symbolami nieterminalnymi. Reguły produkcji zapisane za pomocą gramatyki przedstawiają dozwolone modyfikacje symboli nieterminalnych. Symbole a, c, e, p mogą zostać zamienione na macierze zawierające zera albo jedynki. W ten sposób określona zostaje struktura macierzy incydencji.

W omawianym przypadku chromosom zawiera jedynie symbol początkowy S oraz symbole nieterminalne A, B, C, D, tak jak pokazano na rys. 9.6. Krzyżowanie osobników



Rysunek 4.6: Przejście od gramatyki do reprezentacji macierzowej

$$S \rightarrow \begin{matrix} A & B \\ C & D \end{matrix}$$

$$A \rightarrow \begin{matrix} c & p \\ a & c \end{matrix} \quad B \rightarrow \begin{matrix} a & a \\ a & e \end{matrix} \quad C \rightarrow \begin{matrix} a & a \\ a & a \end{matrix} \quad D \rightarrow \begin{matrix} a & a \\ a & b \end{matrix}$$

$$a \rightarrow \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} \quad b \rightarrow \begin{matrix} 0 & 0 \\ 0 & 1 \end{matrix} \quad c \rightarrow \begin{matrix} 1 & 0 \\ 0 & 0 \end{matrix} \quad e \rightarrow \begin{matrix} 0 & 1 \\ 0 & 1 \end{matrix} \quad p \rightarrow \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}$$

Rysunek 4.7: Przykładowe reguły produkcji pozwalające na zapis struktury sieci neuronowej

polega na wymianie części genotypu, ale zawsze tak, aby na początku występował symbol S. Mutacja opiera się o zamianę pojedynczego symbolu na inny, dostępny z puli symboli nieterminalnych. Ocena każdego rozwiązania polega na przejściu od postaci zakodowanej, poprzez kolejne kroki produkcji, do sieci neuronowej. W etapie kolejnym sprawdza się czy sieć potrafi rozwiązać zadany problem.

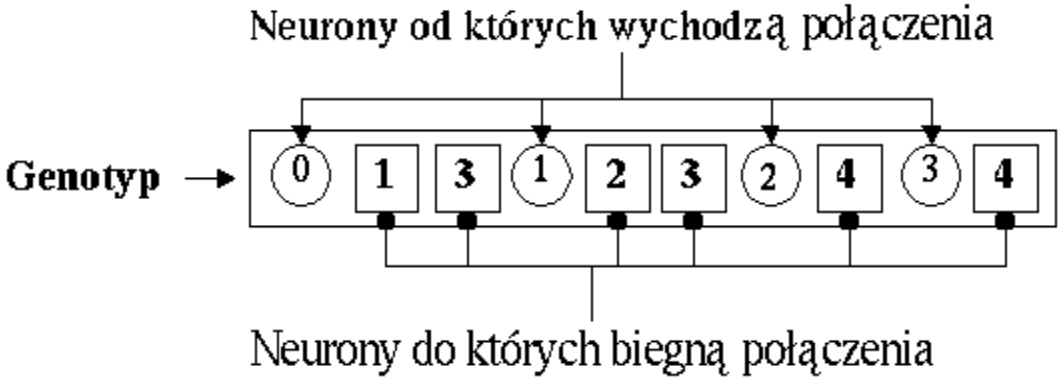
S	A	B	C	D

Rysunek 4.8: Chromosom zawierający reguły produkcji

4.4 Reprezentacja w postaci listy

W strukturach listowych informacje na temat połączeń między jednostkami zapisywane są w liście połączeń. Taki zapis genotypu pozwala na znaczne zmniejszenie jego rozmiarów. W przypadku reprezentacji macierzowej rozmiar genotypu był określony na podstawie liczby jednostek podniesionej do kwadratu lub $\frac{(liczba\,jednostek-1)\cdot liczba\,jednostek}{2}$ w przypadku przedstawienia genotypu tylko za pomocą jednej trójkątnej połówki macierzy. Dla genotypu w postaci listowej rozwiązanie wymaga znacznie mniej pozycji ze względu na fakt, że zapamiętane w niej są tylko połączenia aktywne. Reprezentacja listowa przykładu przedstawionego na rysunku 9.6 jest zaprezentowana na rysunku 9.6. Neurony obwiedzione

okręgami reprezentują jednostki, od których wychodzą połączenia do innych neuronów, zaś jednostki obwiedzione kwadratami reprezentują neurony docelowe.



Rysunek 4.9: Reprezentacja listowa sieci neuronowej

Oczywiście taki zapis wymaga wprowadzenia nowego operatora krzyżowania. Możemy zauważyć, że prosty podział osobnika może nie wystarczyć, ze względu na fakt, że osobniki mogą mieć różne długości. Operacja krzyżowania musiałaby polegać na wstępnej ocenie gdzie rozpoczyna się opis kolejnej jednostki (elementy zaznaczone), a następnie podzieleniu drugiego genotypu tak, aby punkt podziału przypadał na pierwszą jednostkę o indeksie większym od tej w pierwszym genotypie. Dodatkowo zmianie musiałby ulec operator mutacji. Jest to o tyle niezbędne, że operacja mutacji nie może polegać już tylko na prostej zamianie wartości danej pozycji, a musi również uwzględniać możliwość umieszczenia elementu na liście połączeń. W tej reprezentacji dodanie dodatkowego połączenia między jednostkami wymaga zmiany rozmiaru listy połączeń.

5 Równoległe algorytmy genetyczne

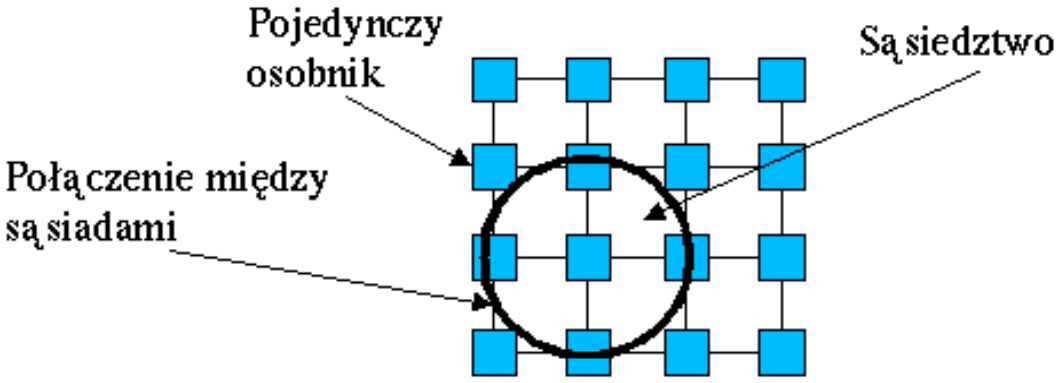
W rozdziale tym zostaną przedstawione metody zrównoleglania algorytmu genetycznego, które umożliwiają skrócenie czasu obliczeń kosztem zwiększenia mocy obliczeniowej.

Aplikację równoległą powinna cechować równoległa architektura. W niniejszym rozdziale w krótki sposób zostaną przedstawione najbardziej popularne architektury równoległe oraz szczegóły dotyczące rozwiązania zastosowanego w prezentowanej pracy.

5.1 Algorytm komórkowy

Komórkowy algorytm genetyczny cechuje się tym, że pojedyncze osobniki nie tworzą populacji w pełnym jej słowa znaczeniu. W tego rodzaju algorytmach wymiana materiału genetycznego następuje tylko w obrębie jego sąsiedztwa. Zwykle ustalane jest ono na podstawie promienia sąsiedztwa. Ze względu na fakt, że każda komórka posiada tylko jednego osobnika wymiana informacji w zasadzie sprowadza się do wymiany genotypów pomiędzy najbliższymi sąsiadami.

Działanie każdej komórki polega na obliczaniu wartości przystosowania osobnika, a następnie na rozesłaniu go do swoich sąsiadów i odebraniu od nich osobników już ocenionych.

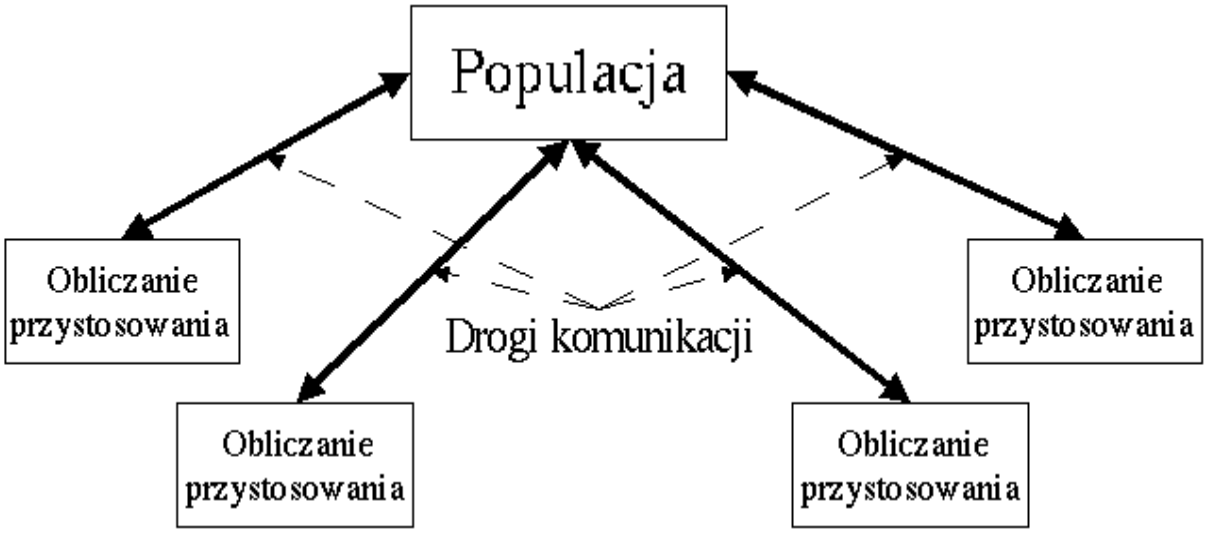


Rysunek 5.1: Populacja komórkowa

W przypadku implementacji niewątpliwym atutem wykorzystania takiej reprezentacji jest fakt, że obliczenia zostają znacznie rozproszone. Pozwala to na bardziej równomierne rozłożenie obliczeń. Dodatkowo w momencie komunikacji między poszczególnymi komórkami nie są one wzajemnie uzależnione w dużym stopniu. W zasadzie tylko sąsiedzi są zaangażowani w proces komunikacji, gdy tymczasem pozostałe procesy mogą swobodnie przetwarzać informacje.

5.2 Populacja globalna

Alternatywnym rozwiązaniem jest zastosowanie populacji globalnej zlecającej niejako obliczanie przystosowania poszczególnych osobników modułom do tego przeznaczonym. W rozwiązaniu takim mamy do czynienia z jedną tylko populacją oraz zbiorem specjalizowanych modułów. Oczywiście kanały komunikacyjne występują tylko między populacją, a modułami pomocniczymi.



Rysunek 5.2: Globalna populacja wraz z procesami oceniającymi osobniki

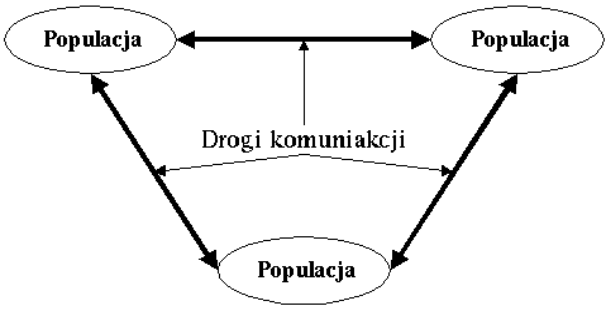
Rozwiązanie takie cechuje się znaczną ilością komunikacji od i do populacji. Dla problemu poszukiwania optymalnej struktury sieci neuronowej i tak nie ma to większego znaczenia, bo sam proces poszukiwania jest na tyle czasochłonny, że czas komunikacji nie ma większego wpływu na czas obliczeń.

Niewątpliwą zaletą tego modelu jest to, że w każdym momencie mamy dostęp do pełnej informacji na temat wszystkich rozwiązań. W przypadku algorytmu komórkowego, czy dalej opisywanego, wyspowego zdobycie takiej informacji wiąże się z dodatkowym obciążeniem procesów. Niestety tracimy możliwość podzielenia populacji na mniejsze odrębne populacje. Zaletę taką posiada kolejny model określany mianem modelu wyspowego.

5.3 Wyspowy algortym genetyczny

Wyspowy algorytm genetyczny cechuje się tym, że początkowa populacja zostaje równomiernie podzielona między populacje potomne. Poszukiwanie optymalnego rozwiązania wykonywane jest jednocześnie przez kilka równoległych populacji. Wprowadza się również element komunikacji, dzięki czemu najlepsze rozwiązania są rozprowadzane do innych populacji (rysunek 9.6). Działanie sekwencyjnego algorytmu zostaje zastąpione poprzez działanie współpracujących ze sobą rozłącznych, ale nie pozbawionych możliwości

komunikacji grup osobników. Dokładniejszy opis tego rozwiązania można znaleźć w rozdziale opisującym rozwiązanie zastosowane w niniejszej pracy.



Rysunek 5.3: Wyspowy algorytm równoległy

6 Szczegółowy opis równoległego przyjętego rozwiązania

Zrównoleglenie obliczeń wydaje się bardzo korzystnym rozwiązaniem. Możliwość jednoczesnej oceny osobników sprawia, że czas obliczeń zostaje znacznie skrócony. Algorytmy genetyczne komórkowy oraz z globalną populacją posiadały zasadniczą wadę, mianowicie znaczny stopień komunikacji między modułami. W przypadku algorytmu komórkowego mamy do czynienia z algorytmem o małym ziarnie przetwarzania. Oznacza to, że stosunek ilości przetwarzanej informacji do ilości komunikatów jest znaczny. W przypadku rozwiązania z globalną populacją sytuacja jest zupełnie inna, ale z kolei wszystkie operacje związane z populacją zostały scentralizowane.

Algortym wyspowy wydaje się być ciekawą alternatywą. Mamy tu do czynienia z rozłącznymi populacjami, a jednocześnie ziarno przetwarzania nadal pozostaje duże. Oznacza to, że stosunek ilości informacji przesyłanej do ilości informacji przetwarzanej jest mały. Jest to bardzo korzystne zjawisko. W takiej sytuacji algorytmy równoległe cechują się doskonałymi parametrami przetwarzania, a osiągana efektywność jest bardzo wysoka.

W modelu zastosowanym w niniejszej pracy komunikacja została rozwiązana w oparciu o promień sąsiedztwa równy jedności. Każda populacja wymienia informacje ze swoimi sąsiadami. Wymianie podlega część materiału genetycznego w pewnej z góry ustalonej ilości. Dodatkowo cykle komunikacyjne występują co określoną liczbę cykli populacyjnych. W ten sposób otrzymujemy wyraźny podział na część obliczeniową oraz komunikacyjną.

Działanie równoległego algorytmu genetycznego jest zbliżone do sekwencyjnego algorytmu genetycznego. Zasadnicza różnica polega na tym, że w algorytmie równoległym występuje jeszcze element komunikacji między wszystkimi procesami biorącymi udział w przetwarzaniu.

6.1 Metoda zrównoleglenia

Proces implementacji aplikacji oparty został o metodologię Fostera [16]. W oparciu o nią określona została optymalna architektura równoległa, którą okazał się być *mesh-wrap-around.*

W metodologii tej konieczny jest wybór rodzaju podziału zadania. Pierwszy podział, określany mianem podziału dziedzinowego, polega na podziale danych przeznaczonych do obliczeń. Każdy z procesów biorących udział w obliczeniach otrzymuje część danych a następnie je przetwarza. Dodatkowo należy określić sposób wymiany danych pomiędzy procesami. Drugi rodzaj podziału problemu, to podział funkcjonalny. Ten rodzaj podziału opiera się o zdefiniowanie różnych zadań w obrębie procesów biorących udział w przetwarzaniu. W trakcie projektowania rozpatrywana była architektura *Client-Server.* Założenia dotyczące tej architektury dotyczyły określenia takich funkcji procesów, aby część z nich odpowiedzialna była tylko za przetwarzanie danych, zaś inne procesy pełniły rolę pośredników w trakcie wymiany informacji. Model ten okazał się jednak zbyt skomplikowany, jeżeli chodzi o synchronizację komunikacji, co prowadzić mogło do bardzo

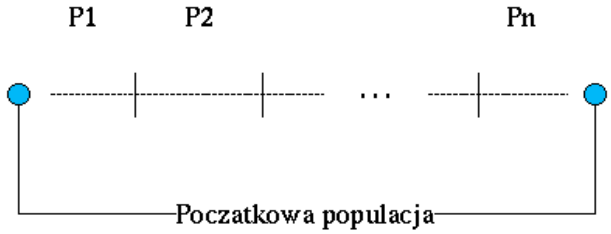
niekorzystnych sytuacji takich jak na przykład brak nowego materiału genetycznego w każdej z populacji.

Docelowo wybrany został podział dziedzinowy. Danymi podlegającymi podziałowi są osobniki z populacji początkowej. Dokładniej rzecz biorąc początkowa populacja dzielona jest na równe, mniejsze populacje współpracujące ze sobą w celu określenia optymalnego rozwiązania.

6.2 Podział danych

Jako dane rozumieć będziemy zbiór wszystkich osobników w populacji początkowej. W aplikacji równoległej rozprowadzane są one pomiędzy procesami w taki sposób, że każdy z procesów otrzymuje identyczną liczbę osobników. Tym samym początkowa populacja dzielona jest na równe podpopulacje. Przetwarzanie równoległe z komunikacją między populacjami cechuje się tym, że mimo posiadania n rozdzielnych populacji mamy tak naprawdę do czynienia z populacją, której poszczególne części prowadzą poszukiwania osobno. Wprowadzenie elementu wymiany danych polegało na okresowej wymianie najlepszych rozwiązań między populacjami. Miało to doprowadzić do wyznaczenia optymalnego rozwiązania pośród wszystkich procesów., a jednocześnie pobudzać do działania populacje słabsze.

Rysunek 9.6 przedstawia wykorzystany w implementacji podział danych. Przez P_i oznaczono kolejne populacje powstające w wyniku podziału populacji pierwotnej.



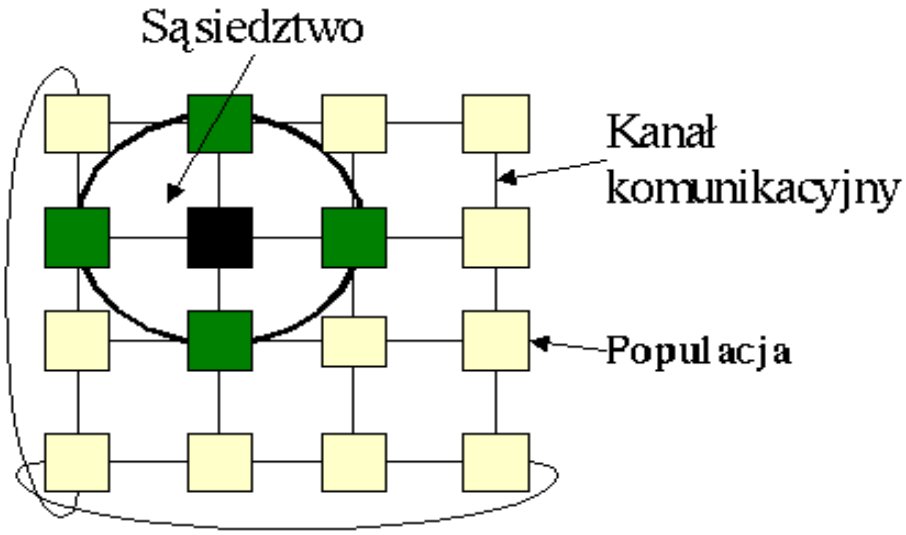
Rysunek 6.1: Podział populacji początkowej

Każdy z procesów przetwarza w zasadzie mniejszą rozmiarami populację. Trzeba pamiętać o tym, że dla zbyt małych rozmiarów populacji algorytm genetyczny może nie dawać zadowalających rezultatów. Z drugiej jednak strony zbyt duża populacja w przypadku algorytmu sekwencyjnego będzie prowadziła również do małej efektywności poszukiwania rozwiązania optymalnego. W związku z tymi niedogodnościami przyjęty rozmiar populacji początkowej powinien cechować się w miarę optymalną efektywnością zarówno dla początkowego rozmiaru populacji P jak i dla rozmiaru $\frac{P}{p}$; P zoznacza rozmiar populacji początkowej, zaś p liczbę procesów w algorytmie równoległym.

6.3 Komunikacja

Oczywiście sam podział danych jest niewystarczający. Koniecznym jest opracowanie architektury oraz modelu komunikacji między procesami.

W implementacji równoległego algorytmu genetycznego przyjęta została architektura *mesh-wrap-around* (rys. 9.6). Cechy tej architektury przemawiały za jej wyborem. Jasna i w pełni przewidywalna komunikacja między procesami. Klarowna struktura połączeń oraz jasny model teoretyczny obliczeń.



Rysunek 6.2: Architektura mesh-wrap-around

W modelu tym każdy z procesów posiada maksymalnie czterech sąsiadów. Dodatkowo nawet w przypadku gdy mamy do czynienia z mniejszą ich ilością możemy założyć, że dalej mamy do czynienia z czterema sąsiadami. Po prostu traktujemy te same procesy jako różne. Siatka jest zawinięta (rys. 9.6) co sprawia, że połączenia dostępne są również na jej brzegach, a indeksy sąsiadów wyznaczane są modulo szerokość i modulo wysokość siatki. Na rysunku 9.6 zaznaczony został również proces (najciemniejszy kolor) wraz z jego sąsiedztwem (trochę jaśniejsze kwadraty). Jak możemy zauważyć maksymalna liczba sąsiadów wynosi cztery.

Wszelka komunikacja odbywa się wewnątrz sąsiedztwa danego procesu. W ten sposób pokrywamy komunikacją całą przestrzeń siatki. Dodatkowo zapewniamy sobie, że każdy z procesów otrzyma infomację od najdalej położonego procesu nie wcześniej niż po $\sqrt{n} + \sqrt{m}$ krokach komunikacyjnych, jeżeli przyjmiemy, że n wyznacza szerokość siatki, a m wyznacza jej wysokość. Dzięki temu mamy pewność, że genotyp nie zostanie zbyt szybko wymieszany, a w konsekwencji faktycznie będziemy mieli do czynienia z kilkoma rozłącznymi populacjami.

Komunikacja między procesami występuje co zadaną liczbę cykli algorytmu genetycznego. Najlepsze rozwiązania są przesyłane do pozostałych procesów, ale tylko w obrębie sąsiedztwa. Rozsyłanie danych realizowane jest w czterech krokach:

- rozesłanie w kolumnach w prawo,
- rozesłanie w kolumnach w lewo,

- rozesłanie w rzędach w górę,
- rozesłanie w rzędach w dół.

Po zakończeniu cyklu komunikacyjnego materiał genetyczny zostaje rozrzucony po siatce dzięki czemu kolejne populacje mogą czerpać z dotychczasowych wyników innych populacji.

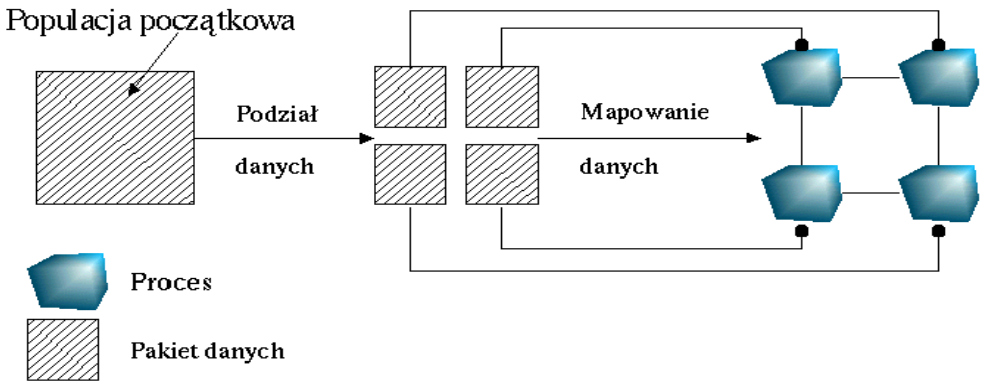
6.4 Aglomeracja obliczeń

Za wyborem architektury *mesh-wrap-around* przemawia dodakowo możliwość aglomeracji danych. Zabieg ten pozwala na zmniejszenie dostępnej liczby jednostek przetwarzających poprzez zwielokrotnienie danych przeznaczonych do obliczeń. Otrzymujemy wtedy wprawdzie procesy o znacznej ilości danych, ale zyskujemy na zmniejszeniu liczby komunikatów przesyłanych przez sieć. Oczywiście nie pozostaje to bez cech ujemnych. Zwykle wydłużona zostaje długość komunikatu oraz następuje szybsze ujednolicenie populacji. Niemniej jednak potencjalnie mamy możliwość wykorzystania mniejszej liczby jednostek liczących pozostając nadal przy tym samym modelu teoretycznym.

6.5 Początkowy podział danych

W pierwszej fazie działania programu równoległego przeznaczona do obliczeń populacja zostaje równomiernie podzielona. Początkowy podział danych odbywa się na podstawie równomiernego podziału początkowej populacji na mniejsze takie, że każda zawiera początkową liczbę osobników podzieloną przez liczbę procesów biorących udział w obliczeniach. Generowanie populacji początkowej, jej następny podział oraz rozesłanie realizowane jest przez proces „0“ obrany arbitralnie za proces inicjujący obliczenia. W przypadku, gdy realizowany jest algorytm sekwencyjny proces „0“ nie rozsyła początkowej populacji tylko wykonuje samodzielne obliczenia.

Proces mapowania danych przedstawiony został na rysunku 9.6.



Rysunek 6.3: Rozesłanie początkowej populacji do wszystkich procesów

6.6 Model teoretyczny obliczeń

Dla przedstawionego modelu wyspowego opracowany został model teoretyczny pozwalający na ocenę rozwiązania. Podstawowym wyznacznikiem jakości zrównoleglenia algorytmu jest jego efektywność. Przez efektywność E rozumiemy stosunek czasu wykonania algorytmu sekwencyjnego do czasu wykonania algorytmu równoległego zwielokrotnionego o liczbę wykorzystanych procesów. Przyjmując następujące oznaczenia:

E - szukana efektywność,
T_s - czas przetwarzania algorytmu sekwencyjnego,
T_p - czas przetwarzania pojedynczego procesu równoległego,
p - liczba równolegle pracujących procesorów,

możemy wyrazić efektywność za pomocą wzoru (6.1):

$$E = \frac{T_s}{T_p} \tag{6.1}$$

Jako czas sekwencyjny T_s przyjęto (rów. 6.2)

$$T_s = t_c \cdot p \cdot P \tag{6.2}$$

gdzie:

T_s - szukany czas działania programu sekwencyjnego,
t_c - czas przetwarzania pojedynczego osobnika,
P - liczba osobników w populacji.

Dodatkowo zakłada się, że algorytm sekwencyjny jest optymalnym algortymem dla zadanego problemu. Przyjmując oznaczenia:

T_p - sumaryczny czas działania równoległego algorytmu,
t^i_{comp} - czas przeznaczony na obliczenia przez i-ty procesor,
t^i_{comm} - czas przeznaczony na komunikację przez i-ty procesor,
t^i_{idle} - czas bezczynności i-tego procesu,
T_{comm} - uśredniony czas komunikacji wszystkich procesów,
T_{comp} - uśredniony czas przetwarzania wszystkich procesów,

czas pracy T_p równoległego algorytmu wyrażamy wzorem (6.3)

$$T_p = \frac{1}{p} \cdot \sum_{i=0}^{p} t^i_{comp} + t^i_{comm} + t^i_{idle} \tag{6.3}$$

gdzie T_{comm} wyrażone jest równaniem (6.4)

$$T_{comm} = 2 \cdot (t_s + t_w \cdot w) \cdot p \cdot 4 \tag{6.4}$$

Jeżeli pominiemy czas bezczyności T_{idle}, to czas przetwarzania T_p wyrazić możemy za pomocą wzoru (6.5)

$$T_p = \frac{T_{comm} + T_{comp}}{p} \tag{6.5}$$

gdzie:

t_s - czas startu konieczny na zainicjowanie komunikacji,
t_w - czas przesłania jednostki danych,
w -wielkość danych przeznaczonych do wysłania,
p - liczba wszystkich procesów.

Ze względu na fakt, że w każdym cyklu komunikacyjnym procesy wysyłają oraz odbierają informację od swoich sąsiadów, ilość przesyłanej informacji jest równa dwukrotnej ilości wysyłanych komunikatów.

Uśredniony czas przetwarzania T_{comp} procesów wyrażony jest wzorem (6.6)

$$T_{comp} = t_c \cdot p \cdot P \tag{6.6}$$

gdzie:

t_c - czas konieczny na obliczenie przystosowania pojedynczego osobnika,
p - liczba procesów,
P - liczba osobników w każdej populacji.

Czas T_{idle} określamy mianem czasu bezczynności. Jest czasem, w którym aktywność procesu jest równa zeru. W zasadzie okresy bezczynności wystąpić mogą tylko w dwóch przypadkach. Pierwszy z nich to ten, gdy proces zakończył obliczenia, podczas gdy inne procesy jeszcze pracują. Drugi przypadek to taki, w którym proces oczekuje na informację od innego procesu. Jest to związane z cyklem komunikacyjnym.

Uwzględniając powyższe uwagi otrzymujemy wyrażenie pozwalające obliczyć czas T_p

$$T_p = \frac{2 \cdot (t_s + t_w w) \cdot 4p + t_c \cdot p \cdot P}{p} \tag{6.7}$$

$$T_p = 2 \cdot (t_s + t_w w) \cdot 4 + t_c \cdot \frac{p \cdot P}{p} \tag{6.8}$$

Ostatecznie:

$$E = \frac{t_c \cdot p \cdot P}{\left[2 \cdot (t_s + t_w w) \cdot 4 + t_c \cdot \frac{p \cdot P}{p}\right] \cdot p} \tag{6.9}$$

$$E = \frac{t_c \cdot p \cdot P}{2 \cdot (t_c + t_w w) \cdot 4p + t_c \cdot p \cdot P} \tag{6.10}$$

Czasy t_w oraz t_s powinny być odczytane z literatury fachowej. Niemniej jednak ze względu na fakt, że czas t_s jest zwykle niewielki można go w zasadzie pominąć. Czas t_w można wyznaczyć doświadczalnie. W zasadzie czas komunikacji jako taki można rozpatrywać jako całość i nie uwzględniać poszczególnych składowych.

Ze wzoru jasno wynika, że im mniejszy czas komunikacji, tym lepsza efektywność algorytmu (bliższa jedności). Ze względu na fakt, że algorytm poszukujący optymalnej struktury sieci jest algorytmem o dużym ziarnie przetwarzania, stosunek czasu przetważania do czasu komunikacji jest bardzo wysoki. W konsekwencji efektywność algorytmu również jest bardzo wysoka.

Rozważania teoretyczne potwierdzają obserwacje. Faktycznie uzyskane w trakcie doświadczeń czasy przetwarzania, komunikacji oraz bezczynności wskazują na bardzo dobrą jakość zrównoleglenia. Wynika ona z doskonałych wręcz warunków, czyli małej ilości przesyłanej informacji.

7 Testy wydajnościowe rozwiązania

Celem ustalenia jakości zrównoleglenia algorytmu oraz oceny przydatności tego typu narzędzi do tworzenia struktury sieci neuronowych zostały przeprowadzone testy wydajnościowe. Badaniom podlegały:

- zależność czasu działania aplikacji od liczby epok algorytmu genetycznego,
- zależność czasu działania aplikacji od liczby osobników w populacji,
- zależność czasu wykonania aplikacji od liczby procesorów jednocześnie współpracujących.

Dodatkowo badane były zależności między przystosowaniem średnim, maksymalnym i minimalnym. Testy te pozwoliły nie tylko na ocenę samej aplikacji, lecz również modelu wyspowego algorytmu genetycznego.

W ramach testów system opracowywał struktury sieci neuronowych dla czterech problemów. Pierwszy z testów polegał na określeniu struktury sieci neuronowej rozpoznającej parzystość liczby zapisanej binarnie. Kolejny test polegał na generowaniu sieci neuronowej zdolnej do konwersji liczb zapisanych binarnie na skalę termometrową. Kolejne z zadań postawionych przed systemem polegało na określeniu struktury sieci neuronowej realizującej funkcję XOR. Ostatni test miał na celu stwierdzenie czy system wygeneruje poprawną strukturę dla problemu rozpoznawania irysów.

Wydaje się, że system potrafi dosyć dobrze określać strukturę sieci neuronowej. Wprawdzie pojawiały się w trakcie testów rozwiązania nieoptymalne niemniej jednak było one nadal lepsze od stosowanych sieci warstwowych z połączeniami pełnymi między warstwami.

Dalsze części rozdziału prezentują szczegółowo przebieg testów, wyniki oraz wnioski dotyczące otrzymanych rezultatów.

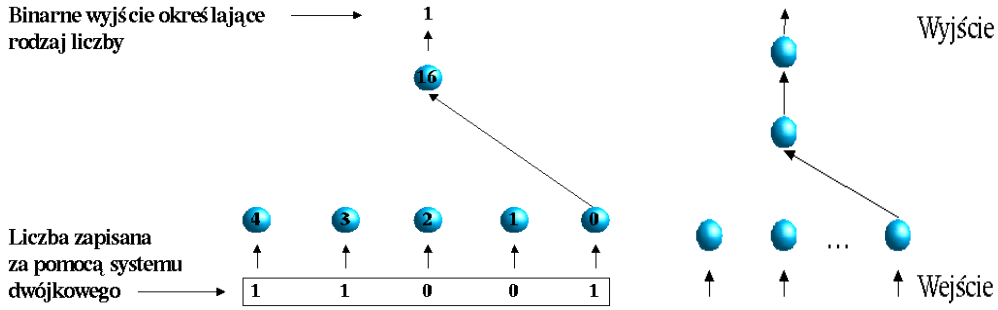
7.1 Problem określenia parzystości liczby

Problem określania parzystości liczby jest w zasadzie banalny z punktu widzenia matematycznego. Jeżeli liczba jest podzielna bez reszty przez liczbę 2 to ta pierwsza jest liczbą parzystą. W przeciwnym wypadku tak nie jest. Dla liczb zapisanych w systemie binarnym powyższe stwierdzenie sprowadza się do określenia czy najmłodszy bit jest ustawiony na 1 czy też nie. Przed algorytmem genetycznym postawione zostało zadanie znalezienia optymalnej struktury sieci neuronowej to znaczy takiej, która składałaby się tylko z jednego połączenia. Powinno ono występować właśnie pomiędzy wejściem reprezentującym najmłodszy bit liczby, a jej wyjściem.

Sieć uczona była za pomocą zbioru zawierającego pary, wektora wejściowego o pięciu pozycjach oraz binarnego wyjścia określającego czy liczba reprezentowana przez wektor jest parzysta czy też nie. Każda z pozycji wektora odpowiadała pozycji w zapisie dwójkowym liczby. Dodatkowo nie wszystkie liczby mieszczące się w zakresie $< 0; 2^5 - 1 >$ wykorzystane

były jako wzorce uczące. Część z nich została wykorzystana jako zbiór testowy, mający na celu określenie zdolności uogólniania sieci neuronowej.

Testy prowadzone były dla sieci o maksymalnym rozmiarze 10, 16 oraz 20 neuronów. Sieć z powodzeniem potrafiła wyznaczyć optymalne rozwiązanie składające się z dokładnie jednego połączenia. Zaprezentowane przez aplikację rozwiązanie miało postać jak na rysunku 9.6. Pierwsze rozwiązanie jest rozwiązaniem optymalnym. Drugie nie jest wprawdzie optymalne, ale jest poprawne.



Rysunek 7.1: Dwa różne rozwiązanie problemu parzystości

Bardzo interesującym wydaje się fakt, że sieć potrafiła znaleźć struktury poprawne, ale mniej optymalne. Algorytm potrafił generować struktury, w których sygnał propagowany był przez sieć w sposób jak na schemacie drugim. Można zauważyć, że neuron pośredniczący w wymianie sygnału jest całkowicie zbędny i nic do obliczeń nie wnosi. Można go swobodnie usunąć. Możemy więc zauważyć, że struktura jest prawie optymalna. Fakt, że struktury takie się pojawiały może świadczyć o niezbyt trafnym doborze wag w funkcji oceny algorytmu genetycznego.

7.1.1 Funkcja fitness

Przypomnieć należy, że funkcja oceny wyrażona jest za pomocą sumy (7.1):

$$fit = w_0 f_0 + w_2 f_2 + ... + w_{16} f_{16} \tag{7.1}$$

W zależności od tego, z jaką siecią mamy do czynienia: błędną, poprawną lub zdolną się uczyć, zestaw wag przyjmuje różne wartości. Pozwala to na proste zróżnicowanie wartości przystosowania sieci.

Wektory wag funkcji oceny, dla problemu rozpoznawania parzystości liczb, miały postać jak we wzorze (7.2)

$$\begin{array}{ll} w_1 =< 0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,-1 > & dla\, niepoprawnych \\ w_2 =< 0,0,0,1,0,0,1,0,0,1,0,1,0,1,1,0,0 > & dla\, nienauczonych \\ w_3 =< 0,0,0,10,0,0,10,0,0,10,0,10,0,10,10,0,0 > & dla\, nauczonych. \end{array} \tag{7.2}$$

Wzór (7.2) wymaga dodatkowego komentarza. Dla struktur nienauczonych pod uwagę brane były następujące czynniki funkcji fitness: $\frac{conGen}{conMax}$, $conGen$, $conMax$. Dzięki temu

promowane były sieci posiadające większą liczbę połączeń. Takie rozwiązanie rokowało nadzieje na powstanie w kolejnych pokoleniach struktur zdolnych do nauczenia się na podstawie wzorców uczących. Struktury nienauczone, ale poprawne pod względem struktury, wykorzystywały z kolei czynniki: $\frac{learnCnt}{learnMax}$, $\frac{accTestCnt}{testCnt}$, $\frac{1}{aConGen}$, $\frac{1}{conGen}$, $\frac{1}{learnCnt}$, $accTestCnt$. Człony $\frac{learnCnt}{learnMax}$oraz $\frac{1}{learnCnt}$ miały za zadanie promować struktury, które uczyły się krócej. Człony $\frac{accTestCnt}{testCnt}$ i $accTestCnt$ z kolei odpowiedzialne były za promowanie struktur lepiej generalizujących. Wyrażone to było stosunkiem wzorców rozpoznanych do liczby wszystkich wzorców testujących. Człon $accTestCnt$ służył wzmocnieniu oddziaływania jakości generalizacji na wartość funkcji oceny. Im więcej wzorców zostało rozpoznanych tym większą wartość miał ten człon. Wreszcie człony $\frac{1}{aConGen}$ oraz $\frac{1}{conGen}$ miały na celu minimalizację połączeń w obrębie sieci. Dla sieci poprawnych i nauczonych wagi funkcji były takie same jak dla struktur błędnie nauczonych. Różnica polegała na tym, że wartości wag były znacznie większe.

Ciekawym jest, że proces poszukiwania postępował dopiero wtedy, gdy różnica między wagami funkcji fitness wynosiła co najmniej rząd wielkości. W przeciwnym wypadku proces poszukiwania optymalnej struktury przebiegał mało efektywnie. Fakt, że wszystkie wagi wektora w_3 mają wartość 10 nie ma większego znaczenia. Ważnym jest, że sieci nauczone były promowane znacznie bardziej.

7.1.2 Proces uczenia

Proces uczenia sieci opierał się o zestaw wzorców uczących i testujących. Wzorce uczące pokrywały równomiernie przestrzeń wszystkich wzorców. Wzorce testujące zostały wybrane spośród zbioru uczącego, który w ten sposób został pomniejszony. Wzorce zawierały pary reprezentujące wejście oraz oczekiwane wyjście sieci. Klasyfikacji podlegały liczby z zakresu $< 0; 2^5 - 1 >$ stąd wektor wejściowy miał pięć pozycji. Wektor wyjściowy składał się z jednej pozycji. Wartość jeden na wyjściu oznaczała liczbę nieparzystą z kolei wartość zero liczbę parzystą. Szczegóły reprezentacji wejść i wyjść przedstawione zostały w tabeli 9.6.

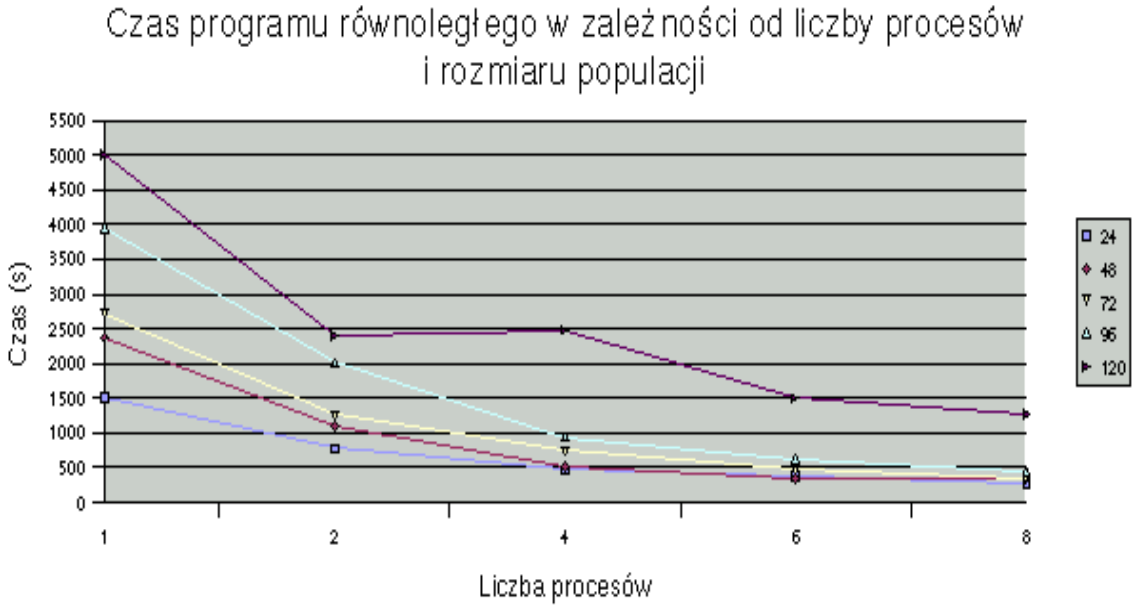
Tabela 7.1: Zestaw wejść i wyjść dla problemu rozpoznawania parzystości liczb.

wejście					wyjście
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
.....					
1	1	1	1	0	0
1	1	1	1	1	1

Pierwsza kolumna tabeli 9.6 zawiera wejścia, kolejne liczby zapisane w systemie binarnym poczynając od zera. Druga zawiera numer klasy, do której należy zaklasyfikować daną liczbę.

7.1.3 Obliczenia równoległe

Celem sprawdzenia jakości równoległych obliczeń przeprowadzony został szereg testów mających na celu stwierdzenie, czy równoległa aplikacja cechuje się parametrami takimi jak w modelu teoretycznym. Ze względu na znaczny czas obliczeń testy zostały przeprowadzone zaledwie czterokrotnie dla każdego rozmiaru populacji i dla każdej liczby procesorów. Aplikacja była uruchamiana na 1, 2, 4, 6 oraz 8 procesorach. Rozmiar populacji wahał się od 24 do 120 osobników w populacji początkowej. Za każdym razem następował transfer od jednego do 3 najlepszych osobników z każdej populacji. Liczba ta zależała od rozmiaru populacji początkowej. Ze względu na wygodę przyjęto, że w miarę zwiększania rozmiaru populacji początkowej bedzie on zwiększany o 24 osobniki. Liczba ta bowiem dzieli się bez reszty przez każdą z liczb procesorów wykorzystanych w obliczeniach. Pomiary wykazały, że istotnie czas pracy aplikacji równoległej jest znacznie mniejszy niż czas pracy algorytmu sekwencyjnego (wykres na rysunku 9.6).

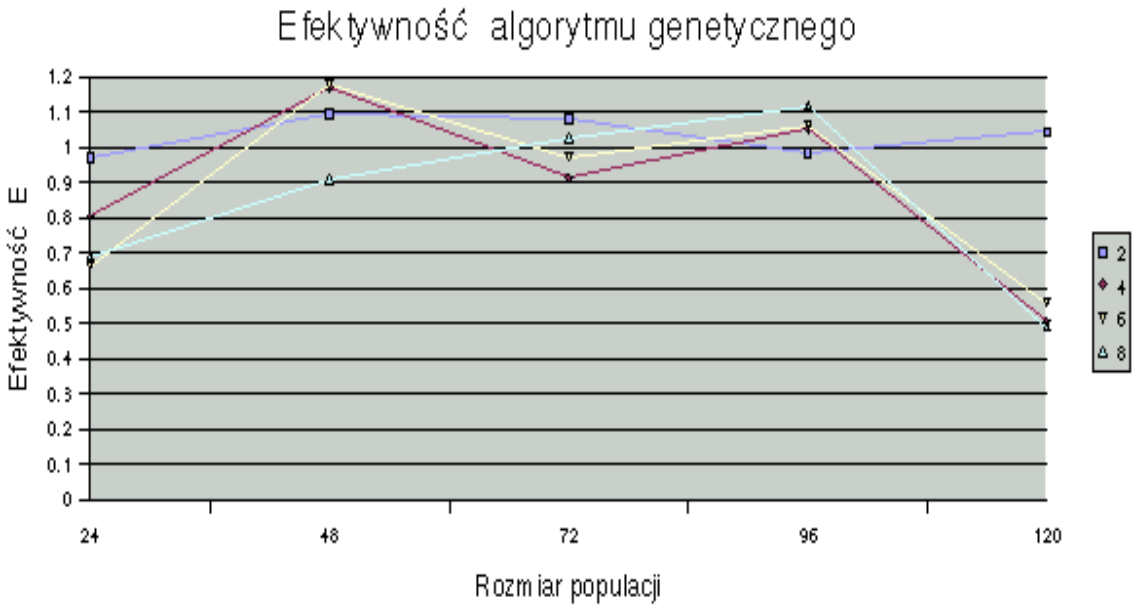


Rysunek 7.2: Czas pracy aplikacji równoległej dla problemu parzystości liczb. Legenda przedstawia liczbę osobników w populacji.

Na wykresie możemy zauważyć, że w trakcie pomiarów musiała wystąpić znaczna aktywność systemu. Mogło to być spowodowane wykorzystaniem komputera przez innych użytkowników. W takiej sytuacji zasoby dzielone są równomiernie, co powoduje znaczne spadki wydajności aplikacji równoległej. W modelu teoretycznym zakłada się, że mamy do czynienia z komputerem idealnym, w którym czas aktywności systemu równy jest zeru. Niestety w rzeczywistości jest to niemożliwe. Aplikacje innych użytkowników również zabierają cenny czas procesora.

Kolejnym krokiem było wyznaczenie efektywności aplikacji równoległej. Zgodnie z modelem teoretycznym miała się ona cechować doskonałymi parametrami. Niestety

praktyka i teoria różnią się znacznie. Szczegóły przedstawia rysunek 9.6.



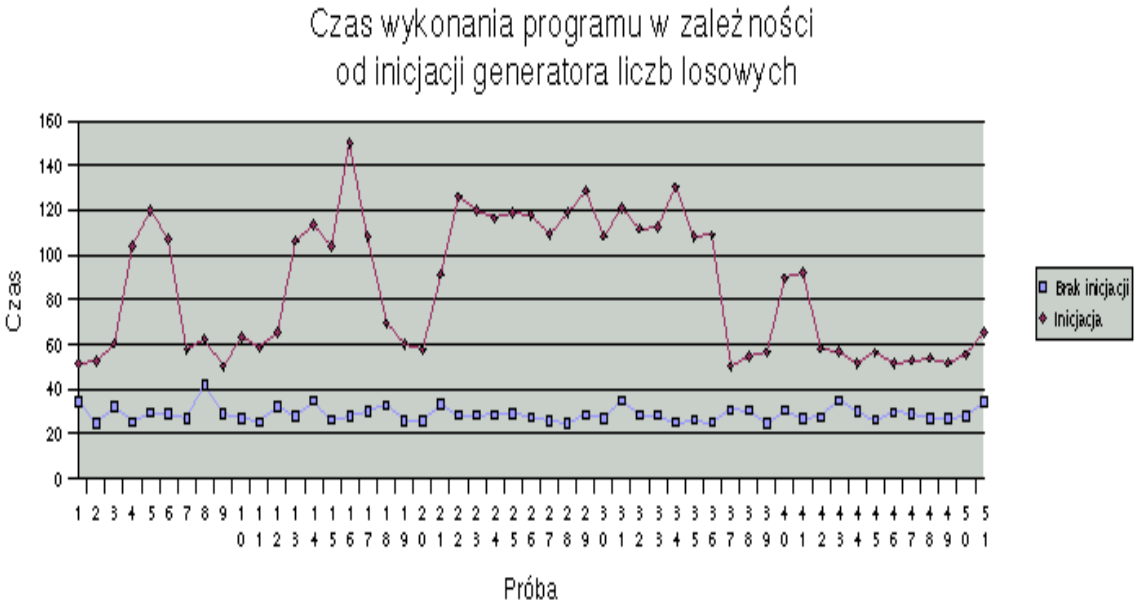
Rysunek 7.3: Efektywność algorytmu równoległego. Legenda przedstawia liczbę procesorów.

Wykres efektywności powinien asymptotycznie zbliżać się do wykresu funkcji $f(x) = 1$. Jak możemy zauważyć na rysunku 9.6 tak jednak nie jest. Jedynie wykres dla 2 oraz 8 procesorów można uznać za w miarę zgodny z teoretycznym. Należy zwrócić uwagę na fakt, że wszystkie pomiary czasu były uśreniane. Wynikało to z faktu, że moc obliczeniowa komputerów wykorzystanych do obliczeń znacznie się równiła.

Wyniki takie nie są zadowalające. Fakt, że czas przetwarzania jest mniejszy w przypadku aplikacji równoległej nie powinien dziwić. Nie jest jednak dobrym zjawiskiem, gdy efektywność drastycznie maleje, chodzi tu o czsas przetwarzania dla rozmiaru popluacji równego 120 (rys. 9.6). Można podejrzewać, że rozmiar problemu (chodzi tu o liczbę 120 osobników) jest liczbą powyżej optymalnego rozmiaru problemu. Oznacza to najczęściej, że po przekroczeniu tej liczby wzrasta znacznie aktywność systemu. Jest to głównie spowodowane dużą aktywnością plików typu *swap*.

Wyniki takie mogą niepokoić. W celu wyjaśnienia problemu przeprowadzone zostały dodatkowe testy. Miały one na celu stwierdzenie jak wygląda przebieg pracy algorytmu sekwencyjnego, a dokładniej, ile wynosi czas przetwarzania algorytmu sekwencyjnego. Zostało przeprowadzonych 100 dodatkowych testów dla mniejszej liczby cykli populacji. Zmniejszenie liczby cykli miało na celu zmniejszenie czasu testów do rozsądnej wielkości. Wyniki są zaskakujące (rys. 9.6)

Warto zwrócić uwagę na fakt, że czas pracy programu, w którym ziarno generatora liczb losowych było inicjowane różnymi wartościami, waha się od 50 do 150 sekund. Aplikacja, w której ziarno generatora było zawsze inicjowane liczbą 0 wygląda całkiem inaczej. Ten fakt



Rysunek 7.4: Zależność czasu przetwarzania od początkowego stanu sieci neuronowej. Stan ten zależy od generowanych liczb losowych.

przesądził o tym, że prawdziwie dokładny pomiar efektywności jest prawie niemożliwy. Wyjściem byłoby przeprowadzenie ogromnej liczby testów dla pełnej gamy parametrów, zarówno algorytmu genetycznego, jak i sieci neuronowej. Należy zwrócić uwagę na fakt, że mogą wystąpić w pracy algorytmu równoległego super przyspieszenia. Zjawisko takie wystąpi wtedy, gdy czas wykonania aplikacji równoległej będzie znacznie krótszy od czasu wykonania aplikacji sekwencyjnej.

Nadal jednak wydaje się być pocieszającym fakt, że czas przetwarzania w przypadku aplikacji równoległej jest krótszy. Pojawia się jednak pytanie, skąd takie różnice w czasie wykonania aplikacji sekwencyjnej. Proces uczenia sieci neuronowej w znacznym stopniu opiera się o element losowości. Wygenerowanie początkowych wartości wag z dala od wartości optymalnych, to znaczy takich dla których wzorce zostaną zapamiętane, oznacza długi okres uczenia. Jeżeli z kolei wagi zostaną wylosowane blisko optimum czas uczenia będzie znacznie krótszy. Szerzej o tym problemie będzie traktował rozdział dotyczący rozwiązania problemu XOR.

7.2 Konwersja liczb zapisanych binarnie na skalę termometrową

Biorąc pod uwagę czas przetwarzania oraz fakt, że aplikacja działała bardzo nierównomiernie testy przeprowadzane były dla mniejszej liczby cykli populacji. Testy miały jedynie na celu stwierdzić, jakie cechy posiada algorytm równoległy. Poszukiwanie optymalnej struktury sieci neuronowej dużo lepiej przebiega na średnim komputerze domowej klasy. Wadą tego rozwiązania jest brak możliwości oceny zachowań algorytmu równoległego. Te cechy były badane w środowisku równoległym. Po stwierdzeniu, że dany

zestaw parametrów może doprowadzić do generacji struktury optymalnej problem był redukowany, a następnie uruchamiany w środowisku równoległym.

Testy zostały przeprowadzone w takim samym zakresie jak dla poprzednio opisywanego problemu. Badane były przyspieszenia algorytmu dla różnych wielkości procesorów biorących udział w przetwarzaniu. Dodatkowo badana była efektywność algorytmu dla różnych rozmiarów danych.

7.2.1 Funkcja fitness

Dla problemu przyjęty został ten sam zestaw wag, który był prezentowany w przypadku określania parzystości (7.2) liczby. Zestaw ten był całkowicie wystarczający, jeżeli chodzi o określenie minimalnej struktury sieci neuronowej dla problemu konwersji liczb.

7.2.2 Proces uczenia

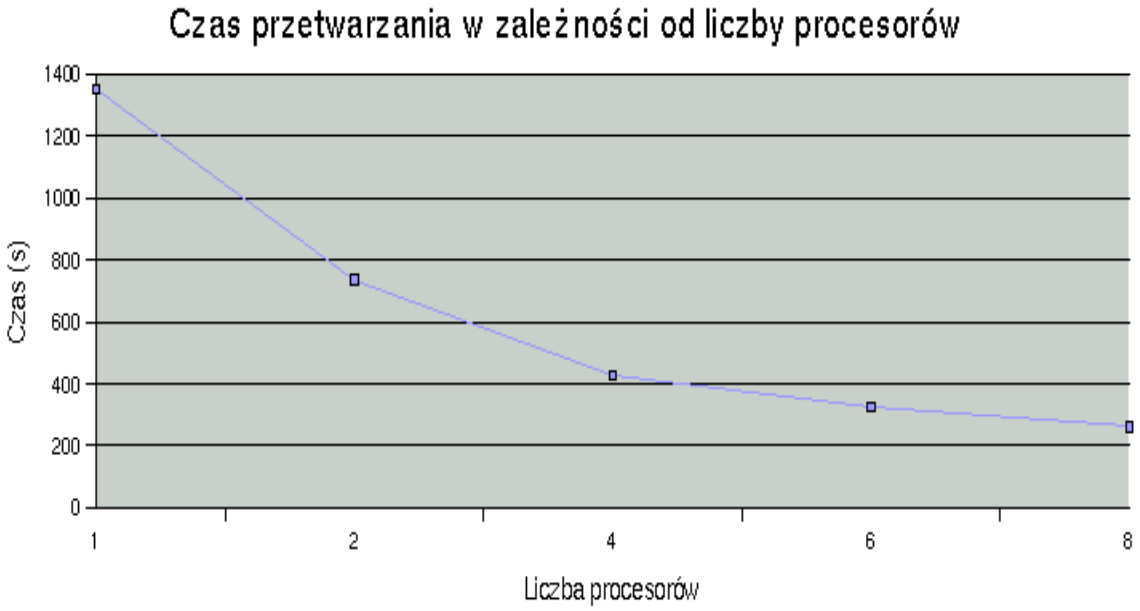
Proces uczenia opierał się o zbiór wzorców uczących. Wejścia sieci reprezentowały kolejne liczby binarne, z kolei wyjścia ich odpowiedniki w skali termometrowej. W procesie uczenia wykorzystane były wzorce jak w tabeli 9.6.

Tabela 7.2: Wzorce uczące dla problemu konwersji liczb.

wejście			wyjście						
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1	0
0	1	1	0	0	0	0	1	0	0
1	0	0	0	0	0	1	0	0	0
1	0	1	0	0	1	0	0	0	0
1	1	0	0	1	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0

7.2.3 Obliczenia równoległe

Obliczenia równoległe zostały przeprowadzone na dwa sposoby. Pierwszy, w środowisku rzeczywistym, polegał na kolejnym uruchomieniu aplikacji równoległej i sekwencyjnej w środowisku heterogenicznym. Ze względu na duży czas przetwarzania wykonane zostały pomiary dla stałego rozmiaru populacji oraz różnej liczby procesorów. Na rysunku 9.6 możemy zauważyć, że czas przetwarzania maleje wraz ze zwiększaniem liczby procesorów biorących udział w przetwarzaniu.



Rysunek 7.5: Czas obliczeń aplikacji równoległej dla problemu konwersji liczb z postaci binarnej na termometrową.

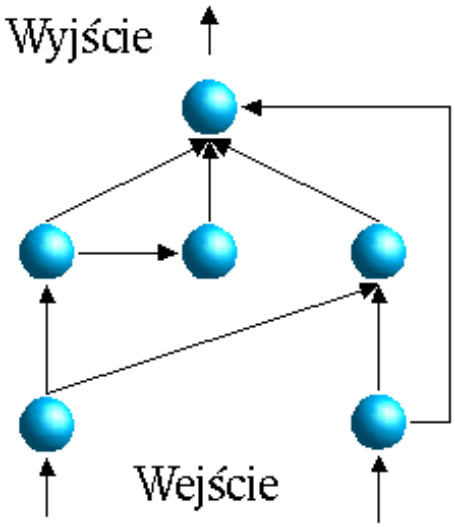
7.3 Problem XOR

Kolejnym problemem stawianym przed aplikacją było określenie struktury sieci neuronowej pozwalającej na rozwiązanie problemu funkcji XOR. Funkcja ta cechuje się bardzo ciekawymi własnościami. Do poprawnej klasyfikacji tego problemu wymaganych jest co najmniej 3 lub pięć neuronów. Zależy to od tego, w jaki sposób będzie propagowany sygnał do sieci. Jeżeli wejście sieci jest równomiernie rozprowadzane do wszystkich jednostek wejściowych, to potrzeba trzech neuronów oraz sześciu połączeń, aby problem został poprawnie rozwiązany.

Najlepsza ze znalezionych sieci wprawdzie nie była optymalna, ale do takiego stanu niewiele jej brakowało (rys. 9.6).

7.3.1 Funkcja fitness

Funkcja fitness przyjęła niemalże identyczną postać, jak w przypadku problemu znajdowania struktury sieci rozpoznającej parzystość liczb. Zasadnicza różnica polegała na nieznacznym zwiększeniu wag odpowiadających za czas uczenia sieci. Problem XOR jest o tyle ciekawym, z punktu widzenia pomiaru czasu, że silnie zależy od początkowych wartości wag. Czas uczenia może się wahać w granicach od 2000 epok uczących, z zastosowaniem współczynnika momentum, do około 50000, bez tego współczynnika. Stosowany był oczywiście algorytm ze współczynnikiem momentum, niemniej jednak czas uczenia nadal wykazywał duże wahania. Fakt ten spowodował, że na sieć został nałożony wymóg jak najkrótszego czasu uczenia, co zapewniało, że nawet jego pesymistyczny przypadek był



Rysunek 7.6: Najlepsza ze znalezionych sieci - realizująca funkcję XOR

nadal zadowalający.

W tym celu zostały zmodyfikowane nieznacznie wagi odpowiedzialne za ważenie dwóch parametrów funkcji fitness. Parametrami tymi były $\frac{learnCnt}{learnMax}$ oraz $\frac{1}{learnCnt}$. Obydwa promują sieci szybko uczące się. Zwiększenie współczynnika polegało na przemnożeniu go dwa razy. Ośiągnięte to zostało poprzez zmodyfikowanie odpowiednich wag we wzorze (7.2). Pozwoliło to na zachowanie własności funkcji fitness. Szerzej o problemie doboru wag traktuje rozdział opisujący problem rozpoznawania irysów.

7.3.2 Proces uczenia

Proces uczenia oparty był o zbiór uczący zaprezentowany w tabeli 9.6. Każda z pozycji składała się z pary wejście oraz oczekiwane wyjście. Wejściem do sieci była para liczb binarnych, zaś wyjściem wynik funkcji XOR działającej na tych liczbach. Dane uczące miały postać jak w tabeli 9.6.

Tabela 7.3: Wzorce uczące dla problemu XOR.

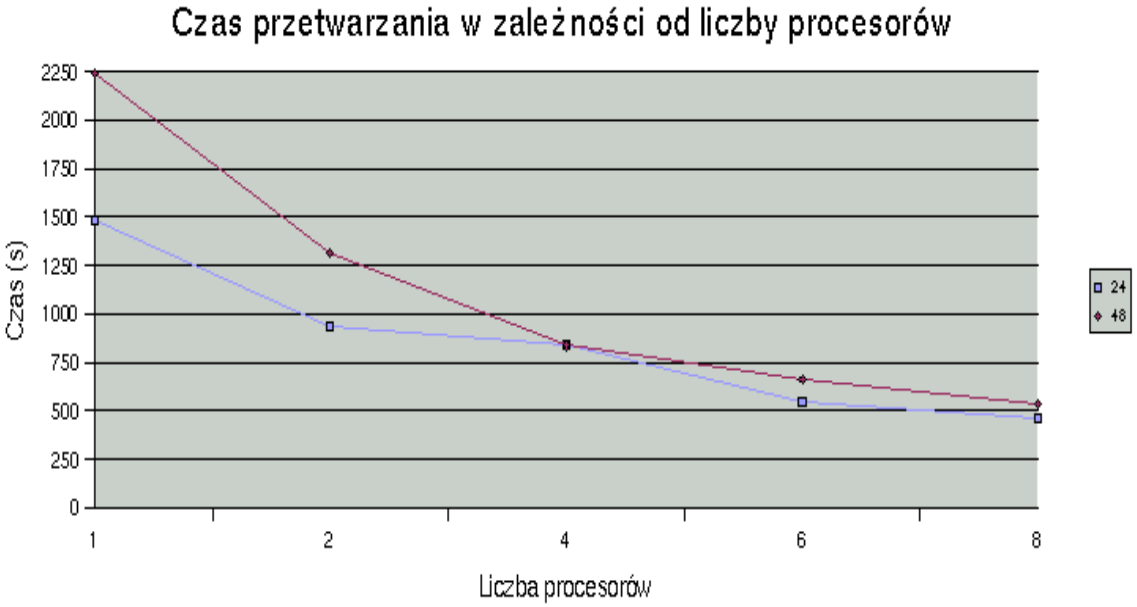
wejście		wyjście
0	0	0
0	1	1
1	0	1
1	1	1

Proces uczenia polega na takim doborze wag sieci, aby wzorce klasyfikowane były do jednej z dwu klas. Problem pojawiający się przy funkcji XOR to brak liniowej rozłączności klas przynależności. Powoduje on znaczny wzrost czasu obliczeń.

7.3.3 Obliczenia równoległe

W celu przeprowadzenia obliczeń równoległych w trakcie testów wykorzystano inny rodzaj reprezentacji wejść oraz wyjść niż ten przedstawiony w tabeli 9.6. Okazało się, że wykorzystanie reprezentacji bipolarnej danych znacznie skraca proces uczenia sieci.

W trakcie obliczeń wykorzystano zestaw 1, 2, 4, 6 oraz 8 procesorów. Środowisko przetwarzania było identyczne jak dla poprzednio opisanych testów. Wyniki (rys. 9.6) jednoznacznie wskazują, że przyspieszenia w algorytmie występują. Znaczny spadek czasu przetwarzania wskazuje, że wykorzystanie architektury równoległej może poprawić wydajność algorytmu.



Rysunek 7.7: Czas przetwarzania algorytmu równoległego dla problemu XOR. Legenda zawiera informacje o rozmiarze populacji.

7.4 Rozpoznawanie irysów

Rozpoznawanie irysów polega na klasyfikacji kwiatów do jednego z trzech gatunków (Iris Setosa, Iris Versicolour, Iris Virginica), na podstawie czterech cech opisujących strukturę kwiatu. Cechy te to:

- szerokość płatków,
- długość płatków,
- szerokość działek kielicha,
- długość działek kielicha.

Sieci neuronowe są w stanie bardzo dobrze klasyfikować prezentowane wzorce. Dodatkowo potrafią w bardzo dużym stopniu generalizować.

7.4.1 Funkcja fitness

Dla tego problemu musiały być zmodyfikowane wagi funkcji fitness. Dla zestawu identycznego jak ten w przypadku badania parzystości funkcji okazało się, że rozmiar sieci jest znaczny. Koniecznym było zwiększenie nacisku na parametr odpowiadający za wielkość struktury sieci. W końcowym efekcie wartości wag we wzorze (7.2) odpowiadające za ważenie członów odpowiedzialnych za rozmiar struktury sieci ($\frac{1}{aConGen}$ oraz $\frac{1}{conGen}$) zostały podwojone.

Ze względu na dużą czasochłonność pracy aplikacji współczynniki dobierane były tylko częściowo doświadczalnie. W czasie wstępnych testów stwierdzono, że zbytnia różnica między wagami może prowadzić do zmniejszenia czasu uczenia kosztem rozmiaru sieci oraz odwrotnie. Nieznaczne zwiększenie wag odpowiedzialnych za rozmiar sieci prowadzi natomiast do oczekiwanego efektu przy optymalnym czasie uczenia i odwrotnie. Dodatkowo zaobserwowano, że zbyt duże wartości wag prowadzą do błyskawicznej degradacji populacji. Powoduje to błyskawiczne rozprzestrzenienie się osobników podobnego typu, co z kolei powstrzymuje proces poszukiwania optimum funkcji.

7.4.2 Proces uczenia

Proces uczenia odbywał się w taki sam sposób, jak dla wszystkich innych przypadków. Prezentowane wzorce pozwalały na taką modyfikację wag, aby rozpoznawanie przebiegało poprawnie. Wzorce uczące zostały dobrane w taki sposób, aby w miarę dokładnie pokryć całą przestrzeń danych. Wykorzystywane były dwa zestawy wzorców. Zestaw większy, składający się ze 150 wzorców, uruchamiany był na domowym komputerze autora. Ze względu na dużą złożoność czasową problemu został on ograniczony do 20 wzorców w przypadku uruchamiania aplikacji równoległej. Zbiór testowy składał się z kolejnych odpowiednio 20 lub 5 wzorców nie wchodzących w skład zestawu uczącego.

Wzorce uczące stanowiły pary <wejście, wyjście>, gdzie wejściem był wektor o czterech składowych. Każda odpowiadała za jedną cechę kwiatu. Wyjściem był wektor składający się z trzech pozycji. Każda pozycja odpowiadała za jeden gatunek. Szczegóły przedstawia tabela 9.6.

Uczone sieci wykazywały dosyć dobre zdolności uogólniania. Dodatkowo cechowały się niewielką liczbą połączeń wewnątrz sieci.

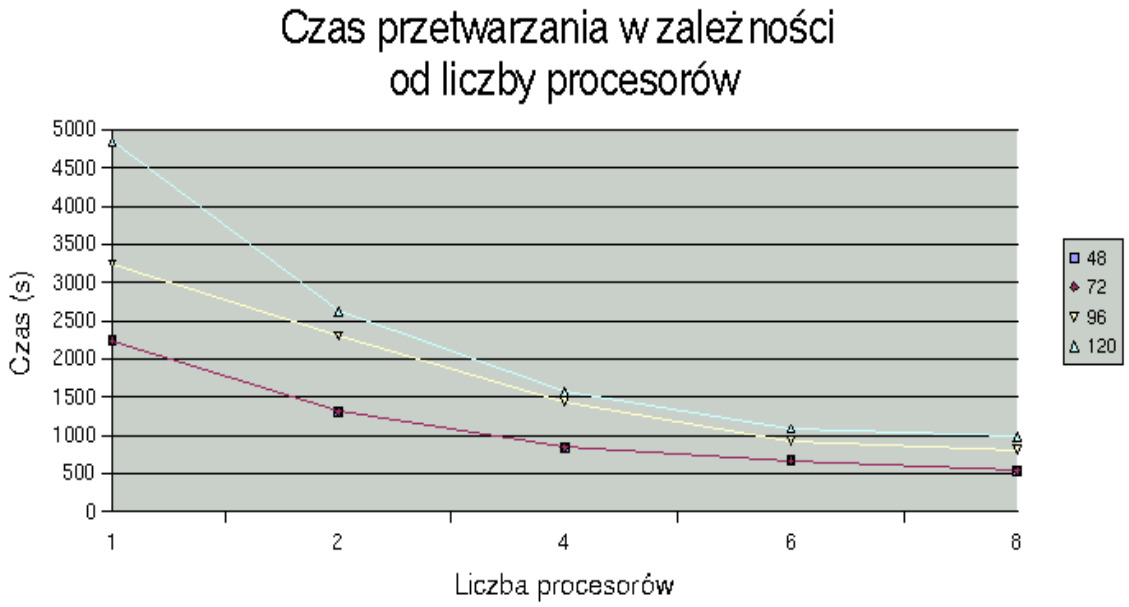
7.4.3 Obliczenia równoległe

Ze względu na długi czas obliczeń testy przeprowadzone zostały na dwa różne sposoby. Pierwszy z nich polegał na uruchomieniu aplikacji w heterogenicznym środowisku równoległym, drugi na sprawdzeniu wydajności algorytmu w środowisku z jednym procesorem.

Tabela 7.4: Wzorce uczące dla problemu rozpoznawania irysów.

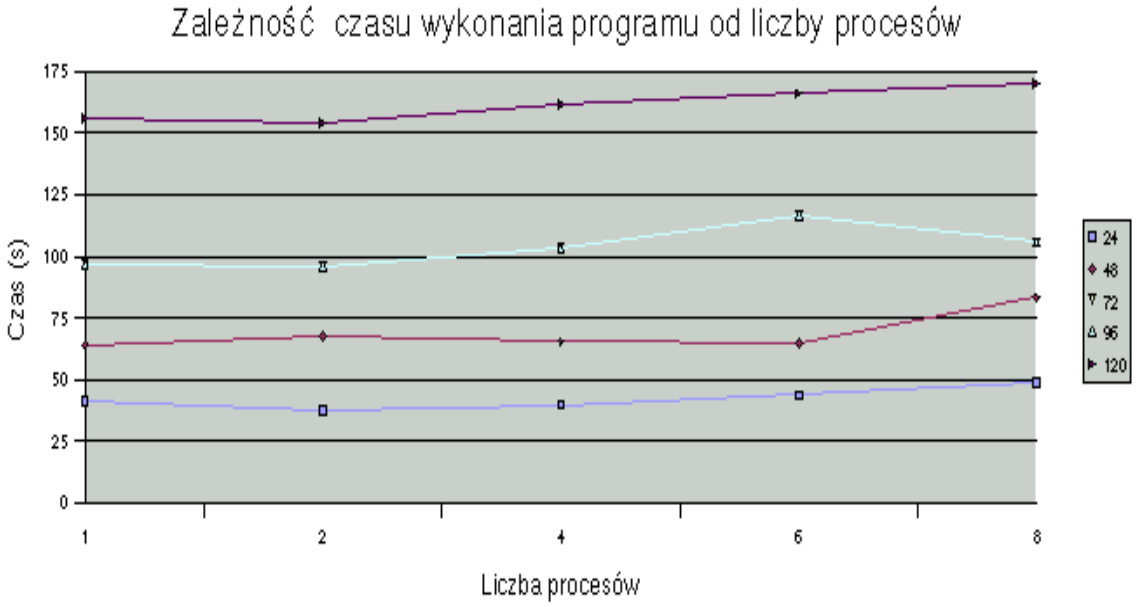
wejście				wyjście		
dł. kielicha	szer. kielicha	dł. płatka	szer. płatka			
6.1	2.9	4.7	1.4	0	0	1
6.3	3.4	5.6	2.4	0	1	0
5	3.6	1.4	0.2	1	0	0
....				....		
5.2	3.5	1.5	0.2	1	0	0

Testy wykonane w środowisku równoległym potwierdzają ogólną tendencję algorytmu (rys. 9.6). Czas przetwarzania jest oczywiście większy dla populacji złożonych z większej liczby osobników niemniej jednak zwiększenie liczby procesorów zawsze skutkuje skróceniem czasu obliczeń.



Rysunek 7.8: Czas przetwarzania algorytmu równoległego dla problemu rozpoznawania irysów. Legenda przedstawia liczbę osobników w populacji.

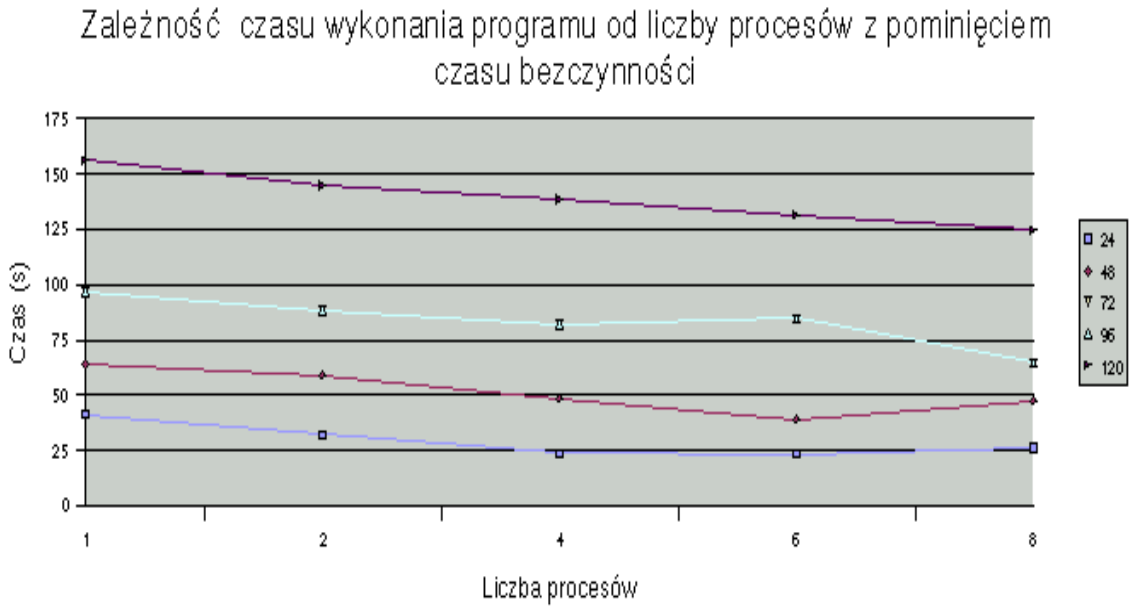
Kolejny test miał na celu zmierzenie wydajności algorytmu równoległego uruchamianego na maszynie z jednym procesorem. Należało stwierdzić, czy wydajność takiej aplikacji wzrośnie. Fakt, że przetwarzanie przebiegało na maszynie jednoprocesorowej sprawił, że przyspieszenia nie są imponujące. Dodatkowo należy zwrócić uwagę na fakt, że obciążenie procesora dwoma procesami prowadzi do zmniejszenia czasu przetwarzania. Zwiększenie liczby procesów biorących udział w przetwarzaniu prowadzi natomiast do zbytniego obciążenia procesora. W konsekwencji prowadzi to do znacznego wzrostu czasu obliczeń (rys. 9.6).



Rysunek 7.9: Czas programu równoległego uruchamianego na pojedynczym procesorze.

Całkiem odmiennie ma się sytuacja w przypadku odrzucenia czasów bezczynności. Fakt, że czasy te są znaczne może być spowodowany między innymi tym, że przy dużej liczbie procesów wzrasta aktywność programu pośredniczącego w wymianie informacji między poszczególnymi procesami, co za tym idzie, czas oczekiwania na kolejne dane wzrasta. Niemniej jednak sam czas przetwarzania maleje. Oznacza to, że jeżeli chodzi o ilość informacji koniecznej do przetworzenia nie jest ona ilością powodującą spadek wydajności procesora.

Dodatkowo na duże czasy oczekiwania może mieć wpływ fakt, że w programie po realizacji komunikacji każdorazowo następuje okres oczekiwania na pozostałe procesy. Zabieg ten miał na celu stworzenie równych szans różnorodnym maszynom (HP i SUN). W przypadku pojedynczego procesora usunięcie tych fragmentów programu mogłoby znacznie poprawić wydajność (rys. 9.6).



Rysunek 7.10: Czas przetwarzania programu równoległego bez uwzględnienia czasu bezczynności.

Wydaje się, że liczba dwóch procesów stanowi najbardziej optymalne rozwiązanie. Wzrost czasu bezczyności jest w tej sytuacji niwelowany przez znaczne zmniejszenie czasu przetwarzania. W pozostałych przypadkach możemy mieć do czynienia z sytuacją, w której uruchomienie procesu równoległego doprowadzi do znacznego wzrostu czasu przetwarzania.

Należy zwrócić uwagę na fakt, że wyniki te w odróżnieniu od wcześniej przedstawionych dotyczą przetwarzania na jednym procesorze. Wyniki te nie mogą być porównywane bezpośrednio z tymi, które zostały uzyskane w poprzednich testach. Potwiedzają one natomiast, że uruchamianie aplikacji równoległej w środowisku z jednym procesorem również skutkuje zwiększeniem wydajności obliczeń.

8 Podsumowanie

Na podstawie testów można stwierdzić, że proces poszukiwania optymalnej sieci neuronowej za pomocą równoległego algorytmu genetycznego napotyka na wiele trudności. Silne związki z procesem generowania liczb losowych powodują, że czasy wykonania tej samej aplikacji mogą różnić się diametralnie.

W trakcie trwania testów równoległych okazało się, że środowisko równoległe bardzo zależy od warunków, w jakich pracuje system operacyjny. Część testów wyraźnie pokazuje brak stabilności środowiska heterogenicznego. Tego rodzaju objawy można jedynie niwelować wykorzystując silne komputery wieloprocesorowe.

Kolejnym elementem znacznie wpływającym na zmniejszenie efektywności równoległego algorytmu genetycznego jest algorytm propagacji wstecznej błędu wykorzystywany do uczenia sieci neuronowej. Jest to bardzo czasochłonny proces. Dodatkowo algorytm ten silnie zależy od stanu początkowego, który jest generowany w sposób losowy.

Wydaje się, że jedynym rozsądnym rozwiązaniem w tej sytuacji jest zastosowanie takiej reprezentacji sieci neuronowej, która nie wymagałaby wielokrotnego uczenia sieci neuronowej. Reprezentacja taka powinna cechować się tym, że zapisane w niej będą wartości wag sieci neuronowej.

Przeprowadzone doświadczenia wykazały wprawdzie przyspieszenia aplikacji, ale są one bardzo nierówne. W związku z tym nie można jednoznacznie stwierdzić, jakie są ich wartości. To, że aplikacja wykonuje się w krótszym czasie jest faktem. Nie można jednak z pewnością stwierdzić, jakie są wartości tego przyspieszenia.

Aby postawić bardziej ogólne tezy konieczne byłoby przeprowadzenie ogromnej liczby testów. Testy takie pozwoliłyby na dokładne określenie wpływu parametrów uczenia sieci oraz parametrów algorytmu genetycznego na czas działania aplikacji równoległej. Należałoby również bardzo dokładnie zbadać zależność czasu działania algorytmu propagacji wstecznej od stanu początkowego, co za tym idzie od wartości liczb losowych.

9 Opis aplikacji

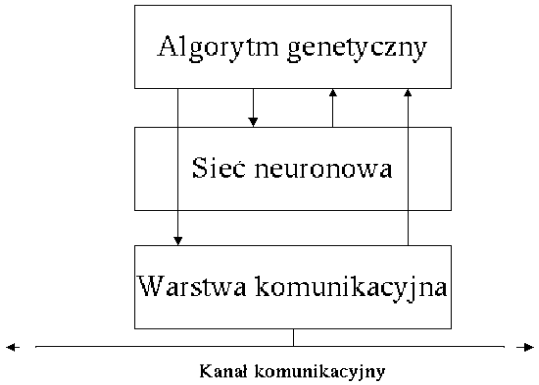
Celem wykonania testów niezbędnych do oceny równoległego modelu została zaprojektowana, a następnie zaimplementowana aplikacja realizująca algorytm równoległy. Wyróżnić w niej można trzy podstawowe warstwy. Warstwę komunikacji, warstwę sieci neuronowych oraz warstwę algorytmu genetycznego. Ze względu na fakt, że istotą problemu było badanie własności równoległego algorytmu genetycznego warstwa odpowiedzialna za opracowanie populacji jest niejako nadrzędną w stosunku do pozostałych dwóch. Oczywiście każda z warstw uległa dalszemu rozdrobnieniu, co pozwala na elastyczne modyfikacje zarówno struktury genotypu, realizacji sieci neuronowej czy zmiany modelu komunikacji.

9.1 Warstwy modelu

Podział na trzy rozdzielone warstwy ma mocne uzasadnienie. Taka realizacja pozwala na wprowadzenie niezbędnych tylko relacji między odpowiednimi warstwami. Dzięki zastosowaniu takiego podziału poszczególne warstwy komunikują się między sobą tylko wtedy, gdy faktycznie zachodzi taka potrzeba. Rysunek 9.6 przedstawia ten właśnie podział. Wyraźnie możemy zauważyć, że komunikacja występuje tylko między modułami algorytmu genetycznego i sieci neuronowej oraz między modułem algorytmu genetycznego i modułem komunikacyjnym. Podział taki podyktowany jest względami praktycznymi. Wymiana informacji pomiędzy warstwą komunikacyjną i siecią neuronową jest całkowicie zbędna.

Kanał komunikacyjny między warstwą algorytmu genetycznego i sieci neuronowej zawiera dodatkowy element, adapter, pozwalający na poprawne dekodowanie genotypu poszczególnych osobników. W ten sposób możemy całkowicie abstrakcyjnie patrzeć na reprezentację osobników. Dla modułu sieci neuronowej ważne są tylko informacje na temat liczby jednostek, liczby połączeń między nimi, wartości wag przypisanych do każdego z połączeń. Dodatkowo warstwa ta musi posiadać informacje na temat liczby wejść oraz wyjść sieci. Wszystkie parametery dostarczane są po uprzednim przetworzeniu genotypu przez adapter. Na rysunku nie został on zaznaczony, gdyż nie jest on oddzielną warstwą a raczej częścią algorytmu genetycznego. Niemniej jednak musimy mieć świadomość, że każda nowa reprezentacja sieci neuronowej wymaga ponownej implementacji adaptera.

Opisany model znalazł swoje odwzorowanie w implementacji aplikacji. Poszczególne moduły zostały opracowane oddzielnie a następnie spojone dodatkowymi modułami realizującymi komunikację między warstwami. Dodatkowo należy zauważyć, że wprowadzona musiała być jeszcze komunikacja między procesami równoległymi. Zaznaczony na rysunku 9.6 kanał komunikacyjny symbolizuje połączenie między poszczególnymi „wyspami“. Przesyłane za jego pomocą osobniki mogą być następnie adaptowane przez każdą z populacji.

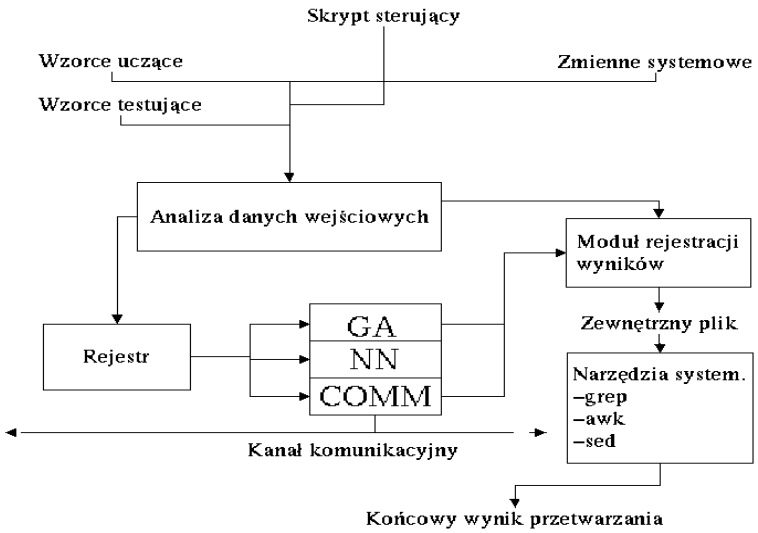


Rysunek 9.1: Podział problemu na mniejsze moduły funkcjonalne

9.2 Realizacja

Przedstawione wcześniej rozwiązanie zostało wzbogacone o elementy niezbędne dla aplikacji cechującej się dobrą funkcjonalnością. Niezbędnym było wprowadzenie metody komunikacji między użytkownikiem a programem. Funkcjonalność ta realizowana jest za pomocą czterech, przygotowywanych przez użytkownika parametrów. Są nimi:

- skrypty sterujące,
- wzorce uczące,
- wzorce testujące,
- zmienne systemowe.



Rysunek 9.2: Szczegółowy schemat aplikacji

W skład skryptów sterujących wchodzą dwa uzupełniające się zestawy parametrów. Pierwszy, przygotowywany przez użytkownika, drugi zawierający domyślne parametry przygotowany przez autora pracy. Parametry domyślne mają na celu zwolnienie przyszłego użytkownika z wielokrotnego przygotowywania zestawu parametrów zwykle pokrywających się w znacznym stopniu.

Wzorce uczące zawierają zestaw wzorców prezentowanych sieci w trakcie procesu uczenia. Na ich podstawie modyfikowane są wagi sieci neuronowej. Zdolność generalizacji testowana jest za pomocą zestawu wzorców testujących. Obydwa zestawy przygotowywane są przez użytkownika. Ich postać zostanie omówiona w późniejszej części opracowania.

Przedstawiony na rysunku 9.6 *Rejestr* zawiera niezbędne dla poprawnej pracy programu informacje. W dużej mierze są to odpowiednio zinterpretowane, a następnie przetworzone informacje zawarte w skrypcie startowym. Program rozpoczyna swoje działanie właśnie od tej czynności. Po wstępnej analizie informacji zawartych w skryptach następuje ustalenie architektury równoległej. Wszystkie spośród uruchomionych procesów zostają przeindeksowane. Po tej fazie następuje określenie struktury abstrakcyjnej siatki procesów.

Po poprawnym ustaleniu struktury komunikacyjnej dane przeznaczone do obliczeń zostają równomiernie podzielone, a następnie rozesłane do wszystkich procesów biorących udział w obliczeniach. Od tej pory każdy proces przetwarza przypisany mu zestaw danych. Fakt, że algorytm jest algorytmem wyspowym wymaga istnienia komunikacji między procesami przetwarzającymi dane. W aplikacji równoległej jest to realizowane za pomocą procedury globalnej komunikacji. Co określony interwał czasu następuje wymiana informacji między wszystkimi populacjami. Wymianie podlegają tylko najlepsze rozwiązania. W ten sposób uzyskujemy równomierne rozprzestrzenianie się materiału genetycznego w obrębie siatki procesów. Aby zapobiec zbyt szybkiej wymianie materiału genetycznego przyjmuje się, że wymiana następuje co określoną liczbę epok algorytmu genetycznego oraz ustala się liczbę osobników podlegających wymianie na taką, by była znacznie mniejsza niż rozmiar populacji. Dodatkowo należy nadmienić, że w trakcie wymiany osobników z populacji usuwane są rozwiązania najsłabsze. Ich miejsce zajmują nowo przybyłe osobniki z innych populacji.

9.3 Współdziałanie modułów

Każdy z modułów wykonuje tylko te akcje, za które jest bezpośrednio odpowiedzialny. Poprawny przebieg algorytmu zapewnia *Rejestr* zawierający niezbędne informacje. Dzięki *Rejestrowi,* istnieje możliwość scentralizowania wszystkich parametrów oraz późniejszy szybki i łatwy do nich dostęp. Współpraca modułów jest niejako koordynowana właśnie przez moduł *Rejestru.* Pośrednio wpływa on na wszystkie podejmowane przez program akcje, ponieważ zawiera warunki ich wykonania. Należy zwrócić uwagę na fakt, że Rejestr jest przeznaczony tylko do odczytu. Zapewnia to stałe wartości parametrów w trakcie przebiegu algorytmu. Początkowy zapis Rejestru odbywa się we wstępnej fazie pracy programu. Analiza skryptów startowych pozwala na poprawne wypełnienie bazy Rejestru niezbędnymi informacjami. Jeżeli użytkownik nie zdefiniuje wartości parametrów zostanie poinformowany o tym, a wartości te zostaną przyjęte wedle zasad zapisanych w skrypcie z

parametrami domyślnymi programu.

Kolejnym modułem wspomagającym pracę programu jest moduł rejestracji wyników. Pozwala on na zebranie wszystkich niezbędnych do obliczeń parametrów pracy aplikacji równoległej. Nie uwzględnia się czasu globalnego, co za tym idzie obliczane mogą być tylko czasy działania poszczególnych procesów równoległych. W trakcie wykonywania obliczeń rejestrowane są rozmaite aktywności aplikacji. W ich skład wchodzą:

- czas przetwarzania,
- czas bezczynności,
- czas komunikacji,
- wartości przystosowania osobników,
- genotypy osobników,
- informacje o anomaliach wewnątrz programu,
- informacje o błędnym działaniu programu,
- informacje o stanie komunikacji między procesami (głównie o błędnym jej działaniu).

Późniejsze przetworzenie tych informacji pozwala na określenie własności algorytmu. Możemy wyznaczyć stosunek komunikacji do przetwarzania, czy też efektywność algorytmu.

Celem lepszego wyznaczenia czasu przetwarzania operacje dyskowe zostały zminimalizowane. Rejestracja aktywności programu przebiega w obrębie przydzielonej pamięci operacyjnej. W przypadku stwierdzenia braku miejsca na kolejne rejestracje następuje zapis bazy na dysk. Czas zapisu oraz wszystkich towarzyszących mu operacji jest również rejestrowany. W trakcie testów modułu rejestracji stwierdzono, że zapis pojedynczej aktywności aplikacji jest pomijalny, zaś łączny czas zapisu rejestru na dysk, przy średnim czasie trwania równym 0.1 sekundy, również może zostać pominięty. Biorąc pod uwagę, że w trakcie pracy dokonywanych było około 10 zapisów rejestru (co daje łącznie około sekundy na operacje dyskowe), zaś czas pracy aplikacji wahał się od kilkuset do kilkunastu tysięcy sekund; czas dokonywania zapisu na dysk możemy pominąć.

Ze względu na fakt, że system UNIX dysponuje znacznym zapleczem programów wspomagających pracę programisty zaniechano wstępnego przetwarzania rejestru aktywności programu. Wszelkie analizy odbywają się za pomocą dostępnych w tym środowisku aplikacji takich jak: grep, sed, awk oraz języka skryptowego powłoki csh. Przetworzone w ten sposób dane były analizowane za pomocą arkusza kalkulacyjnego.

9.4 Sterowanie aplikacją

Do sterowania aplikacją wykorzystuje się skrypty startowe. Rozwiązanie takie zostało przyjęte z dwóch zasadniczych powodów. Pierwszym jest możliwość wielokrotnego

uruchamiania aplikacji po uprzednim przygotowaniu zestawu skryptów. Dzięki temu użytkownik nie musi bezpośrednio sterować przebiegiem testów.

Przykładowo. Przeprowadzenie testu badającego czas działania aplikacji w zależności od liczby osobników sprowadza się do wcześniejszego opracowania odpowiednich skryptów, a następnie uruchomienia testu. Od tego momentu użytkownik nie musi już korygować czy modyfikować parametrów aplikacji. Dodatkowym wsparciem dla użytkownika są zaprojektowane skrypty pozwalające na wygodne uruchamianie najbardziej typowych testów.

Kolejnym atutem takiego rozwiązania jest jego łatwa rozbudowa. Dzięki zastosowaniu jednolitego Rejestru z parametrami aplikacji nawet w przypadku późniejszej rozbudowy systemu nie ma obaw o kolejne, dodatkowe parametry systemu.

9.5 Budowa skryptów startowych

Skrypt startowy został zaprojektowany w taki sposób, by możliwe było przekazywanie znacznej ilości informacji do programu przy minimalizacji procedur do tego koniecznych. Przekazywane informacje zawsze są w postaci tekstowej. Skrypt startowy może zawierać ciągi znakowe, dane liczbowe całkowite, dane liczbowe zmiennoprzecinkowe oraz listy parametrów. Na liście umieszczane mogą być ciągi tekstowe oraz wartości liczbowe. Wykorzystane w skryptach znaki specjalne to białe spacje oznaczające odstęp, znak „,“ stosowany jako separator elementów listy, znak „#“ stosowany jako znacznik komentarza, symbol „=“ stosowany jako znacznik przypisania, znak „;“ stosowany jako separator.

Postać skryptu jest stosunkowo prosta. Poszczególne linie skryptu mogą zawierać linie puste, komentarze lub przypisania wartości zmiennym. Przypisanie można zapisać za pomocą następującej gramatyki:

$< id >=< parameter >$

$< parameter >=< listd > | < listi > | < lists >$

$< listi >=< listi, int > | < int >$

$< listd >=< listd, double > | < double >$

$< lists >=< lists, string > | < string >$

Zakłada się, że lista parametrów może być jednolitego typu. Nie dopuszcza się możliwości mieszania typów parametrów na liście. Jest to oczywiście znaczne ograniczenie, niemniej jednak na potrzeby aplikacji obarczona nawet tym ograniczeniem gramatyka całkowicie wystarcza. Zestaw parametrów dostępnych poprzez skrypt przedstawiony został w dodatku.

Ponadto przygotowany musi być zestaw wzorców uczących. Tutaj również przyjęto zasadę, że wzorce uczące będą zapisane w postaci pliku testowego. Dane reprezentowane są przez ciąg wartości odpowiadających wejściu oraz oczekiwanemu wyjściu. Oddzielone są one znakami spacji albo tabulacji, co pozwala na wstępną wizualizację treści pliku. Pozwala to na ocenę rozmiaru wejścia i wyjścia sieci. Informacje na temat liczby wejść oraz wyjść przechowywane są w pliku startowym, zaś liczba wzorców jest rozpoznawana w trakcie analizy pliku wejściowego.

Oprócz skryptów dostępne są jeszcze trzy zmienne powłoki

GABP_LOG_NAME=nazwa katalogu
GABP_SRC_NAME=nazwa katalogu
GABP_OUT_NAME=nazwa katalogu

Zmienne te pozwalają na przekazanie programowi informacji dotyczącej miejsca w którym szukać skryptów startowych. Dzięki temu całkowicie wyeliminowana została konieczność przekazywania parametrów do programu za pomocą linii poleceń. Ustawienie wartości tych zmiennych odbywa się w plikach .profile, .bashrc, lub .cshrc. W przypadku wykorzystania powłoki sh lub csh umieścić należy odpowiednio w pliku .profile lub .cshrc następujące informacje:

setenv GABP_LOG_NAME=nazwa_katalogu_docelowego
setenv GABP_SRC_NAME=nazwa_katalogu_docelowego
setenv GABP_OUT_NAME=nazwa_katalogu_docelowego

Jeżeli wykorzystujemy powłokę bash należy w pliku .bashrc umieścić następujący zestaw poleceń:

GABP_LOG_NAME=nazwa_katalogu_docelowego
GABP_SRC_NAME=nazwa_katalogu_docelowego
GABP_OUT_NAME=nazwa_katalogu_docelowego
export GABP_LOG_NAME GABP_SRC_NAME

W ten sposób zapewnimy sobie dostępność niezbędnych dla programu informacji w trakcie jego pracy. Katalog docelowy zawiera oczywiście nazwę katalogu, w którym umieszczone zostały uprzednio skrypty startowe.

9.6 Opracowywanie wyników

Jak zostało już wspomniane opracowanie wyników w głównej mierze opiera się o zastosowanie języka skryptowego powłoki csh. Należy podkreślić, że przebieg testów oraz późniejsze ich opracowanie jest niemalże całkowicie automatycznym procesem. Jedyną aktywnością użytkownika jest analiza danych końcowych. Oczywiście niestandardowe testy lub takie, które wymagają dodatkowych informacji, wymagają od użytkownika opracowania nowych skryptów powłoki.

Poprawne opracowanie pliku zawierającego wynik przebiegu programu wymaga niestety wiedzy na temat aplikacji systemowych systemu UNIX. Wprawdzie plik rejestru zapisany jest w postaci pliku tekstowego, niemniej jednak umiejętne wyciągnięcie z niego informacji wymaga znajomości programów grep, sed i awk. Użytkownik wprawdzie posiada skrypty pomocnicze niemniej jednak przeznaczone są one do uzyskiwania standardowych informacji, takich jak wartości przystosowania średniego, maksymalnego czy minimalnego, czasu przetwarzania czy informacji na temat postaci genotypu.

Po uprzednim opracowaniu wyników możemy poddać je obróbce za pomocą dowolnego arkusza kalkulacyjnego. Przy opracowywaniu wyników tej pracy wykorzystany został

darmowy arkusz kalkulacyjny znajdujący się w pakiecie StarOffice 5.2.

Dodatek A. Opis parametrów skryptu startowego

W niniejszym dodatku zostały zaprezentowane wszystkie parametry skryptu startowego wraz z krótkim ich opisem:

- Inputs - liczba wejść sieci,
- Outputs - liczba wyjść sieci,
- LearnRate - współczynnik uczenia występujący w algorytmie propagacji wstecznej,
- MomentumRate - współczynnik momentum. Wpływa na znaczne skrócenie procesu uczenia sieci neuronowej,
- ActivationFunctionMode - określa rodzaj wykorzystywanej funkcji aktywacji. Do wyboru dostępnych jest sześć funkcji:
  - 0 - funkcja RAW, zwracająca wartość net neuronu,
  - 1 - funkcja unipolarna,
  - 2 - funkcja bipolarna,
  - 3 - funkcja sigmoidalna unipolarna,
  - 4 - funkcja sigmoidalna bipolarna,
  - 5 - funkcja logarytmiczna jednostkowej aktywacji. Funkcja ta cechuje się tym, że wysoka wartość aktywacji występuje dla dużych wartości bezwzględnych argumentu.
- ActivationFunctionParametersList - jest to lista parametrów funkcji aktywacji. Przyjęta została konwencja, w której pierwszy parameter jest przesunięciem wartości argumentu, zaś pozostałe są zwykle mnożnikami argumentu lub funkcji. W przypadku funkcji sigmoidalnej drugi współczynnik jest odpowiednikiem spotykanego w literaturze współczynnika beta,
- ActivationDerivativeParametersList - współczynniki pochodnej funkcji aktywacji. Ze względu na fakt, że funkcje te są na sztywno zdefiniowane w programie może być koniecznym zdefiniowanie tych współczynników. Dla funkcji już zaimplementowanych współczynniki te powinny się pokrywać,
- DataMaximalPresentationsCount - wartość ta określa, jak wiele może być wykonanych prezentacji wzorców w trakcie procesu uczenia sieci neuronowej,
- RandomPattern - współczynnik ten określa, czy w trakcie procesu uczenia wzorce będą prezentowane w sposób losowy, czy w sposób wynikający z ich kolejności w pliku wejściowym,

- MaximalError - określa jaki jest dopuszczalny największy błąd sieci w trakcie procesu uczenia,
- LeftWeight - minimalna wartość wagi w trakcie inicjacji procesu uczenia,
- RightWeight - maksymalna wartość wagi w trakcie inicjacji procesu uczenia,
- BiasMode - wartość biasu,
- FitnessParametersBad - zestaw siedemnastu wartości reprezentujących wektor wag funkcji aktywacji. Ten zestaw jest wykorzystywany do oceny struktur błędnych,
- FitnessParametersGood - jak wyżej, ale dotyczy struktur poprawnych nienauczonych,
- FitnessParametersLearned - jak wyżej, ale dotyczy struktur nauczonych danego problemu,
- ConnectionsInsideInput - parametr ten określa, czy połączenia między neuronami wejściowymi mogą wystąpić,
- ConnectionsInsideOutput - parametr określa, czy mogą wystąpić połączenia w obrębie neuronów wyjściowych,
- SignalPropagation - parametr decyduje o tym, czy sygnał wejściowy rozprowadzany jest pomiędzy wszystkie neurony wejściowe, czy też każda składowa wektora wejściowego odpowiada pojedynczemu neuronowi,
- WeightModificationMode - współczynnik ten określa, w jaki sposób będą modyfikowane wartości wag sieci w trakcie procesu uczenia. Dostępne są trzy możliwości:
  - 0 - zwykła metoda delta ze współczynnikiem *momentum*,
  - 1 - metoda delta ze zmodyfikowanym współczynnikiem momentum. Współczynnik opiera się o poprzednią wartość pochodnej, a nie wartość różnicy między wagami,
  - 2 - algorytm QuickProp.
- QuickPropParameter - współczynnik występujący w algorytmie QuickProp. Współczynnik ten decyduje, czy wartość zmiany pochodnej jest wystarczająco duża, aby zmiana wag mogła nastąpić,
- ActivationThreshold - jest to przesunięcie progu aktywacji. Zostało jednak zastąpione pierwszym współczynnikiem funkcji aktywacji. Zabieg ten sprawił, że dobór parametrów stał się bardziej intuicyjny,
- InitMode - parametr ten nie jest wykorzystywany,

- CrossOverParameter - współczynnik krzyżowania w algorytmie genetycznym,
- MutationParameter - współczynnik mutacji w algorytmie genetycznym,
- IndividualsCount - liczba osobników w populacji,
- GenotypeSize - rozmiar genotypu. Jest to maksymalna liczba neuronów w sieci,
- EpochsCount - parametr ten określa liczbę epok algorytmu genetycznego,
- TransferParameter - współczynnik określający, jaki procent populacji zostanie rozesłany do pozostałych populacji. Jeżeli liczba osobników jest mniejsza od zera wysyłany jest tylko jeden osobnik,
- CommunicationEpoch - parametr określający co jaką liczbę epok algorytmu genetycznego następować ma komunikacja między procesami,
- Normalization - parametr określa, czy wartości przystosowania mają być normalizowane,
- LeftNorm - określa lewy brzeg przedziału normalizacji. Nowe wartości przystosowania będą większe od tej wartości,
- RightNorm - określa prawy brzeg przedziału normalizacji. Nowe wartości przystosowania będą mniejsze od tej wartości,
- FitnessByPosition - parametr ten decyduje o tym, czy wartość przystosowania ma być liczona na podstawie pozycji osobnika w populacji,
- FitnesCount - parametr decyduje, ile razy sieć będzie uczona w celu określenia jej wartości przystosowania,
- PopulationLogCount - parametr ten określa ile najlepszych rozwiązań zostanie zapisanych do pliku rejestrującego przebieg obliczeń,
- MeshWidth - parametr określa szerokość siatki procesów,
- MeshHeight - parametr określa wysokość siatki procesów,
- OutputDir - parametr definiuje katalog, w którym zapisywane będą wszelkie dane wyjściowe,
- FileBaseName - ten parametr określa podstawową nazwę pliku. Dodatkowo w jej skład będą wchodziły: numer procesu na siatce procesów, nazwa hosta, numer procesu w systemie,
- LogSize - określa rozmiar rejestru przebiegu programu. Po przekroczeniu tej wartości następuje zapis rejestru na dysk,

- TestNetwork -opcja ta pozwala na przetestowanie określonego rozwiązania. Jako wejście dla danych wykorzystuje standardowe wejście,
- ShowLearningProcess - opcja ta pozwala na obserwowanie procesu obliczania wartości fitness każdego osobnika,
- StdOut - parametr ten określa, gdzie ma zostać skierowane standardowe wyjście procesu. Można je skierować albo na *stdout* (wartość 0), lub do zewnętrznego pliku (wartość 1),
- StdOutFileName - parametr ten określa nazwę zewnętrznego pliku, do którego kierowane będzie wyjście z programu. Do nazwy zostanie dołączony numer procesu na siatce procesów, nazwa hosta oraz numer procesu w systemie,
- Force - wymusza na programie działanie w przypadku stwierdzenia braku jednego z parametrów. Wykorzystywane są wtedy wartości znajdujące się w pliku *defaults.dat,*
- LearnDataFileName - nazwa pliku zawierającego wzorce wejściowe,
- TestDataFileName - nazwa pliku zawierającego zestaw wzorców testujących.

Szczegółowe informacje dotyczące każdego z parametrów można znaleźć w pliku *defaults.dat*. W pliku tym opisane są wartości domyślne oraz określone są wymagania stawiane przed watościami parametrów.

Uruchomienie programu odbywa się w oparciu o plik sterujący *parameters.dat*. Wykorzystując ten plik program pobiera niezbędne informacje, a następnie rozpoczyna proces obliczeniowy.

Pliki z danymi uczącymi powinny być przygotowane w taki sposób, że każda linia zawiera zestaw w postaci *<wejście,wyjście>*. Przy czym każda wartość musi być oddzielona znakiem spacji lub tabulacji.

Korekta poprawności danych wejściowych odbywa się na podstawie informacji o liczbie wejść i wyjść sieci. Jeżeli występują niezgodności program przerywa działanie i informuje użytkownika, w której linii wystąpił błąd.

Przygotowanie skryptu może odbywać się w oparciu o plik *defaults.dat*. Należy jedynie usunąć z niego wszystkie prefiksy *default*.

Skorowidz

W

Z

Literatura

[1] David E. Goldberg, „Algorytmy genetyczne i ich zastosowania", WNT, W-wa 1989

[2] Zbigniew Michalewicz, „Algorytmy genetyczne + struktury danych = programy ewolucyjne", WNT, W-wa 1999

[3] John Hertz, „Wstęp do teorii obliczeń neuronowych", WNT, W-wa 1995

[4] Stanisław Osowski, „Sieci neuronowe w ujęciu algorytmicznym", WNT, W-wa 1996

[5] Ryszard Tadeusiewicz, „Sieci neuronowe", Akademicka Oficyna Wydawnicza RM, W-wa 1993

[6] Danuta Rutkowska, „Sieci neuronowe, algorytmy genetyczne i systemy rozmyte", PWN, Łódź 1999

[7] Laurens Jan Pit, „Parallel Genetic Algorithms", Department of Computer Science, Leiden University 1995

[8] Martin Mandischer, „Non-binary and Dynamic Encoding for the Evolution of Neural Networks", University of Dortmund 1995

[9] Martin Mandischer, „Representation and Evolution of Neural Networks", University of Dortmund 1993

[10] B.A. Thijssen, „Adaptive Genetic Algorithms with Multiple Subpopulations and Multiple Parents", 1997

[11] Jurgen Branke, „Evolutionary Algorithms for Neural Network Design and Training", University of Karlsruhe 1995

[12] Johr R. Koza, „Genetic Generation of Both the Weights and Architecture for Neural Network", Stanford University 1991

[13] D. Koll, „Massively Parallel Training of Multi Layer Perceptrons with Irregular Topologies", University of Karlsruhe 1995

[14] Chris Thornton, „Design of Artificial Neural Networks Using Genetic Algorithms: review and prospect", University of Sussex 1994

[15] Riccardo Poli, „Parallel Distributed Genetic Programming", The University of Birmingham 1996

[16] Ian Foster, „Design and Building Parallel Programs", http://www-unix.mcs.anl.gov/dbpp/text/book.html, 1995

Spis rysunków

Spis tabel