Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

Ewolucyjna metoda syntezy skalowalnych i modularnych sieci obliczeniowych

mgr inż. Marcin Suchorzewski

promotor: prof. dr hab. inż. Andrzej Piegat

Rozprawa doktorska

Katedra Metod Sztucznej Inteligencji i Matematyki Stosowanej
Wydział Informatyki
Zachodniopomorski Uniwersytet Technologiczny

Szczecin 2011

Streszczenie

Ewolucyjne sieci neuronowe, lub inaczej neuroewolucja, są interesującym i obiecującym sposobem syntezy programów i systemów adaptacyjnych, łączących w sobie ewolucję i uczenie. Jednym z najważniejszych wyzwań stojących przed sieciami ewolucyjnymi jest znalezienie reprezentacji, zapewniającej możliwość efektywnej ewolucji coraz bardziej złożonych rozwiązań dla coraz bardziej złożonych problemów. W niniejszej pracy przedstawiono nową metodę ewolucji sieci obliczeniowych — będących pewnym uogólnieniem sieci neuronowych — która oparta jest o rozwojową reprezentację genetyczną, umożliwiającą generowanie rozwiązań odznaczających się skalowalnością i modularnością.

Skalowalność polega tutaj na zdolności do odzwierciedlenia pewnej regularności problemu w fenotypie sieci oraz do jej uchwycenia w genotypie sieci. Modularność, będąca poniekąd istotnym elementem skalowalności, oznacza zdolność do rozłożenia sieci na wiele mniejszych, funkcjonalnie niezależnych i możliwie wielokrotnie wykorzystanych podsieci. Te dwie cechy zostały zademonstrowane na wielu przykładach i problemach testowych. W szczególności dla problemu parzystości i symetrii wyewoluowano rozwiązania, które są doskonale skalowalne ze względu na liczbę wejść. W wyniku eksperymentów otrzymano również skalowalne i modularne nieważone rekurencyjne sieci zdolne do uczenia nadzorowanego w prostym generycznym zadaniu klasyfikacji. Ponadto wykazano, że sieci ewoluowane za pomocą opracowanej metody mogą być dla pewnych problemów bardziej efektywne pod względem jakości i szybkości uczenia aniżeli standardowe sieci neuronowe. Zaproponowane kodowanie genetyczne odznacza się kompletnością, co oznacza, iż dowolna sieć rekurencyjna — a tym samym dowolna maszyna Turing'a — może być teoretycznie wygenerowana.

Opracowana metoda jest bardzo elastyczna, pozwala generować sieci jednokierunkowe i rekurencyjne, ważone i nieważone, o połączeniach plastycznych i modulowanych, a nawet o dowolnych funkcjach przejścia i uczenia, które same również mogą podlegać adaptacji ewolucyjnej. Metoda pozwala wykorzystać uczenie oparte o wsteczną propagację błędu oraz neuromodulację, co zademonstrowano na przykładach. Na przykładzie dwóch problemów testowych wykazano również, że metoda jest konkurencyjna wobec aktualnie rozwijanej metody neuroewolucji HyperNEAT.

Spis treści

1 Wstęp **1**

2 Wprowadzenie **9**
2.1 Adaptacja: ewolucja i uczenie 11
2.2 Ewolucyjne sztuczne sieci neuronowe 15
2.2.1 Rozwojowe reprezentacje genetyczne 18
2.3 Skalowalność i modularność sieci 19
2.3.1 Skalowalność 19
2.3.2 Modularność 22
2.4 Metody ewolucji sieci 23
2.4.1 Kodowanie Gramatyczne 24
2.4.2 Kodowanie Komórkowe 24
2.4.3 Analogiczne Kodowanie Genetyczne 27
2.4.4 HyperNEAT 29
2.4.5 Podsumowanie 32

3 Rozwojowe Kodowanie Symboliczne **35**
3.1 Algorytm ewolucyjny 36
3.2 Fenotyp sieci 39
3.3 Genotyp sieci 41
3.4 Rozwój 45
3.5 Operatory genetyczne 47
3.6 Właściwości metody 48

4 Badania symulacyjne **53**
4.1 Skalowalność i modularność 54
4.1.1 Problem symetrii 54
4.1.2 Problem parzystości 55
4.1.3 Problem siatkówki 58
4.1.4 Problem topologii docelowej 61
4.1.5 Dyskryminacja wizualna 65
4.1.6 Problem uczenia nadzorowanego w klasyfikacji 68
4.2 Ewolucja skalowalnych i modularnych topologii 74

4.2.1 Sieci bezskalowe 74
4.2.2 Skalowalne sieci modularne 75
4.2.3 Regularne sieci modularne 78
4.3 Ewolucja równoległa a procedury 80
4.4 Wsteczna propagacja błędu 81
4.4.1 Problem klasyfikacji 83
4.4.2 Problem spiral 84
4.5 Neuromodulacja 89
4.6 Sieci ewolucyjne a generyczne 91
4.7 Problemy nieregularne 94
4.7.1 Problem sztucznej mrówki 94
4.7.2 Problem podwójnego odwróconego wahadła 97

5 Podsumowanie i wnioski **101**

Słownik pojęć **106**

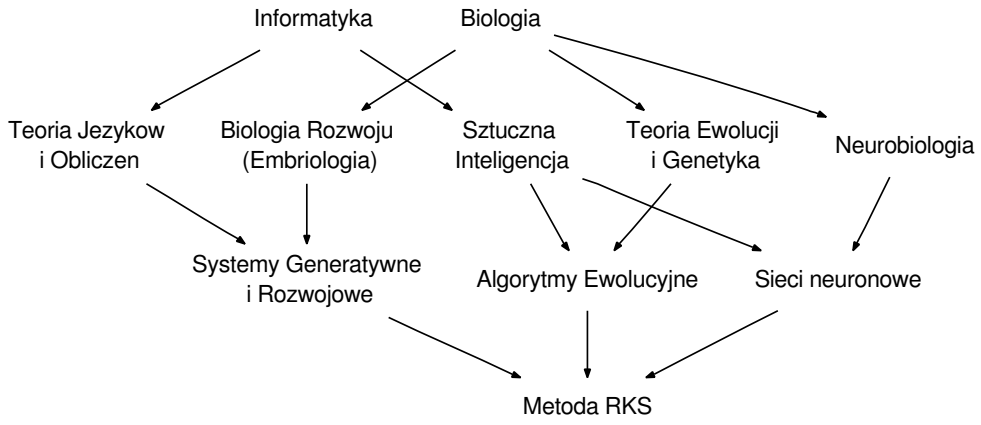
Bibliografia **108**

Rozdział 1

Wstęp

Natura, jej złożoność i bogactwo form, jest niewyczerpanym źródłem inspiracji. Metody obliczeniowe inspirowane naturą od wielu lat znajdują coraz szersze zastosowania w nauce i technice. Algorytmy immunologiczne, automaty komórkowe, inteligencja roju, a przede wszystkim algorytmy ewolucyjne i sztuczne sieci neuronowe pozwalają rozwiązywać coraz trudniejsze problemy, a ich znaczenie i możliwości będą najprawdopodobniej wzrastać wraz z dostępną mocą obliczeniową komputerów.

Niniejsza praca wpisuje się w tenże nurt badań nad metodami inspirowanymi naturą. Podejmuje próbę powiązania dwóch w dużym stopniu odrębnych metod obliczeniowych — algorytmów ewolucyjnych i sztucznych sieci neuronowych — za pomocą jeszcze innego systemu inspirowanego naturą: systemu rozwojowego, czerpiącego inspiracje z biologii rozwojowej i embriologii (rys. 1.1).



Rysunek 1.1: Problematyka pracy w kontekście obszarów badań.

Algorytmy ewolucyjne (AE) zdolne są do rozwiązywania wszelkiego rodzaju problemów optymalizacyjnych, lecz w ogólności nie umożliwiają adaptacji na bieżąco (*online*). Z kolei głównym atutem sztucznych sieci neuronowych (SSN)[1] jest zdolność do szybkiej adaptacji w reakcji na zmiany w środowisku, jednak ich projektowanie wymaga wiedzy i zaangażowania eksperta, i nawet wówczas niekoniecznie dobrze są dopasowane do problemu. Fuzja tych dwóch metod ma na celu uzyskanie metody łączącej w sobie ich zalety, czyli zdolność do uczenia się, która w procesie ewolucji, z pokolenia na pokolenie będzie coraz bardziej efektywna. Wykorzystanie AE do syntezy sieci obliczeniowych ma wiele zalet, a w szczególności:

- Prostota w narzucaniu ograniczeń i wymagań projektowych sieci.
- Zdolność do wykrycia i wykorzystania ukrytych właściwości problemu (środowiska), które byłyby trudne do uchwycenia przez eksperta i wyzyskania przez zaprojektowane przez niego sieci.
- Ewolucyjne (automatyczne) ustalenie kompromisu pomiędzy specjalizacją a generalizacją sieci, regulowane poprzez odpowiedni dobór funkcji przystosowania.
- Możliwość wyewoluowania dowolnej sieci rekurencyjnej, a więc teoretyczna możliwość rozwiązania dowolnego problemu obliczalnego (w sensie Turing'a).

Fuzja AE i SSN nie jest koncepcją zupełnie nową. Istnieją metody, znane jako ewolucyjne sztuczne sieci neuronowe (ESSN), lub krócej jako metody neuroewolucyjne (*neuroevolution*), które taką fuzję realizują. Jednakże sposobów realizacji powiązania AE i SSN jest bardzo wiele, a złożoność tego rodzaju metod sprawia, że trudno jest oszacować ich efektywność i użyteczność. Tym bardziej nie sposób oszacować ich przydatności przed ich opracowaniem i przetestowaniem. Niniejszą pracę można więc potraktować jako propozycję nowej metody, o której użyteczności niewiele można jeszcze powiedzieć, ale której interesujące właściwości, zarówno teoretyczne jak i praktyczne, zostały wykazane, a jej działanie zademonstrowane na wielu problemach testowych.

Zaproponowana w pracy metoda została nazwana Rozwojowe Kodowanie Symboliczne (RKS, *developmental symbolic encoding*). Jest to metoda ewolucji sieci, ogólnie nazwanych obliczeniowymi, choć zazwyczaj są one równoważne sieciom neuronowym. Wprowadzenie terminu „obliczeniowy" w miejsce „neuronowy" ma na celu podkreślenie większej swobody w określaniu różnych elementów architektury sieci, w sposób niekoniecznie zgodny z naturalnymi

[1] Sztuczna sieć neuronowa jest przypadkiem nieco bardziej szczególnym aniżeli sieć obliczeniowa, choć w pracy najczęściej używamy tych pojęć zamiennie.

wzorcami. Przykładowo, w rozpatrywanych sieciach obliczeniowych wykorzystywane są funkcje przejść o charakterze logicznym czy połączenia modulowane (łączące ze sobą 3 lub więcej węzłów), a więc elementy niespotykane w ramach SSN. Ponadto możliwe jest jeszcze dalej posunięte uogólnienie modelu obliczeniowego sieci, który wykroczy poza ramy SSN, jak na przykład wykorzystanie węzłów o określonym interfejsie, tj. przetwarzających sygnały wejściowe w zależności od ich rodzaju, a nie tylko dokonujących agregacji w postaci sumowania.

Motywacja i cel pracy

Ogólną motywacją dla podjęcia tematu pracy jest potrzeba uniwersalnej metody automatycznej syntezy programów, a więc potrzeba sięgająca korzeniami podstaw informatyki i jak dotąd w niewielkim stopniu zrealizowana. Na dodatek wizja efektywnej, uniwersalnej metody została oddalona przez twierdzenie o „darmowym lunchu" (Wolpert and Macready, 1997), według którego nie istnieje taka uniwersalna metoda rozwiązywania problemów, która byłaby przeciętnie efektywniejsza od losowego poszukiwania. Twierdzenie to jest niezwykle demotywujące dla dalszych poszukiwań metody uniwersalnej, gdyż sugeruje ono, że jedynym sposobem zwiększenia efektywności metody jest wyspecjalizowanie jej dla określonej klasy problemów. Ponieważ najbardziej interesujące są problemy rzeczywiste i „praktyczne", poszukiwania metody można skoncentrować na takich właśnie problemach. Możliwe jest tu zyskanie wielkiej efektywności, gdyż problemy praktyczne stanowią niewątpliwie tylko niewielką część przestrzeni wszystkich problemów, choć niestety trudno jest wyznaczyć tę klasę problemów w ścisły sposób. Choć metoda może być wyspecjalizowana dla pewnej klasy problemów, teoretycznie nadal może być uniwersalna, jeśli tylko zdolna jest do — choćby niezbyt efektywnego — rozwiązania dowolnego problemu obliczalnego (w sensie Turing'a). W ten sposób dochodzimy do koncepcji metody uniwersalnej, lecz wyspecjalizowanej dla problemów rzeczywistych.

Jakkolwiek koncepcja brzmi rozsądnie, zdefiniowanie klasy problemów rzeczywistych jest trudne. Cel pracy sformułowany jest zatem nie w kategoriach klasy adresowanych problemów, a w kategoriach cech, jakimi muszą odznaczać się ich rozwiązania. Jak wskazuje na to sam temat, celem pracy jest opracowanie metody syntezy sieci obliczeniowych, odznaczających się skalowalnością i modularnością. Cechy te odnoszą się zarówno do samych sieci, tj. ich fenotypu, jak i do sposobu ich reprezentacji, tj. ich genotypu. Tak więc przez skalowalność można tu rozumieć taki sposób zakodowania sieci, który pozwala bądź ułatwia generowanie sieci rozwiązujących problemy o zmiennej złożoności lub wymiarowości. Przez modularność można rozumieć taki sposób zakodowania sieci, który pozwala bądź ułatwia generowanie sieci składających się z wielu podobnych bądź identycznych fragmentów strukturalnych, tj. podsieci. Te

dwie cechy są ze sobą powiązane, przy czym modularność w pewnym sensie zawiera się w skalowalności. Nacisk na generowanie sieci skalowalnych i modularnych wynika stąd, że wiele problemów rzeczywistych ma tego rodzaju regularny charakter, który od rozwiązań wymaga odzwierciedlenia również w postaci pewnych regularności — wzorców strukturalnych, powtarzających się i przeplatających zgodnie z pewną ukrytą prawidłowością. Przykładem struktury, która odznacza się taką regularnością jest kora mózgowa u ssaków.

Zakres i tezy pracy

W ramach pracy opracowano metodę syntezy sieci obliczeniowych, której najważniejszą częścią jest reprezentacja genetyczna sieci. Metoda odznacza się kilkoma istotnymi właściwościami o znaczeniu teoretycznym oraz praktycznym. Opis metody oraz jej najważniejszych właściwości przedstawiono w rozdziale 3. W Rozdziale 4 wykazano doświadczalnie jej właściwości i zademonstrowano potencjał praktyczny na wielu przykładach i problemach testowych. Poważne, lecz nie bezzasadne ograniczenie pracy dotyczy zakresu eksperymentalnego porównania opracowanej metody z innymi metodami syntezy sieci. To ograniczenie wynika ze złożoności tego rodzaju metod, ich „niedookreśloności" wynikającej z mnogości i arbitralności szczegółów, oraz z wielości ich parametrów. Ponadto konkurencyjne metody nie zawsze są publicznie dostępne, a nakład pracy na ich re-implementację byłby znaczny.

Cztery główne tezy stawiane w pracy są następujące:

1. Metoda umożliwia wygenerowanie rozwiązania dla dowolnego problemu obliczalnego w sensie Turing'a.

2. Metoda umożliwia syntezę adaptacyjnych sieci odznaczających się skalowalnością i modularnością.

3. Metoda umożliwia syntezę sieci uczących się co najmniej równie efektywnie jak sztuczne sieci neuronowe.

4. Istnieją takie problemy, dla których metoda pozwala generować sieci uczące się efektywniej aniżeli standardowe sieci neuronowe.

Teza 1. ma charakter czysto teoretyczny. Jej dowód, przedstawiony w §3.6, polega na wykazaniu, że metoda umożliwia wygenerowanie dowolnej sieci rekurencyjnej, co zgodnie z twierdzeniem Siegelmann'a i Sontag'a (patrz Haykin, 1998) oznacza, że umożliwia wygenerowanie dowolnej maszyny Turing'a, a więc rozwiązania dla dowolnego problemu obliczalnego.

Dowód tezy 2. ma charakter częściowo empiryczny. Możliwość generowania modularnych i skalowalnych sieci wynika z właściwości reprezentacji genetycznej wykorzystywanej w metodzie, ale również może być wykazana na podstawie

rozwiązań, które takimi cechami się odznaczają (§4.1). Bardziej ścisłe ujęcie pojęć modularność i skalowalność znajduje się w §2.3.

Teza 3. zbliżona jest do tezy 1. z tym, że odnosi się do efektywności metod powszechnie stosowanych — sztucznych sieci neuronowych. Dowód tej tezy jest względnie prosty: metoda umożliwia wygenerowanie dowolnej sieci obliczeniowej — w ostateczności tej sieci neuronowej, z którą dokonane ma być porównanie pod względem efektywności uczenia się. Można jednak przypuszczać, że dla większości problemów istnieją sieci znacznie lepsze niż sieci standardowe, a zaproponowana metoda daje szansę na ich znalezienie.

Dowód tezy 4. ma charakter empiryczny i polega na przedstawieniu takich problemów, dla których rozwiązania wyewoluowane są pod względem określonych kryteriów lepsze aniżeli generyczne, czyli standardowe SSN (patrz rozdział 4). W szczególności takim kryterium jest efektywność i czas uczenia się, opcjonalnie przy uwzględnieniu złożoności sieci. Teza 4. związana jest z ogólną cechą adaptacji ewolucyjnej, która umożliwia znajdowanie optymalnego, lub prawie optymalnego kompromisu pomiędzy specjalizacją i ogólnością rozwiązania dla zadanej funkcji przystosowania.

Oprócz postawionych powyżej tez, w pracy przedstawiono również kilka innych praktycznych właściwości metody i zademonstrowano je na przykładach.

Metoda badawcza

Opracowywanie nowej metody to proces dość złożony, wymagający szerokiego rozeznania w problematyce, analizy istniejących rozwiązań i nowych pomysłów na rozwiązania oraz ich eksperymentalną weryfikację. Ponadto jest to proces, który może być cyklicznie powtarzany celem uzyskania coraz lepszych wyników. Tego rodzaju procedura wymyka się typowej klasyfikacji metod badawczych, gdy tymczasem jest ona bardzo naturalna — „algorytm opracowywania metody” mocno przypomina algorytm ewolucyjny:

1: Stan początkowy: wiedza i metoda wyjściowa
2: **Dopóki** jest jeszcze czas **wykonuj**
3: Propozycje modyfikacji i nowych rozwiązań
4: Weryfikacja rozwiązań
5: Akumulacja wiedzy i korzystnych rozwiązań

Wprawdzie struktura populacji jest w tym algorytmie bliżej nieokreślona — może to być nawet pojedynczy osobnik — widać wyraźnie odpowiedniki reprodukcji i wariacji (krok 3), oceny (4) oraz selekcji (5). Interesujące, że charakter ewolucyjny można przypisać nie tylko niniejszej pracy, ale również całej dyscyplinie sztucznej inteligencji, informatyce oraz całej nauce. Co więcej, wizjonerzy zastanawiają się nawet jak systematycznie wykorzystać ewolucję w badaniach:

> Ewolucja jest rozwiązaniem problemu „jak rozwiązywać problemy”. Największa korzyść z algorytmów ewolucyjnych może być odniesiona przy zastosowaniu ich do tych problemów, które jak dotąd opierały się analizie ekspertów, gdyż obliczenia ewolucyjne nie ograniczają się do odwzorowania postępowania człowieka. „Spostrzeżenie, iż metoda naukowa sama jest procesem ewolucyjnym, który może być symulowany, otwiera możliwości automatyzacji wyższych procesów intelektualnych.”[2] Tego rodzaju przedsięwzięcie z pożytkiem może połączyć badania nad systemami ekspertowymi, rozmytymi oraz sieciowymi. Podejście ewolucyjne wcale nie wyklucza wiedzy człowieka, a wprost przeciwnie, jest jak najbardziej otwarte na hybrydyzację, co tylko zwiększa jego użyteczność (Fogel, 2006).

Wyniki pracy

W ramach pracy opracowano nową metodę syntezy sieci obliczeniowych. Główną częścią metody jest reprezentacja genetyczna sieci, która pozwala generować sieci odznaczające się skalowalnością i modularnością, co zostało wykazane doświadczalnie. Wykazano również prawdziwość postawionych tez. Ponadto zademonstrowano szereg innych właściwości praktycznych metody. Metoda umożliwia syntezę sieci, o niemal dowolnej architekturze:

- Dowolna topologia — jednokierunkowa oraz rekurencyjna.
- Dowolne rodzaje połączeń — nieważone, ważone, plastyczne (adaptacyjne), modulowane.
- Połączenia plastyczne i modulowane mogą mieć dynamikę (tj. funkcje przejść, parametry) ustaloną z góry lub podlegającą ewolucji.
- Dowolne rodzaje węzłów (tj. operacje, funkcje przejść) sieci, które również mogą podlegać ewolucji.
- Dowolnie określony interfejs sieci.
- Dowolne ograniczenia i wymagania projektowe, dające się wyrazić poprzez funkcję przystosowania.
- Możliwość wykorzystania wstecznej propagacji błędu do uczenia sieci.

W ramach doświadczeń przedstawiono rozwiązania dla następujących problemów:

- Funkcja parzystości dla dowolnej liczby wejść, §4.1.2.

[2] L. J. Fogel, A. J. Owens, M. J. Walsh. Artificial Intelligence Through Simulated Evolution. Wiley, New York, 1966

- Funkcja symetrii dla dowolnej liczby wejść, §4.1.1.
- Problem sztucznej siatkówki, §4.1.3.
- Problem topologii docelowej, §4.1.4.
- Dyskryminacja wizualna, §4.1.5.
- Uczenie nadzorowane w generycznym zadaniu klasyfikacji, §4.1.6.
- Przeplatające się spirale; rozwiązanie wykorzystujące wsteczną propagację błędu, §4.4.2.
- Klasyfikacja; przy wykorzystaniu wstecznej propagacji błędu, §4.4.1.
- Ewolucja sieci bezskalowych, przy narzuconych ograniczeniach, §4.2.1.
- Ewolucja sieci modularnych i regularnych przy narzuconych ograniczeniach, §4.2.2 i §4.2.3.
- Ewolucja równoległa — 4-argumentowe funkcje logiczne, §4.3.
- Uczenie nadzorowane w generycznym zadaniu klasyfikacji „ruchomy pas”, §4.6.
- Sztuczna mrówka na szlaku Santa Fe, §4.7.1.
- Sterownik podwójnego odwróconego wahadła, §4.7.2.

Rozdział 2

Wprowadzenie

W artykule „Computing Machinery and Intelligence” z 1950 r. Alan Turing rozważa problem możliwości i sposobu wytwarzania inteligentnych maszyn:

> Nie należy oczekiwać, iż dobra maszyna potomna zostanie znaleziona przy pierwszej próbie. Niezbędne będą eksperymenty z jej uczeniem i ocena jej zdolności. Następnie można będzie wypróbować inną maszynę i je porównać. Istnieje więc wyraźny związek pomiędzy tą procedurą a ewolucją:
> ‘struktura maszyny potomnej’ = materiał genetyczny
> ‘zmiany w maszynie potomnej’ = mutacje
> ‘selekcja naturalna’ = ocena eksperymentatora

Od tego czasu, koncepcja Turing’a została wielokrotnie sformalizowana w postaci takich metod, jak programowanie ewolucyjne (Lawrence J. Fogel, 1960), strategia ewolucyjna (Ingo Rechenberg, 1973), algorytm genetyczny (John Holland, 1975) czy programowanie genetyczne (John R. Koza, 1990), oraz wielu innych odmian i wariantów tychże, ogólnie zaliczanych do kategorii algorytmów ewolucyjnych (AE). Szczególnie bliskie koncepcji Turing’a jest właśnie programowanie genetyczne (PG), a to z tego względu, iż w odróżnieniu od innych metod, bezpośrednio podejmuje problem wytwarzania programów, czyli maszyn Turing’a.

Gdyby można było udowodnić, że PG jest optymalną metodą automatycznego wytwarzania programów, niniejszej rozprawy z pewnością by nie było. Na taki dowód jednakże wcale się nie zanosi, dlatego poszukiwanie metody, która byłaby od PG dla problemów praktycznych efektywniejsza wydaje się zupełnie zasadne.

Programowanie genetyczne jest uniwersalną, ewolucyjną metodą indukcji programów, która znalazła szerokie zastosowanie i dostarczyła wielu interesujących rozwiązań w różnych dziedzinach (Koza et al., 2003; Poli et al., 2008; Koza, 2010). Uniwersalną, znaczy zdolną do rozwiązywania bardzo szerokiego zakresu problemów, a w przypadku pewnego wariantu tej metody, teoretycznie dowolnego problemu obliczalnego w sensie Turing'a (Teller, 1994). Nie znaczy to oczywiście, że PG świetnie sobie radzi z każdym problemem. Metoda ta ma swoje słabe strony i ograniczenia, m.in. ze względu na złożoność problemu, jaki może być przez nią efektywnie rozwiązany (Woodward and Bai, 2009). Niemniej, Koza (2010) przytocza 76 przykładów rozwiązań wygenerowanych przy pomocy PG, które zaliczają się do kategorii *human-competitive results* — czyli rozwiązań w określony sposób konkurencyjnych wobec dokonań człowieka, a konkretniej spełniających co najmniej jeden z poniższych warunków:

A Rozwiązanie było w przeszłości opatentowane, stanowi ulepszenie opatentowanego rozwiązania, lub jest patentowalne obecnie.

B Rozwiązanie jest równe lub lepsze niż rozwiązanie uznane za naukowe w czasie publikacji w czasopiśmie naukowym.

C Rozwiązanie równe lub lepsze niż rozwiązanie umieszczone w bazie rozwiązań utrzymywanej przez międzynarodową komisję ekspertów.

D Rozwiązanie jest samo w sobie na tyle znaczące, że zasługujące na publikację w czasopiśmie naukowym.

E Rozwiązanie równe lub lepsze aktualnie znanym rozwiązaniom dla starego i dobrze znanego problemu, dla którego sukcesywnie proponowano coraz lepsze rozwiązania/

F Rozwiązanie równe lub lepsze niż rozwiązanie uznane za osiągnięcie w czasie jego publikacji.

G Rozwiązanie dla problemu o niekwestionowanej trudności w danej dziedzinie.

H Rozwiązanie wygrywające w konkursie z udziałem ludzi (jako graczy lub jako programistów).

Ewolucyjna synteza inteligentnych agentów, czy to sprzętowych czy programowych, jest obszarem wciąż wzrastającego zainteresowania (Balakrishnan and Honavar, 2001a; Floreano and Mattiussi, 2008). Agenty mogą być ewoluowane dla niemal dowolnego reprodukowalnego problemu, a najbardziej interesujących rozwiązań można się spodziewać dla tych problemów, których analiza jest trudna, o których wiedza jest skromna i dla których brakuje sprawdzonych metod rozwiązywania.

Przypuszczalnie jednym z głównych czynników ograniczających możliwości PG jest reprezentacja genetyczna jaką ta metoda (standardowo) wykorzystuje.

Reprezentacja ta należy do rodzaju reprezentacji bezpośrednich, które cechują się tym, że genotyp i fenotyp są nierozdzielne, tj. stanowią tę samą strukturę.[1] Metaforycznie ujmując, przestrzeń genotypów (syntaktyczna) i przestrzeń fenotypów (semantyczna) są tu ze sobą tak niefortunnie sklejone, iż podobnym genotypom z reguły nie odpowiadają podobne fenotypy — przestrzenie te nie są do siebie podobne, nie są ze sobą skorelowane (patrz Rothlauf, 2006). Ta właśnie cecha przypuszczalnie wpływa na ograniczoną „ewolucyjność" PG. Inną wadą reprezentacji bezpośredniej, jest wynikająca z niej nieskalowalność rozwiązań, tj. nieefektywność przy rozwiązywaniu problemów o wzrastającej wymiarowości. Generalnie, reprezentacja bezpośrednia ma swoje zalety, takie jak prostota i efektywność w rozwiązywaniu niewielkich problemów, ale odznacza się przy tym ograniczonymi możliwościami uchwycenia regularności w problemie.

Alternatywą dla reprezentacji bezpośrednich są reprezentacje niebezpośrednie, inaczej zwane rozwojowymi. Kierunek badań znany jako *generative and developmental systems* (Stanley, 2008; Harding and Banzhaf, 2008) próbuje odpowiedzieć na pytanie w jaki sposób reprezentacja rozwojowa i sam proces rozwoju może zwiększyć efektywność algorytmów ewolucyjnych i umożliwić im rozwiązywanie coraz bardziej złożonych problemów. W jaki sposób rozdzielić genotyp programu od jego funkcji tak, aby przestrzenie genotypów i fenotypów były skorelowane, i jednocześnie aby umożliwić zwięzły zapis fenotypu za pomocą genotypu, czyli *de facto* umożliwić uchwycenie regularności fenotypu. Czerpiąc inspiracje z teorii ewolucji, genetyki i rozwoju, badania zwróciły się ku sieciom neuronowym, jako naturalnej postaci fenotypu ewoluujących programów. Właściwość uniwersalnej aproksymacji cechująca sieci wielowarstwowe, a także zdolność do uniwersalnych obliczeń cechująca sieci rekurencyjne, pozwalają przyjąć tę właśnie postać fenotypu bez utraty teoretycznej zdolności do rozwiązania dowolnego problemu obliczalnego. Uwzględniając dodatkowo bogatą wiedzę na ich temat i szereg algorytmów uczenia, które mogą być zaaplikowane do wyewoluowanej struktury, sieci neuronowe wydają się być znakomitym wehikułem do podążania za wizją Turing'a odnośnie budowy inteligentnych maszyn. Kluczowy problem jaki pozostał do rozwiązania to kwestia reprezentacji genetycznej dla takich sieci. Jest to problem, który przykuł uwagę wielu badaczy i doczekał się wielu propozycji rozwiązań. Przegląd najważniejszych metod ewolucji sieci neuronowych przedstawiono w §2.4.

2.1 Adaptacja: ewolucja i uczenie

Algorytmy ewolucyjne są wykorzystywane do rozwiązywania szerokiego zakresu problemów, mających najczęściej postać statycznych problemów opty-

[1] Interesujące, iż w świecie biologii taka sytuacja dotyczy tylko najprostszych, replikujących się łańcuchów RNA.

malizacyjnych. Adaptacja czysto ewolucyjna ma jednak swoje ograniczenia, a mianowicie nie umożliwia rozwiązywania problemów o zmiennym, dynamicznym lub niepewnym charakterze. Jeśli problem (środowisko) zmienia się z pokolenia na pokolenie, lub wręcz „w czasie życia" osobnika, wówczas adaptacja ewolucyjna zwyczajnie za tymi zmianami nie nadąża. Przystosowanie ewolucyjne wymaga z reguły wielu pokoleń, wielu cykli selekcji i reprodukcji, aby mogło być zrealizowane.

Do rozwiązywania problemów wymagających szybkiej adaptacji, znakomicie nadają się sztuczne sieci neuronowe (SSN). Są one zdolne do uczenia się na bieżąco, w czasie wykonywania zadania. Proces uczenia może zachodzić szybko, w reakcji na nowe i zmieniające się warunki problemu (nowe dane). Jednak poważnym problemem przy wykorzystaniu SSN jest problem doboru ich architektury. Dobór sieci — czyli wszystkich elementów jej architektury, takich jak liczba i rodzaje neuronów, topologia i algorytm uczenia — z reguły polega na mniejszej lub większej wiedzy i doświadczeniu eksperta, ale nawet duża wiedza i doświadczenie eksperta nie zawsze prowadzą do dobrych rozwiązań.

Rozwiązaniem, które ma na celu połączenie obydwu form adaptacji są ewolucyjne sztuczne sieci neuronowe (ESSN). Zadaniem ewolucji jest tutaj znalezienie optymalnej architektury sieci dla stacjonarnych warunków zadania, natomiast uczenie się odpowiada za dobór parametrów sieci (wag) w odpowiedzi na warunki zmienne:

> Ewolucja i uczenie są to dwie formy biologicznej adaptacji, która zachodzi na różnych poziomach w czasie i przestrzeni. Ewolucja jest procesem selektywnej reprodukcji w obrębie geograficznie rozlokowanej populacji osobników przejawiających pewne zróżnicowanie. Uczenie natomiast, jest procesem modyfikacji zachodzącym w obrębie poszczególnych osobników w czasie ich życia. Adaptacja ewolucyjna umożliwia uchwycenie zmian w środowisku zachodzących relatywnie powoli, liczonych w pokoleniach, takich na przykład jak właściwości percepcyjne różnych rodzajów pożywienia dla danego gatunku ptaka. Uczenie zaś, pozwala osobnikowi na adaptację do warunków zmieniających w sposób gwałtowny i nieprzewidywalny. Uczenie może wykorzystywać różne mechanizmy zmian adaptacyjnych w czasie życia, takie jak rozwój fizyczny, kształtowanie się układów nerwowych i plastyczność synaptyczna. W końcu, podczas gdy ewolucja operuje na genotypie, uczenie dotyczy tylko fenotypu i jego zmiany nie mogą być odzwierciedlone w genotypie (Nolfi and Floreano, 1999).

Związek pomiędzy ewolucją i uczeniem organizmów biologicznych został dostrzeżony już w XIX w. przez James'a Baldwin'a (Baldwin, 1896), który zwrócił uwagę, że zdolność do uczenia jest jedną z cech — jedną z najważniejszych cech — które zapewniają organizmom przewagę adaptacyjną i tym

samym selekcyjną. A więc zdolność do uczenia się wpływa na przebieg ewolucji, ukierunkowując ją na poszukiwanie coraz lepszych mechanizmów uczenia. Jednak uczenie ma również swój koszt — organizmy wymagają interakcji ze środowiskiem, mogą popełniać błędy, a cały proces wymaga czasu. Dlatego pomiędzy uczeniem i ewolucją musi zostać znaleziony odpowiedni kompromis, taki właśnie, aby stałym cechom środowiska odpowiadało „stałe" przystosowanie ewolucyjne, zaś cechom zmiennym odpowiadała plastyczna zdolność do przystosowania się do nich.

Interakcje pomiędzy ewolucją i uczeniem — przykład

Prosty i elegancki przykład ilustrujący związek pomiędzy ewolucją i uczeniem w kontekście ESSN przedstawił Littman (1996). Rozważmy środowisko, w którym znajdują się dwa bajora A i B, w których mieszkają żarłoki (agenty). Żarłoki odżywiają się glonami, które jednak nie są łatwą zdobyczą. W bajorze A jadalne glony są zielone, zaś trujące są czerwone, w bajorze B z kolei na odwrót. Na dodatek żarłok ma równe szanse przyjścia na świat w obrębie każdego z bajor, a wówczas pozostanie tam już do końca swej próby. Ze względu na tę niepewność, lub też w pewnym sensie zmienność środowiska, adaptacja czysto ewolucyjna jest niewystarczająca dla powodzenia agenta w środowisku. Do tego niezbędne jest nauczenie się, które glony są jadalne, a które trujące. Jednakże w tak sformułowanym zadaniu brakuje informacji, która mogłaby być wykorzystana w procesie uczenia (wykluczamy „podglądanie" co jedzą inni). Rozważmy więc teraz dwa warianty tego problemu, w których agenty uczą się w sposób nienadzorowany oraz ze wzmocnieniem. W obydwu przypadkach przyjmujemy, że sensory przyjmują wartości ze zbioru $\{-1, 1\}$, zaś wyjście ze zbioru $\{0, 1\}$.

Załóżmy, że oprócz koloru, glony posiadają jeszcze zapach, tym razem mający jednakowe znaczenie w obydwu środowiskach — jadalne glony mają zapach słodki, trujące gorzki. Jednak informacja ta jest niepewna, gdyż na skutek turbulencji w powietrzu w 25% przypadków (próbach konsumpcji) zapach jest zdradliwy. Jedząc glony zawsze zielone, oczekiwana wartość przystosowania agenta, mierzona jako skuteczność w spożywaniu glonów jadalnych, wynosiłaby 50%. Gdyby agent jadł zawsze tylko słodko pachnące glony, wartość ta wzrosłaby do 75%. Sposobem na dalsze zwiększenie przystosowania jest wykorzystanie obydwu informacji i nauczenie agenta, który kolor jest skojarzony ze słodkimi glonami. Sieć, rozwiązująca to zadanie przedstawiona jest na rysunku 2.1a. Połączenie pomiędzy percepcją zapachu słodkiego, a akcją spożycia jest w sieci zakodowane genetycznie i nie wymaga zmian, gdyż znaczenie zapachu jest stałą (choć zaszumioną) cechą w środowisku. Nauczenia wymagają natomiast znaczenia kolorów. Mogą one być wykształcone poprzez uczenie nie-

nadzorowane, np. regułą Hebb'a:

$$\Delta w = \eta x y \ . \tag{2.1}$$

Przyrost wagi jest tu proporcjonalny do iloczynu aktywności neuronu aferentnego (wejściowego, x) i eferentnego (wyjściowego, y). Uczenie to działa w następujący sposób. Z początku akcja jedzenie jest podejmowana wtedy, gdy zapach jest słodki i bez względu na kolor. Po każdej takiej akcji waga połączenia pomiędzy kolorem zjedzonego glona a akcją „jedzenie" wzrasta, zaś w przypadku drugiego koloru maleje. W ten sposób tworzy się asocjacja pomiędzy słodkim zapachem i kolorem jadalnych glonów, która zwiększa skuteczność odżywiania.

W drugim wariancie mamy do czynienia z uczeniem ze wzmocnieniem. Przyjmijmy, że zamiast zapachu glonów, agent może percepować swoje samopoczucie po spożyciu. I tak po spożyciu glonu jadalnego, agent czuje się dobrze, a po spożyciu trującego nie najlepiej. I jest tak niezależnie od bajora, w którym się znalazł. Uczenie nienadzorowane na nic by się tutaj zdało, gdyż informacja o jadalności glonu podana jest dopiero po jego spożyciu, więc nie może być wykorzystana do podjęcia decyzji przed jego spożyciem. Jednakże może być wykorzystana do ustalenia, jakiego koloru glony są jadalne w ogólności. Sieć, która to realizuje przedstawiona jest na rysunku 2.1b. Adaptacja ewolucyjna polega tu na ustaleniu, że dobre samopoczucie to wzmocnienie dodatnie, a złe ujemne. Jakkolwiek brzmi to trywialnie, bez adaptacji ewolucyjnej takie rozstrzygnięcie nie byłoby możliwe. Sygnał samopoczucia wpływa tu na przebieg uczenia. Wagi połączeń plastycznych modyfikowane są zgodnie z regułą Hebb'a ze wzmocnieniem:

$$\Delta w = \eta r x y \ , \tag{2.2}$$

gdzie r jest właśnie samopoczuciem agenta po spożyciu glonu.

Rysunek 2.1: Rozwiązania problemu żarłoków. Sieć uczona w sposób nienadzorowany (a) oraz ze wzmocnieniem (b). Połączenia oznaczone linią ciągłą są zakodowane genetycznie, linią przerywaną natomiast są plastyczne, tj. podlegają uczeniu.

W powyższych dwóch wariantach przykładu widać jaka jest rola ewolucji i uczenia, oraz że kombinacja obydwu adaptacji przynosi największą korzyść.

Założono przy tym, że genetycznie zakodowane mogą być niektóre wagi połączeń w sieci, inne natomiast podlegają uczeniu poprzez z góry przyjęty algorytm uczenia. Trudno jednak oczekiwać, aby mechanizmy uczenia zostały organizmom biologicznym nadane „z góry". Gdyby więc miała być zachowana zgodność przykładu z biologią, również i algorytm uczenia musiałby wyłonić się w drodze ewolucji.

2.2 Ewolucyjne sztuczne sieci neuronowe

Sposobów na powiązanie SSN i AE jest bardzo wiele, i wiele ich zostało jak dotąd zaproponowanych w literaturze. Główna linia podziału pomiędzy metodami dotyczy doboru tych elementów architektury sieci, które podlegają ewolucji, tych, które podlegają uczeniu oraz tych ustalonych *a priori*. Najczęściej adaptacji ewolucyjnej podlegają wybrane tylko elementy architektury sieci i algorytmu jej uczenia. Mogą to być wagi połączeń[2], lub tylko ich wartości początkowe. Również elementy topologii, takie jak liczba warstw czy liczba neuronów w poszczególnych warstwach (Rocha et al., 2007). Czasem rozpatrywana jest kwestia doboru optymalnych funkcji aktywacji (Augusteijn and Harrington, 2004) (choć niekoniecznie w kontekście ewolucji (patrz Duch and Jankowski, 2001)). Nieco rzadziej proponuje się objęcie ewolucją topologii w sposób zupełnie nieograniczony (Gruau, 1994a; Stanley, 2007) albo algorytmu uczenia (Chalmers, 1990; Bengio et al., 1994; Radi and Poli, 2003). Jeszcze inne podejście polega na ograniczeniu przestrzeni możliwych architektur, przede wszystkim poprzez narzucenie więzów na topologię i następnie wykorzystaniu AE do znalezienia najlepszej konfiguracji (Jung and Reggia, 2006). Próby wykorzystania AE, a konkretnie PG, do ewolucji ważonych topologii sieci podejmowane były również przez autora niniejszej pracy (Suchorzewski, 2008a,b). W końcu, zastosowanie adaptacji ewolucyjnej w odniesieniu do wszystkich aspektów architektury sieci pozostaje rzadko podejmowanym wyzwaniem. Po pierwsze ze względu na złożoność wymaganej do tego reprezentacji genetycznej, a po drugie ze względu na ograniczoną moc obliczeniową, jaka byłaby niezbędna do poszukiwania rozwiązań w przestrzeniach o tak wielkiej liczbie wymiarów. Bardziej wnikliwe wprowadzenie do problematyki ESSN w literaturze krajowej można znaleźć w (Kwaśnicka, 2007).

Wykorzystanie ewolucji do syntezy sieci, jakkolwiek brzmi naturalnie, jest problematyczne. Problem ten polega przede wszystkim na opracowaniu odpowiedniej reprezentacji genetycznej sieci, tj. sposobu kodowania i dekodowania sieci (Balakrishnan and Honavar, 2001a). Rozróżnia się przy tym dwa ogólne rodzaje reprezentacji: bezpośrednie oraz rozwojowe. W reprezentacji bezpośredniej, genotyp stanowi swego rodzaju odbitkę fenotypu sieci. Choć

[2]Tego rodzaju rozwiązania określa się czasem jako uczenie/trening genetyczny.

odwzorowanie pomiędzy genotypem i fenotypem nie musi być wzajemnie jednoznaczne, jest to przekształcenie proste, nie wymagające złożonego procesu dekodowania. Przykładem może być tutaj zakodowanie topologii sieci w postaci macierzy połączeń. W reprezentacjach rozwojowych natomiast, genotyp jest receptą, programem budowy sieci, wymagającym złożonego procesu rozwoju. Obydwa rodzaje reprezentacji mają swoje słabe i mocne strony. Reprezentacje bezpośrednie są zwykle proste i bardziej efektywne w rozwiązywaniu problemów nieregularnych, tj. takich, dla których rozwiązania nie odznaczają się żadną regularnością w postaci symetrii czy powtarzalności. Istnieje jednak interesująca klasa problemów, dla których reprezentacje bezpośrednie niechybnie zawodzą — problemy wielkoskalowe i w pewien sposób regularne. Typowym, rzeczywistym przykładem takiego problemu może być problem zamodelowania topologii mózgu ssaków, lub chociaż fragmentu zwanego korą mózgową, którego struktura kolumnowa odznacza się wysokim stopniem regularności. Szanse na uchwycenie tego rodzaju regularności skrywających się w problemach dają właśnie reprezentacje rozwojowe. Skalowalność i modularność są takimi cechami reprezentacji genetycznej, które pozwalają zakodować sieci w zwięzły sposób, uchwytując tym samym pewną regularność ich struktury. Jeśli uchwycona regularność odpowiada regularności tkwiącej w problemie, wówczas może być wykorzystana do rozwiązywania coraz bardziej złożonych instancji problemu.

Być może najtrudniejszym elementem architektury sieci do zakodowania jest algorytm uczenia, a to ze względu na trudność w określeniu przestrzeni możliwych algorytmów i odpowiednich reguł modyfikacji wag. Wczesne eksperymenty polegały na kodowaniu wartości współczynników znanych reguł uczących (Chalmers, 1990). Później wykorzystywano również PG do modyfikacji lub konstrukcji całych reguł (Bengio et al., 1994; Radi and Poli, 2003). Wyniki pokazały, iż konstrukcja bądź optymalizacja reguł uczących za pomocą AE jest możliwa, a uzyskane reguły często okazywały się efektywniejsze niż te szeroko stosowane. Jednakże tego rodzaju eksperymenty najczęściej z góry zakładają określoną architekturę sieci. Przykładowo (Abraham, 2004) zaproponował sieć jednokierunkową, dwuwarstwową, w której optymalizacji genetycznej podlegają wartości początkowe wag, liczba neuronów w warstwie ukrytej i ich funkcje aktywacji oraz rodzaj i parametry algorytmu uczącego. Ewoluowane w ten sposób sieci uzyskiwały lepsze rezultaty w zadaniach klasyfikacji aniżeli SSN o ustalonych architekturach i znanych algorytmach uczenia. Ostatnie badania w obszarze ewolucji algorytmów uczenia dotyczyły możliwości ewolucji lokalnych reguł uczenia naniesionych na topologię sieci w postaci wzorca przestrzennego (Risi and Stanley, 2010) oraz eksperymenty z neuromodulacją (Soltoggio et al., 2008).

Istotną zaletą wykorzystania ewolucji do syntezy sieci jest możliwość swobodnego określania kryterium optymalizacyjnego. Dla zadanego problemu, AE może generować rozwiązania zwięzłe, szybkie lub dokładne, w zależności od

tego, jak sformułowana została funkcja przystosowania. Poprzez modyfikacje funkcji przystosowania, możliwe jest zadawanie niemal dowolnych ograniczeń i warunków jakie powinny spełniać ewoluowane sieci. Co więcej, może to być również dowolna kombinacja tych warunków, ważona pod względem preferencji. Jeśli przykładowo sieć ma się składać z około n węzłów, funkcja przynależności może zawierać składnik zmniejszający wartość przystosowania kiedy tylko liczba węzłów w sieci odbiega od n. Dotyczy to każdego mierzalnego aspektu sieci — liczby węzłów i połączeń, ich rodzajów, funkcji przejścia i uczenia, itd. W końcu możliwe jest również wykorzystanie wielokryterialnych wariantów AE, które oferują większą kontrolę nad sposobem doboru kryteriów, są z reguły efektywniejsze w znajdowaniu rozwiązań optymalizowanych pod wieloma względami oraz umożliwiają analizę przebiegu ewolucji pod względem zadanych kryteriów (Abraham et al., 2005).

Oczywiście są również ograniczenia i wady ewolucyjnego podejścia do syntezy sieci. Przede wszystkim brak gwarancji zbieżności algorytmu, tj. znalezienia satysfakcjonującego rozwiązania, wysokie wymagania odnośnie mocy obliczeniowej, a także trudność w analizie otrzymywanych rozwiązań. Jednakże biorąc pod uwagę wysoki stopień automatyzmu metod ewolucyjnych i niewielką wiedzę potrzebną do ich zastosowania, w pesymistycznym przypadku mogą one być traktowane jako obiecujące metody alternatywne, oferujące jedynie pewną szansę na sukces.

Choć dorobek ESSN w zakresie rozwiązań praktycznych problemów jest jak dotąd skromny, zakres potencjalnych aplikacji jest równie szeroki jak generycznych SSN — od problemów statycznych przez problemy uczenia nienadzorowanego, ze wzmocnieniem i uczenia nadzorowanego w obszarze sterowania systemami dynamicznymi, a w szczególności robotami i systemami wieloagentowymi, jak również w grach, w systemach diagnostycznych i decyzyjnych, rozpoznawaniu obrazów czy predykcji szeregów czasowych. Obszary i dziedziny zastosowań są tak zróżnicowane, że zakres ten prawdopodobnie lepiej jest sformułować pod względem charakterystycznych cech problemów podatnych na rozwiązanie za pomocą ESSN:

- Mała wiedza o problemie lub zmienność problemu.
- Brak sprawdzonych metod rozwiązywania.
- Specyficzne wymagania projektowe lub ograniczenia.
- Stosunkowo niewielka złożoność problemu.

Ponadto istotnym argumentem na rzecz wykorzystania bądź wypróbowania sieci ewolucyjnych jest możliwość łatwego i efektywnego wykorzystania przetwarzania rozproszonego i równoległego.

Powyższe omówienie warto podsumować cytatem z wnikliwego przeglądu metod ESSN:

> Uczenie i ewolucja są to dwie podstawowe formy adaptacji. Poprzez wyczerpujący przegląd literatury wykazano, że kombinacja SSN i AE może prowadzić to znacznie bardziej inteligentnych systemów, aniżeli poleganie na SSN lub AE z osobna (Yao, 1999).

2.2.1 Rozwojowe reprezentacje genetyczne

Kodowanie rozwojowe odznacza się z reguły znacznie większym stopniem skomplikowania aniżeli reprezentacje bezpośrednie, jak np. reprezentacja topologii za pomocą macierzy połączeń. Jakie są więc korzyści z wykorzystania reprezentacji rozwojowej? Teoretyczne rozważania na ten temat w dość zróżnicowany sposób wyrażają motywację dla reprezentacji rozwojowych:

> Z obserwacji organizmów biologicznych możemy wywnioskować, że reprezentacja rozwojowa ma szereg korzystnych cech. Umożliwia ona zwięzły opis potencjalnie bardzo złożonych struktur. Często przytaczanym przykładem jest tutaj różnica pomiędzy rozmiarem genomu a liczbą neuronów i połączeń synaptycznych w mózgu noworodka. Zwięzłość reprezentacji po części wynika z możliwości wielokrotnego wykorzystania kodu genetycznego, określającego sposób budowy podstruktur mózgu, jak gdyby były to procedury wywoływane wielokrotnie. Biologiczny materiał dowodowy wskazuje, że reprezentacje rozwojowe umożliwiają, a wręcz faworyzują wykształcanie się struktur odznaczających się pewnym stopniem modularności i symetrii (Floreano and Mattiussi, 2008).
>
> Idea stojąca za reprezentacją rozwojową to skalowalność kodowania, która nie może być osiągnięta metodami bezpośrednimi. W reprezentacji bezpośredniej problemy wymagające dużych sieci, wymagałyby dużych chromosomów. Wynikowa przestrzeń poszukiwań byłaby bardzo duża i przez to bardzo trudne byłoby znalezienie właściwego rozwiązania przez algorytm ewolucyjny (Boers and Sprinkhuizen-Kuyper, 2001).
>
> Jedynym sposobem uchwycenia tak złożonej struktury [jak mózg] jest wykorzystanie przekształcenia z kilku wymiarów genotypu na wiele wymiarów fenotypu. (...) W rozwoju biologicznym, DNA jest mapowane na fenotyp w procesie rozwoju, który stopniowo buduje ten fenotyp. Rozwój umożliwia wielokrotne wykorzystanie genów, ponieważ ten sam gen może być uaktywniony w dowolnym miejscu i czasie podczas rozwoju. Tak więc niewielki zbiór genów może zakodować znacznie większy zbiór komponentów strukturalnych (Stanley, 2006).

Kodowanie rozwojowe nie odnosi się tylko do sztucznych sieci neuronowych czy obliczeniowych. Podobnie jak rozwój biologiczny nie dotyczy tylko układów nerwowych. Prawa i właściwości rozwoju organizmów biologicznych (embriogenezy) są bardzo podobne, jeśli nie jednakowe, dla wszystkich części ciała organizmów. Cechy, właściwości i korzyści płynące z tego rodzaju reprezentacji można więc rozpatrywać w znacznie szerszym kontekście problemów. W istocie, reprezentacje rozwojowe były stosowane dla wielu różnych problemów — „od stolików na kawę po szpitale" (Bentley, 1999; Hornby, 2004), i od układów elektronicznych po wirtualne stworzenia włącznie z ich wirtualnymi mózgami. Przegląd tego obszaru badań można znaleźć w (Stanley and Miikkulainen, 2003; Harding and Banzhaf, 2008).

Istotną cechą kodowania rozwojowego, choć nie tylko rozwojowego, jest zdolność do „kompleksyfikacji" rozwiązań (Stanley, 2006). Ewolucyjne przeszukiwanie przestrzeni rozpoczyna się z reguły od małych rozwiązań, które z czasem są rozbudowywane. Do genomu dodawane są nowe geny, które modyfikują przebieg rozwoju i poprawiają jakość rozwiązania. Ten proces pozwala ewolucji znajdować fenotypy bardziej złożone i potencjalnie lepsze, aniżeli byłoby to możliwe przy reprezentacji genetycznej o stałej długości genomu.

Założenie iż wiele złożonych i interesujących problemów jest w dużym stopniu regularnych, leży u podstaw badań nad reprezentacjami rozwojowymi. Typowym przykładem jest tutaj problem imitacji ludzkiej inteligencji, poprzez wygenerowanie struktury analogicznej do ludzkiego mózgu, która to struktura jest w dużym stopniu regularna.

2.3 Skalowalność i modularność sieci

2.3.1 Skalowalność

W ogólnej teorii systemów, system skalowalny to taki, który strukturalnie bądź funkcjonalnie może być dopasowany do różnej skali zadania. W kontekście sieci ewolucyjnych, znaczenie skalowalności mieści się w ramach znaczenia ogólnego, choć bywa rożnie formułowane. Według Harding and Banzhaf (2008) skalowalność oznacza, że „możliwe jest dodanie większej ilości neuronów, jeśli są one niezbędne do rozwiązania coraz trudniejszych problemów". Hornby (2005) oraz Lipson (2007) pojmują skalowalność przez pryzmat regularności, modularności i hierarchii — cech charakterystycznych dla systemów skalowalnych. Stanley i in. (Stanley, 2007; Stanley et al., 2009; Gauci and Stanley, 2007) biorą skalowalność jako zdolność do syntezy sieci wielkoskalowych i ich reprezentacji w dowolnej „rozdzielczości", pozwalającej dopasować sieci do zmiennej ilości wejść oraz wyjść, możliwie bez konieczności dalszej ewolucji. Gruau (1994a) z kolei, sformułował bardziej ścisłą — i bardziej wymagającą — definicję skalowalności, według której jest to zdolność do zakodowania

sieci stanowiącej rozwiązanie problemu o zmiennej liczbie wejść, przy użyciu pojedynczego, możliwie sparametryzowanego genotypu. Blisko związane z tą definicją jest pojęcie zwięzłości, które oznacza zdolność do zakodowania sieci w zwięzły sposób, a więc do uchwycenia pewnej jej regularności. Zwięzłość przejawia się jako wolniejszy wzrost rozmiaru genotypu rozwiązania w stosunku do jego fenotypu, wraz ze wzrostem rozmiaru problemu.

W ujęciu niniejszej pracy, skalowalność oznacza zdolność do odzwierciedlenia regularności problemu w fenotypie sieci i uchwycenia tej regularności w jej genotypie, i dzięki temu efektywniejsze rozwiązywanie problemów o wzrastającym rozmiarze. Regularność problemu można tutaj rozumieć jako regularność jego rozwiązania, tj. takich cech sieci obliczeniowej jak symetria, powtarzalność, powtarzalność z wariacją, rekurencja, gładkość czy samo-podobieństwo.[3] A zatem regularność problemu i jego rozwiązania jest swego rodzaju dualizmem (Lipson, 2007). Uchwycenie regularności występującej w problemie, a przez to w fenotypie jego rozwiązania, z reguły pociąga za sobą redukcję rozmiaru genotypu. Zwięzłe zakodowanie fenotypu, czy też jego kompresja, sprawia, że manipulacje genetyczne są również w pewnym sensie regularne i dzięki temu poszukiwanie rozwiązania jest łatwiejsze, o ile oczywiście znajduje się ono w przestrzeni sieci w jakiś sposób regularnych. Tak więc skalowalność w tym ujęciu rozkłada się na dwa aspekty: wzrost rozmiaru genotypu wraz ze wzrostem złożoności problemu (bądź rozmiaru fenotypu rozwiązania), oraz wzrost kosztu obliczeniowego, niezbędnego do znalezienia rozwiązania, wraz ze wzrostem złożoności problemu. Oczywiście należy również wziąć pod uwagę, że różne problemy mogą odznaczać się regularnością różnego rodzaju i w różnym stopniu, i przez to być w różnym stopniu podatne na zdolności do uchwycenia regularności problemu przez daną metodę.

W ujęciu bardziej formalnym, skalowalność jako zdolność do zwięzłego zakodowania sieci, w taki sposób, że dla problemu o złożoności wzrastającej wraz z n, rozmiar genotypu wzrasta wolniej niż rozmiar fenotypu, jest wówczas gdy:

$$O(|\Gamma(n)|) < O(|G(n)|) \ , \ \text{dla } n = n_0, \dots \ , \tag{2.3}$$

gdzie n oznacza rozmiar problem, np. w postaci liczby wejść, zaś $|\Gamma(n)|$ i $|G(n)|$ oznaczają rozmiary genotypu i fenotypu rozwiązań n-tych instancji problemu. Alternatywnie, skalowalność można rozpatrywać pod względem czasu dekodowania zamiast rozmiaru genotypu. Wówczas reprezentację można określić jako skalowalną, gdy spełnia identyczny warunek, przy czym zamiast $|\Gamma(n)|$ pod uwagę brany jest czas rozwoju, mierzony w instrukcjach, cyklach maszynowych, czy też inaczej zdefiniowanych jednostkach.

Jednym z najprostszych wskaźników skalowalności może być rozmiar genotypu sieci stanowiącej rozwiązanie problemu, w zależności od wymiarowości

[3] Nie chodzi tutaj o regularność w sensie jednakowej liczby połączeń dla każdego węzła, jak w „grafie regularnym”.

tego problemu:

$$S_0(n) = |\Gamma(n)| \ , \quad n = 1, 2, \ldots \ . \tag{2.4}$$

Charakter wynikowego ciągu $S_0(n)$ wskazuje jak skalowalne jest kodowanie genetyczne. Idealnie, nawet dla wzrastającego n, genotyp powinien mieć stały rozmiar, czyli $S_0(n) \in O(1)$. Jeśli rozmiar genotypu wzrasta wraz z wymiarowością problemu n, kodowanie nie jest doskonale skalowalne, lecz trudno jest powiedzieć w jakim stopniu, gdyż wiele zależy od samego problemu. Niemniej, możliwe jest porównywanie ciągów $S_0(n)$ otrzymanych dla różnych metod kodowania, w tym dla ciągu odpowiadającemu kodowaniu bezpośredniemu, który czasem może być uzyskany analitycznie.

Alternatywnie, rozmiar genotypu można mierzyć w stosunku do rozmiaru fenotypu sieci, będącej rozwiązaniem danego problemu:

$$S_1(n) = \frac{|\Gamma(n)|}{|G(n)|} \ , \quad n = 1, 2, \ldots \ . \tag{2.5}$$

Przyjmując, że złożoność problemu równoważna jest rozmiarowi jego minimalnego rozwiązania, i że znajdowane fenotypy rozwiązań są minimalne, wskaźnik względny jest bardziej uniwersalny, gdyż w pewnym stopniu uwzględnia złożoność problemu, przez co daje obraz skalowalności mniej zależny od konkretnego problemu. Ponadto, pozwala lepiej oszacować przeciętną skalowalność, jeśli zostanie uśredniony dla wielu problemów. $S_1(n)$ może oznaczać średnią liczbę instrukcji koniecznych do zakodowania jednego węzła sieci, choć oczywiście szczegóły zależą od sposobu pomiaru rozmiarów. Przykładowo, do rozmiaru fenotypu można włączyć liczbę połączeń, jako np. $2n + m$, gdzie n oznacza liczbę węzłów, zaś m liczbę połączeń.

Ostateczną miarą skalowalności jest jednak nakład obliczeniowy niezbędny do rozwiązania problemu o wzrastającym rozmiarze. Dla rozwiązań doskonale skalowalnych, takich właśnie o jakich mowa była w (Gruau, 1994a), nakład obliczeniowy jest stały względem rozmiaru problemu, tj. $E(n) \in O(1)$. Ten rodzaj skalowalności można mierzyć podobnie jak wyżej, przy czym zamiast rozmiaru genotypu, należałoby rozpatrzyć nakład obliczeniowy potrzebny do znalezienia rozwiązania:

$$S_2(n) = \frac{E(n)}{|G(n)|} \ , \quad n = 1, 2, \ldots \ , \tag{2.6}$$

zakładając, że $|G(n)|$ stanowi rozmiar minimalnego rozwiązania n-tej instancji problemu. Oczywiście zamiast $|G(n)|$ można również podstawić inną wielkość wyrażającą złożoność problemu — w ogólności $C(n)$.

Skalowalność jako cecha RKS wykazana została empirycznie, głównie na podstawie problemów parzystości, symetrii oraz uczenia nadzorowanego w zadaniu klasyfikacji (§4.1), dla których wyewoluowano rozwiązania skalowalne;

ale również w bardziej abstrakcyjnym zadaniu generowania topologii odznaczających się bezskalowością i modularnością w sensie miary przedstawionej w (Newman and Girvan, 2004) (§4.2).

2.3.2 Modularność

Z punktu widzenia ogólnej teorii sieci, sieć jest modularna, jeśli zawiera podsieci silnie połączone wewnętrznie i luźno połączone zewnętrznie, tj. pomiędzy sobą. Moduły są tutaj rozumiane jako „skupiska" węzłów.

Kashtan and Alon (2005) definiują modularność jako możliwość podzielenia systemu na jednostki działające w dużym stopniu niezależnie od siebie; niemniej, miarą modularności stosowaną w (Kashtan and Alon, 2005) jest miara pochodząca z teorii ogólnej (Newman, 2004; Newman and Girvan, 2004). W pracy (Bullinaria, 2007) modularność definiowana jest „w terminach wzorów połączeń[4] pomiędzy ukrytą i wyjściową warstwą neuronów, gdzie moduły składają się z rozłącznych podzbiorów węzłów ukrytych, dzielących wspólny wzorzec połączeń". Clune et al. (2010) przedstawiają podobne rozumienie modularności, jako „lokalizacji funkcji zawartej w obrębie jednostki, z czego wynika gęsta łączność wewnątrz jednostki i rzadka łączność na zewnątrz jednostki". Bliskie temu jest również ujęcie „modularności jako zamkniętej grupy elementów, która może być manipulowana jako całość" (Hornby, 2005), gdzie „manipulowana" z pewnością oznacza również „wielokrotnie wykorzystywana". Być może najkrótsze ujęcie podał Lipson (2007), według którego „funkcjonalna modularność to strukturalna lokalizacja funkcji".

Bardziej ścisłą definicję modularności w kontekście sieci ewolucyjnych podał Gruau (1994a): jeśli sieć G_1 zawiera wiele kopii podsieci G_2, a genotyp sieci G_1 zawiera genotyp kodujący sieć G_2 tylko raz, wówczas genotyp G_1 jest modularny. Definicja ta pozwala jednak stwierdzić tylko „modularność genotypową". W dodatku wyklucza ona sieci, w których podsieci występują tylko jednokrotnie, a które niemniej można by uznać za moduły, jeśli są „zlokalizowane", bądź wymienne czy transferowalne pomiędzy sieciami.

Wielokrotne wykorzystanie pojedynczego fragmentu kodu genetycznego w celu wygenerowania powtarzających się podsieci jest główną oznaką modularności, lecz niekonieczną. Lipson (2007) zwraca uwagę, że w systemie elementy mogą występować jednokrotnie, które niemniej są modułami ze względu na strukturalną lokalizację funkcji. Takie moduły mogą być wymieniane na inne warianty, bądź też — co szczególnie interesujące w przypadku sieci ewolucyjnych — wymieniane pomiędzy systemami, czyli osobnikami.

Biorąc pod uwagę powyższe rozważania na temat modularności sieci i systemów w ogólności, mówimy, że kodowanie genetyczne wspiera modularność, jeśli pozwala generować wiele powtarzających się w fenotypie podsieci (modułów)

[4] *Connectivity patterns*, czyli pewnej regularnej topologii.

na podstawie jednego fragmentu kodu genetycznego oraz jeśli umożliwia bądź ułatwia wymianę pomiędzy osobnikami fragmentów genotypu kodujących moduły fenotypu sieci. Wielokrotne wykorzystanie kodu oraz wymienialność kodu modułów są cechami wzajemnie uzupełniającymi się, choć w ewoluowanych sieciach niekoniecznie muszą występować łącznie. Wymienialność modułów jest cechą szczególnie interesującą w sytuacjach, kiedy wiele mniej lub bardziej podobnych do siebie problemów jest rozwiązywanych równolegle; można wówczas oczekiwać korzyści z wymiany użytecznych podsieci pomiędzy równolegle ewoluującymi populacjami.

Blisko związane z modularnością jest pojęcie hierarchii, które oznacza „rekursywne złożenie struktury i/lub funkcji" Lipson (2007). Hierarchia w systemie pozwala konstruować złożone sieci za pomocą kombinacji podsieci prostszych, a te z kolei konstruować z jeszcze bardziej prymitywnych. Elementami służącymi do budowy coraz wyższych poziomów w hierarchii są z reguły moduły. Wsparcie kodowania dla hierarchii teoretycznie umożliwia rozkład sieci na mniejsze, możliwie zagnieżdżone i wielokrotnie wykorzystane podsieci i dzięki temu efektywniejsze rozwiązywanie złożonych problemów.

Modularność i hierarchia są jawnie wspierane przez RKS, gdyż genotyp ma tutaj postać drzewa procedur. Procedury te zdolne są do generowania wielu podsieci w fenotypie, a ponadto są łatwo wymienialne pomiędzy osobnikami. To sprawia, że możliwe jest wyciągnięcie korzyści z komunikacji pomiędzy populacjami równolegle rozwiązującymi zestaw zadań. Wsparcie proponowanego kodowania dla modularności zademonstrowano na przykładzie problemu parzystości (§4.1.2) oraz problemu uczenia nadzorowanego w zadaniu klasyfikacji (§4.1.6). Ponadto w §4.1.3 przedstawiono eksperyment, w którym za pomocą RKS otrzymano modularne rozwiązania dla problemu „siatkówki", dla którego metoda HyperNEAT okazała się zawodna (Clune et al., 2010).

Modularność systemu jest postrzegana jako istotna cecha pozwalająca zwiększyć jego skalowalność, a więc stanowi składową skalowalności (Hornby, 2005; Lipson, 2007).

2.4 Metody ewolucji sieci

Pierwsze próby wykorzystania AE do syntezy sieci ograniczały się do poszukiwania wag sieci, zakodowanych genetycznie (np. Whitley et al., 1990). Sieć miała ustaloną topologię i architekturę, dzięki czemu wagi mogły być zakodowane w genomie o stałej długości i poszukiwane przy pomocy algorytmu genetycznego (kodowanie binarne) bądź programowania ewolucyjnego (kodowanie zmiennoprzecinkowe). Po tych pierwszych próbach nastąpił dalszy ciąg propozycji — próbowano ewoluować topologie sieci, same neurony (funkcje przejść lub aktywacji), reguły uczenia, jak również jednocześnie topologie oraz wagi połączeń. Prawdopodobnie w dużej mierze ze względu na dostępną ów-

cześnie niewielką moc obliczeniową, rzadko próbowano ewoluować wszystkie aspekty architektury sieci jednocześnie, a stosowane reprezentacje genetyczne były przeważnie bezpośrednie.

Przegląd wczesnych metod ewolucji sieci (sprzed roku 1999) znaleźć można w (Yao, 1999). Poniżej przedstawiamy kilka wybranych i najważniejszych metod, wykorzystujących reprezentacje rozwojowe.

2.4.1 Kodowanie Gramatyczne

Jedną z pierwszych propozycji genetycznej reprezentacji niebezpośredniej dla sieci neuronowych przedstawił Kitano (1990). Opierała się ona na systemie gramatyki formalnej, pokrewnej gramatyce znanej jako L-system (system Lindenmayer'a). Rysunek 2.2 przedstawia koncepcję tego kodowania na przykładzie sieci o topologii odpowiedniej do nauczenia się funkcji logicznej XOR. Genom kodujący sieć składa się tutaj z 5-elementowych bloków, interpretowanych jako reguły produkcyjne o stałej długości. Pierwszy element takiego bloku stanowi symbol nieterminalny (lewa strona reguły), natomiast pozostałe 4 elementy stanowią macierz symboli terminalnych bądź nieterminalnych o rozmiarze 2×2 (prawa strona reguły). Rozpoczynając od symbolu startowego, reguły są kolejno przykładane, budując tym samym macierz połączeń budowanej sieci. Wygenerowana sieć może być następnie uczona algorytmem wstecznej propagacji błędów.

Eksperymenty na problemie koder/dekoder wykazały, że zaproponowany system generował sieci uzyskujące lepsze rezultaty, aniżeli sieci ewoluowane przy pomocy reprezentacji bezpośredniej. Co więcej, zaproponowana reprezentacja odznaczała się skalowalnością, w tym przypadku rozumianą jako zdolność do efektywniejszego rozwiązywania problemu o wzrastającej wymiarowości. Na rysunku 2.3 przedstawiono wykres zaczerpnięty z (Kitano, 1990), na którym porównano błąd sieci (*total sum-squared*, TSS) w zależności od rozmiaru sieci, dla dwóch rozmiarów problemu i dla obydwu reprezentacji. Można zaobserwować, że w przypadku kodowania Kitano, zdolność do rozwiązywania problemu nie maleje wraz z jego wymiarowością, choć wyniki te zostały częściowo zakwestionowane w późniejszych badaniach (patrz Yao, 1999).

2.4.2 Kodowanie Komórkowe

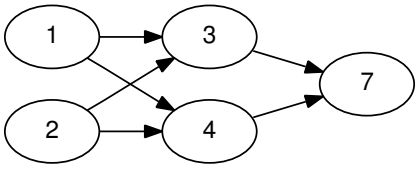
Innym szeroko znanym sposobem reprezentacji sieci neuronowych było zaproponowane w ramach pracy doktorskiej *Cellular Encoding* (CE) (Gruau, 1994a). Proces rozwoju jest tu wzorowany na rozwoju biologicznym, w którym komórki ulegają podziałom i różnicowaniu. Genotyp sieci jest pewnym programem o strukturze drzewa, którego węzły stanowią instrukcje przekształcające rozwijaną sieć. Począwszy od stanu, w którym pojedynczy neuron ukryty połączony jest ze wszystkimi wejściami i wyjściami, sieć jest budowana na pod-

$$SABCD\ Acpac\ Baaae\ Caaaa\ Daaad\ a0000\ b0001\ c1000\ p0101\ e1111$$

$$S \to \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad A \to \begin{bmatrix} c & p \\ a & c \end{bmatrix}, \quad B \to \begin{bmatrix} a & a \\ a & e \end{bmatrix}, \quad C \to \begin{bmatrix} a & a \\ a & a \end{bmatrix}, \quad D \to \begin{bmatrix} a & a \\ a & d \end{bmatrix},$$

$$a \to \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad b \to \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad c \to \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad p \to \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \quad e \to \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$S \longmapsto \begin{bmatrix} A & B \\ C & D \end{bmatrix} \longmapsto \begin{bmatrix} c & p & a & a \\ a & c & a & e \\ a & a & a & a \\ a & a & a & b \end{bmatrix} \longmapsto \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

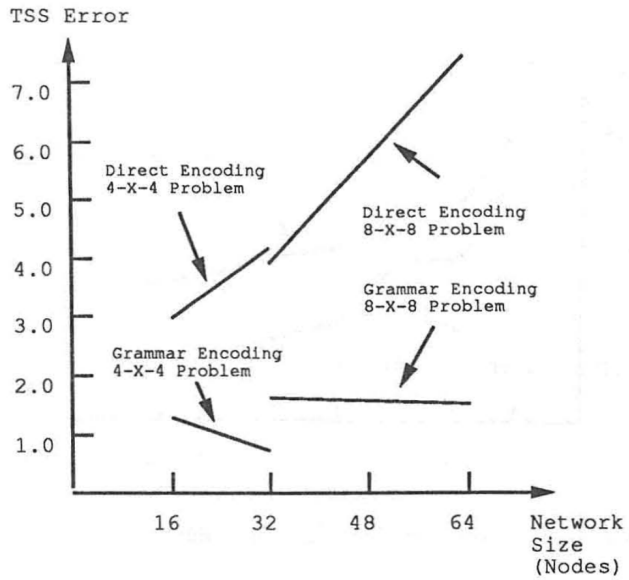


Rysunek 2.2: Genotyp (u góry), reguły przekształceń zakodowane w genotypie (poniżej), proces rozwoju w trzech krokach i wynikowy fenotyp sieci (u dołu).

stawie programu-genotypu. Program wykonywany jest od strony korzenia w kierunku liści. Każde rozgałęzienie w drzewie odpowiada podziałowi neuronu w sposób zgodny z instrukcją w danym węźle rozgałęzienia. Oprócz instrukcji podziału występują też inne instrukcje, dokonujące manipulacji na połączeniach neuronów.

Rysunek 2.4 przedstawia przykład genotypu, kolejne fazy rozwoju i otrzymany fenotyp sieci. Rozwój rozpoczyna się od sieci początkowej, składającej się z jednego neuronu połączonego z wejściami i wyjściami. W pierwszym kroku rozwoju, neuron ukryty wykonuje instrukcję z korzenia drzewa (oznaczoną tutaj jako 1) i w rezultacie dzieli się szeregowo. Wynikowy neuron poprzedzający (tj. od strony wejścia neuronu macierzystego) rozwija się następnie zgodnie z lewym poddrzewem, natomiast neuron następujący rozwija się według prawego poddrzewa. Aby poprawić czytelność przykładu neurony oznaczono zapisem ab:c, który oznacza, że dany neuron wykonał instrukcje ab i w następnym kroku wykona instrukcję c. Instrukcje END są tylko formalnością.

Oczywiście nie wszystkie mechanizmy i rozwiązania zostały na powyższym

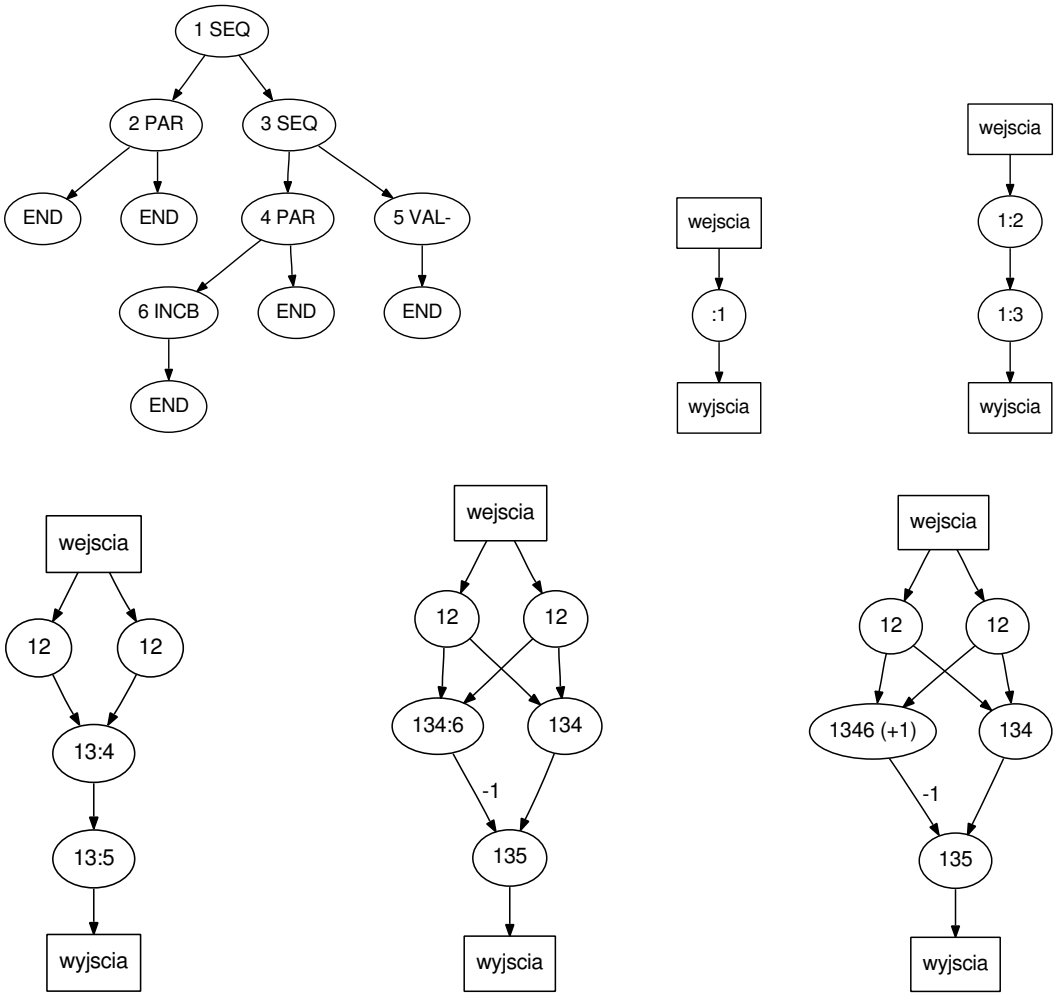


Rysunek 2.3: Porównanie skalowalności reprezentacji bezpośredniej (*direct encoding*) i kodowania gramatycznego (*grammar encoding*). Wykres przedstawia błąd sieci w zależności od jej rozmiaru (Kitano, 1990).

przykładzie przedstawione. Jednym z kluczowych mechanizmów jest „przycinanie" połączeń. Każdy neuron wyposażony jest we wskaźnik połączeń, który podczas tworzenia neuronu ustawiony jest na pierwsze połączenie dochodzące. Może on być inkrementowany i dekrementowany, a połączenie może być zerwane przy pomocy instrukcji CUT. Zasadniczo brak jest natomiast możliwości nawiązywania nowych połączeń. Choć teoretycznie nie stanowi to żadnego ograniczenia możliwości ekspresji, są podstawy by sądzić, że takie rozwiązanie nie sprzyja „ewolucyjności" kodowania. Dwie sieci różniące się zaledwie jednym połączeniem, czyli znajdujące się bardzo blisko siebie w przestrzeni fenotypów, mogą wymagać znacząco różnych genotypów, tak więc mapowanie z przestrzeni genotypów do fenotypów nie ma lokalnego charakteru, lub innymi słowy przestrzenie te są słabo skorelowane (patrz Rothlauf, 2006). Ta cecha utrudnia gładką eksplorację przestrzeni poszukiwań i sprzyjać może utykaniu algorytmu w lokalnych optimach.

Zaletą CE jest możliwość wykorzystania PG jako AE, włącznie z wykorzystaniem standardowych operatorów genetycznych dla drzew, takich jak mutacja i krzyżowanie. CE i PG stanowią razem przykład tzw. rozwojowego PG.

Przykład z rysunku 2.4 przedstawia jedynie koncepcję CE i nie wyczerpuje całego potencjału tej reprezentacji. W pracy Gruau'a przedstawiono wiele różnych instrukcji i wykazano właściwości odpowiednio dobranych zbiorów instrukcji. Wykazano, że kodowanie umożliwia wygenerowanie dowolnej sieci rekurencyjnej (kompletność), oraz że każdy genotyp koduje jakąś sieć, tj.



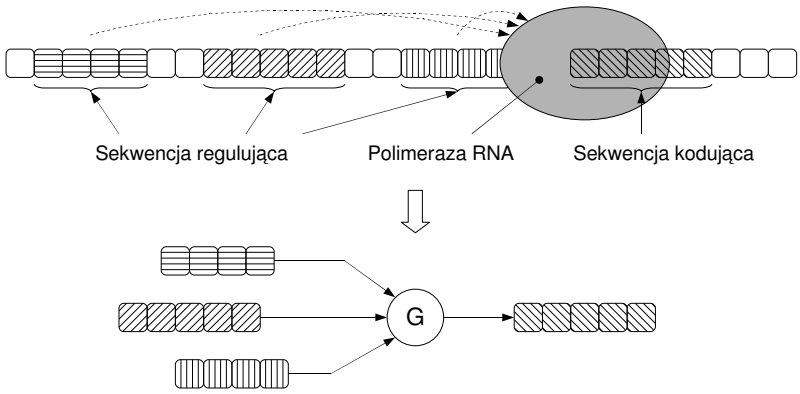
Rysunek 2.4: Przykład genotypu (z lewej u góry), sieć początkowa (środek u góry) oraz kolejne kroki jej rozwoju.

poprawność lub zamkniętość kodowania. Wykazano również, iż wykorzystując instrukcje skoków warunkowych można generować sieci charakteryzujące się modularnością i skalowalnością. Za pomocą tej metody wygenerowano sieci rozwiązujące problem parzystości dla 21 wejść, symetrii dla 40 wejść oraz problem 7-wejściowego dekodera. W późniejszym czasie wyewoluowano również sieci sterujące lokomocją robota wielonożnego (Gruau, 1994b; Gruau and Quatramaran, 1997).

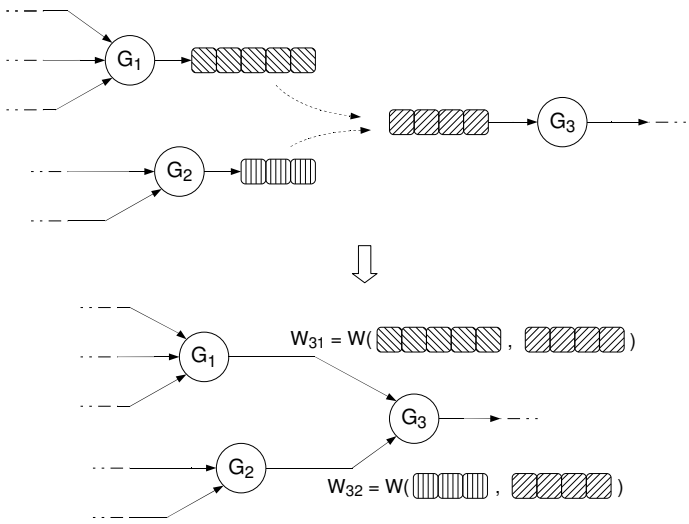
2.4.3 Analogiczne Kodowanie Genetyczne

Nowszą propozycją reprezentacji genetycznej dla sieci, niekoniecznie neuronowych, jest *Analog Genetic Encoding* (AGE) przedstawione w (Mattiussi, 2005; Mattiussi and Floreano, 2007). W kodowaniu tym położono szczególny

nacisk na zgodność biologiczną i choć zgrubne odzwierciedlenie podstawowych mechanizmów regulacji genetycznej, ekspresji genów i rozwoju komórkowego. Genotyp ma tutaj postać ciągu znaków o zmiennej długości, w którym zakodowane są neurony (lub ogólniej „urządzenia") i wagi połączeń pomiędzy nimi. AGE koduje sieci na wzór genetycznej sieci regulacyjnej (*Genetic Regulatory Network*), w której każde urządzenie jest reprezentowane przez gen, a wagi połączeń pomiędzy nimi są zdeterminowane przez interakcje pomiędzy odpowiednimi fragmentami ich „sekwencji nukleotydowych". Koncepcja ta przedstawiona jest na rysunkach 2.5 i 2.6.



Rysunek 2.5: Koncepcja genu w AGE.



Rysunek 2.6: Sieć generowana w wyniku interakcji pomiędzy genami.

Za pomocą AGE wyewoluowano kilka układów elektronicznych, takich jak

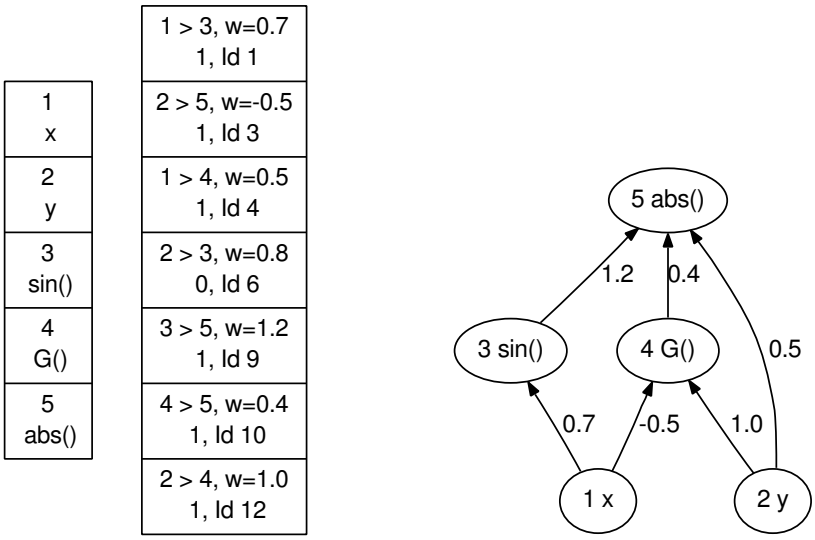
czujniki temperatury czy układ rozwiązujący problem XOR. W pracy (Soltoggio et al., 2007) wygenerowano również sieć zdolną do uczenia się dla problemu niestacjonarnego — symulowanej pszczoły uczonej ze wzmocnieniem, choć w tym przypadku rozwiązanie w dużym stopniu polegało na szczególnej wiedzy o problemie.

Choć AGE inspirowany jest procesami genetycznymi zachodzącymi w organizmach biologicznych, zupełnie nie odwzorowuje sposobu w jaki powstają biologiczne sieci neuronowe. Kodowanie to wprawdzie odzwierciedla w pewnym stopniu genetyczną sieć regulacyjną, jednak w naturze ta sieć służy koordynacji budowy sieci neuronowej, tymczasem tutaj jest ona siecią wynikową. Każdemu węzłowi (urządzeniu) w fenotypie sieci odpowiada osobny gen, a więc jest to reprezentacja w dużym stopniu bezpośrednia. Siłą rzeczy taka reprezentacja jest nieskalowalna. Łatwo sobie wyobrazić, że wraz ze wzrostem liczby wymaganych urządzeń w sieci, proporcjonalnie wzrasta liczba genów i kwadratowo wzrasta liczba interakcji pomiędzy genami. Reprezentacja ta nie odznacza się również modularnością. Jak przyznają sami autorzy, skalowalność kodowania pozostaje zagadnieniem do dalszych badań.

2.4.4 HyperNEAT

Inaczej niż w przypadku AGE, skalowalność od początku była głównym celem w pracach nad metodą *Hypercube-based NeuroEvolution of Augmented Topologies* (HyperNEAT) (Stanley, 2007). Interesujące, że metoda ma na celu możliwość wyrażania regularnych wzorców, tak charakterystycznych dla organizmów biologicznych podlegających rozwojowi, lecz opiera się przy tym na koncepcji, która od naturalnych procesów rozwoju odbiega dość daleko. Jak twierdzą autorzy, metoda zapewnia skalowalność w sposób globalny, nie próbując odzwierciedlić rozwoju biologicznego, który polega na interakcji pomiędzy komórkami.

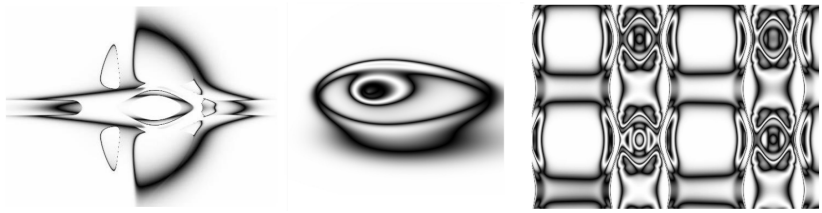
Główna idea metody HyperNEAT polega na tym, aby sieci reprezentować za pomocą pewnej funkcji złożonej, która również ma postać sieci, zwanej *Compositional Pattern Producing Network* (CPPN). Genotyp sieci CPPN i operatory dokonujące na nim wariacji zostały zapożyczone z nieco starszej metody ewolucji sieci — NEAT (Stanley and Miikkulainen, 2002). Tak więc zamiast generować sieci rozwiązujące zadany problem bezpośrednio, używa się metody NEAT do wygenerowania sieci, które będą generować sieci, które będą rozwiązywać zadany problem. Rysunek 2.7 przedstawia przykład sieci CPPN i jej reprezentację genetyczną. Genotyp takiej sieci składa się z dwóch list — węzłów i połączeń. W ogólności możliwe jest swobodne zdefiniowanie struktury elementów obydwu list i przypisanie dowolnych atrybutów zarówno węzłom jak i połączeniom. I tak węzły mają przyporządkowaną funkcję przejścia, zaś połączenia indeksy połączonych węzłów, wagę, informację o stanie aktywności czy pewien identyfikator, który wykorzystywany jest podczas reprodukcji.



Rysunek 2.7: Genotyp (z lewej) i fenotyp sieci CPPN.

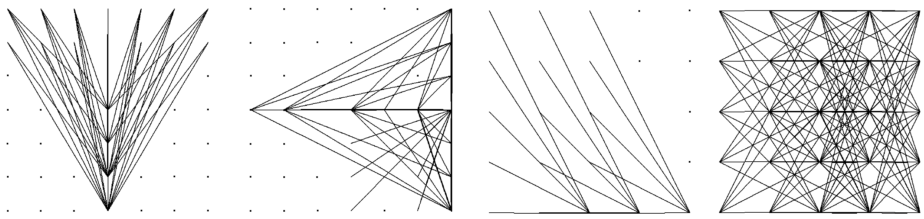
NEAT jest metodą ewolucji topologii sieci, w której każdy neuron i każde połączenie jest reprezentowane z osobna w genotypie. Jest to więc w dużym stopniu reprezentacja bezpośrednia, która nie odznacza się skalowalnością. Uzyskuje ona dobre wyniki w syntezie sieci neuronowych dla problemów nie wymagających skalowalności, takich np. jak sterowanie podwójnym odwróconym wahadłem (Stanley and Miikkulainen, 2002) czy sterowanie agentami w grach komputerowych (Stanley et al., 2005). Co jednak istotniejsze, jak pokazują wyniki działania metody HyperNEAT, uzyskuje interesujące wyniki w syntezie sieci kodujących sieci skalowalne.

Inaczej niż typowe SSN ewoluowane za pomocą NEAT, sieci CPPN zbudowane są z odpowiednio dobranych, charakterystycznych funkcji, odznaczających się symetrią czy okresowością, takich jak wartość bezwzględna, funkcja Gauss'a, sigmoida, sinus, czy funkcja piło-kształtna. Złożenie tych funkcji daje w efekcie funkcję odznaczającą się pewną regularnością, a jednocześnie pewną — co oczywiste — złożonością. Ewolucja sieci CPPN rozpoczyna się od sieci stosunkowo prostych i podlega stopniowej kompleksyfikacji. Rozmiar sieci CPPN, i tym samym złożoność wyrażanych przez nie funkcji, może być w pewien sposób ograniczony (regularyzacja), bądź też wzrastać w nieskończoność, tym samym zapewniając zdolność do wyrażenia zupełnie dowolnej funkcji. Ta prosta idea została najpierw sprawdzona w zadaniu interaktywnej ewolucji obrazów 2D (Stanley, 2006). Jak się okazało, nieregularna sieć, której węzły wykonują pewne proste funkcje, może reprezentować obrazy, które odznaczają się symetrią, częściową symetrią, powtarzalnością motywu z wariacją i bez — czyli geometrycznymi regularnościami, potencjalnie użytecznymi przy rozwiązywaniu złożonych problemów (rys. 2.8).



Rysunek 2.8: Przykłady obrazów wygenerowanych przez interaktywnie wyewoluowane sieci CPPN (Stanley, 2006).

Tak więc fenotyp w reprezentacji wykorzystywanej przez NEAT jest zarazem genotypem w kodowaniu używanym w HyperNEAT. Aby uzyskać fenotyp w ramach HyperNEAT, potrzebne jest dokonanie odpowiednich przeliczeń, czyli swego rodzaju rozwój. Rozwój ten ma całkiem inny charakter niż rozwój biologiczny czy rozwój w większości innych reprezentacji rozwojowych. Jeśli dany jest genotyp w postaci sieci CPPN i w przestrzeni rozłożone są w pewien sposób neurony, to dla każdej pary neuronów oblicza się wagę połączenia pomiędzy nimi poprzez przeliczenie sieci CPPN dla współrzędnych obydwu neuronów. Dla przestrzeni N-wymiarowych, sieć CPPN ma więc $2N$ wejść. Stąd wynika oczywista złożoność obliczeniowa rozwoju — $O(n^2)$, gdzie n oznacza całkowitą liczbę neuronów w fenotypie. Rysunek 2.9 przedstawia przykłady topologii wyewoluowanych za pomocą HyperNEAT.



a) symetria b) częściowa symetria c) repetycja d) repetycja z wariacją

Rysunek 2.9: Przykłady regularnych topologii wyewoluowanych za pomocą HyperNEAT (Stanley et al., 2009).

Oryginalna wersja HyperNEAT posiada dość poważne ograniczenia. Przede wszystkim wymaga z góry ustalonej liczby neuronów i ich rozmieszczenia w przestrzeni. Nie uwzględnia również możliwości ewolucji innych elementów architektury, takich jak funkcje aktywacji neuronów, rodzaje połączeń czy algorytm uczenia. Jednakże bazując na podstawowej koncepcji metody, tj. reprezentacji sieci za pomocą funkcji złożonej, łatwo sobie wyobrazić, że w istocie dowolne atrybuty zarówno węzłów jak i połączeń mogą być zakodowane na tej samej zasadzie co topologia sieci. W istocie tego rodzaju rozszerzenia zostały zaproponowane — możliwość adaptacji liczby neuronów i ich rozłożenia w

(Risi et al., 2010), zaś ewolucja plastyczności połączeń w (Risi and Stanley, 2010).

Eksperymenty z metodą HyperNEAT wykazały jej skalowalność na problemach symulowanego robota poszukującego pożywienia i wizualnej dyskryminacji obiektów przy zmiennej rozdzielczości — zadania, w którym liczba połączeń w sieci sięgała 8 mln (Gauci and Stanley, 2007; Stanley et al., 2009). Wygenerowano również sterownik lokomocji dla symulowanego robota wielonożnego (Clune et al., 2009), sieci zdolne do adaptacji w symulowanym środowisku wieloagentowym w zadaniu zdobywania pożywienia oraz sprzątania pomieszczeń (D'Ambrosio and Stanley, 2008; D'Ambrosio et al., 2010). Podejmowano również próby ewolucji sieci grających w warcaby (Gauci and Stanley, 2010) i sterujących symulowanym pojazdem w uproszczonym środowisku (Drchal et al., 2009).

Kodowanie wykorzystane w metodzie HyperNEAT — sieci CPPN — znajdują również ciekawe zastosowanie jako reprezentacja obrazów 2D w projekcie *Picbreeder*[5] (Secretan et al., 2011). Projekt ten polega na kolaboratywnym, interaktywnym ewoluowaniu wizualnych obrazów przez społeczność internetową. Równocześnie stanowi pewien eksperyment nad możliwością zastosowania AE w całkiem nowym kontekście, w którym ocena i selekcja jest dokonywana przez rozproszoną grupę użytkowników, zaś jedyną rolą AE jest dokonywanie reprodukcji z wariacją.

2.4.5 Podsumowanie

Ewolucyjne sztuczne sieci neuronowe, lub inaczej neuroewolucja, jest interesującym i obiecującym sposobem budowy uniwersalnych programów, łączących w sobie adaptację ewolucyjną oraz uczenie. Jednym z najważniejszych wyzwań neuroewolucji jest problem znalezienia „solidnej" reprezentacji genetycznej, która umożliwiłaby ewolucję coraz bardziej złożonych rozwiązań dla coraz bardziej złożonych zadań. Poszukiwania takiej reprezentacji coraz częściej wkraczają w obszar systemów generatywnych i rozwojowych, czerpiąc inspiracje z biologii rozwoju i embriologii.

W przeciągu ostatnich 20 lat zaproponowano wiele reprezentacji rozwojowych, niekoniecznie związanych z sieciami neuronowymi, których zadaniem jest uchwycenie regularności w podejmowanych problemach i przez to efektywniejsze ich rozwiązywanie (patrz Stanley and Miikkulainen, 2003; Floreano and Mattiussi, 2008; Harding and Banzhaf, 2008). Nasuwa się przy tym pytanie, jakie motywy skłaniają badaczy do dalszych poszukiwań metody syntezy sieci, w sytuacji kiedy istnieją metody, dla których istotne i pożądane właściwości wykazano teoretycznie, a w pewnym stopniu również praktycznie? Odpowiedzi trudno jest znaleźć w literaturze. Próby i eksperymenty z tego ro-

[5] http://www.picbreeder.org

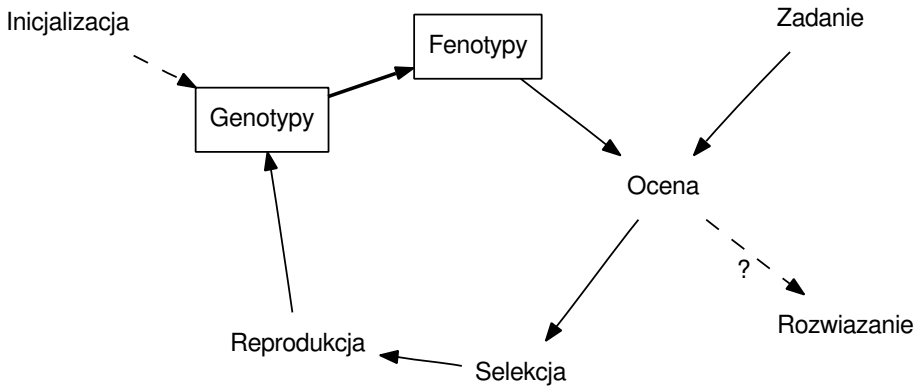
dzaju metodami wskazują, że przyczyny mogą być prozaiczne — działają one dla pewnych problemów, a dla innych zawodzą. Istotną przeszkodą w badaniach nad metodami wykorzystującymi reprezentacje rozwojowe jest trudność w zmierzeniu ich efektywności. Złożoność tych metod, zarówno obliczeniowa jak i programowa, praktycznie uniemożliwia ich definitywne porównywanie. Najlepszym, a może wręcz jedynym sposobem sprawdzenia użyteczności takiej metody jest demonstracja generowanych przez nią rozwiązań — im dłuższa i bardziej zróżnicowana jest lista rozwiązanych problemów, tym większe uznanie dla praktycznych możliwości danej metody i jej uniwersalności, tak jak to jest w przypadku PG (Koza, 2010).

Rozdział 3

Rozwojowe Kodowanie Symboliczne

Kluczowym komponentem metody RKS jest reprezentacja genetyczna sieci wraz z operatorami genetycznymi i określonym sposobem jej rozwoju; a więc kodowanie, modyfikacja i dekodowanie sieci. Jako metoda ewolucyjna, RKS opiera się na pewnym algorytmie ewolucyjnym (AE). AE wykorzystany w metodzie jest dość typowy i poza pewnym powiązaniem z reprezentacją genetyczną oraz doborem kilku dodatkowych technik i mechanizmów, nie stanowi innowacyjnego elementu metody. W istocie takie elementy AE jak selekcja, reprodukcja, struktura populacji czy ocena przystosowania są w dużym stopniu standardowe i od siebie niezależne. Dla koncentracji uwagi na innowacyjnym składniku metody, czyli reprezentacji genetycznej, wykorzystujemy jeden ustalony AE, dający jednak możliwość wykorzystania pewnych opcji. Wczesny wariant metody RKS był opisany w (Suchorzewski, 2010b).

Opis AE wykorzystywanego w pracy przedstawiono w §3.1. W kolejnych punktach przedstawiono model fenotypu sieci (§3.2), sposób kodowania (§3.3) i dekodowania (§3.4) sieci, sposoby modyfikacji genotypu (§3.5) oraz najważniejsze właściwości metody (§3.6).



Rysunek 3.1: Ogólny schemat algorytmu ewolucyjnego.

3.1 Algorytm ewolucyjny

Schemat ogólnego algorytmu ewolucyjnego wykorzystującego reprezentację rozwojową przedstawiono na rysunku 3.1. Algorytm rozpoczyna się inicjalizacją populacji genotypów, po czym następują cyklicznie powtarzane kroki — dekodowanie genotypów do postaci fenotypów, ocena fenotypów na podstawie zadania i określonej funkcji przystosowania, sprawdzenie warunku końcowego, tj. czy znaleziono rozwiązanie lub czy została osiągnięta maksymalna liczba pokoleń, następnie dokonywana jest selekcja osobników i ich reprodukcja z wariacją, tj. mutacją i krzyżowaniem.

Algorytm 1 przedstawia bardziej szczegółowo AE wykorzystywany w metodzie RKS. Być może najbardziej szczególny w stosunku do standardowego algorytmu, wykorzystywanego np. w programowaniu genetycznym (Poli et al., 2008), jest etap reprodukcji i zastosowanie operatorów genetycznych. Przyjęto, że najpierw aplikowany jest operator krzyżowania „jednopotomnego" z prawdopodobieństwem p_c (zazwyczaj 0.1), a następnie w sposób niezależny operator mutacji z prawdopodobieństwem p_m (0.8). Pozostałe kroki algorytmu są standardowe, przy czym stosuje się selekcję turniejową o stałej wielkości turnieju $k = 5$. Warunkiem końcowym jest najczęściej ustalona liczba pokoleń, jednak możliwe jest również przerwanie ewolucji w momencie znalezienia satysfakcjonującego rozwiązanie.

Warto zwrócić uwagę na możliwość rozwiązywania **zadań stochastycznych**, kiedy to nieco inaczej dokonuje się wyłonienia rozwiązania — w przypadku zadania stochastycznego zazwyczaj określony jest "testowy" wariant funkcji przystosowania, który ocenia rozwiązanie znacznie dokładniej niż normalnie. Ponadto, aby uniknąć „obciążenia selekcyjnego" w przypadku zadań stochastycznych, najlepszy osobnik z całego przebiegu ewolucyjnego poddawany jest testowi ponownie i dopiero ta wartość przystosowania jest przyjmowana

```
Inicjalizuj populację M genotypów
Pokolenie t := 0
Powtarzaj
  Jeśli zadanie stochastyczne wtedy
    Odśwież zadanie
  Dla każdego osobnika z populacji wykonuj
    Rozwiń sieć
    Oceń sieć (minimalizacja funkcji przystosowania)
  Jeśli zadanie stochastyczne wtedy
    Testuj najlepszego osobnika w pokoleniu
  Jeśli znaleziono osobnika z rekordowym przystosowaniem wtedy
    Zapisz rekordowego osobnika
  Jeśli spełniony jest warunek zakończenia wtedy
    Przerwij pętlę
  Jeśli model wyspowy wtedy
    Przeprowadź migrację osobników
  {Reprodukcja}
  Dla i = 1, ..., M wykonuj
    Selekcjonuj i kopiuj osobnika jako i-tego
    Jeśli x ~ U[0;1) < p_c wtedy
      Selekcjonuj partnera
      Skrzyżuj osobnika z partnerem
    Jeśli x ~ U[0;1) < p_m wtedy
      Mutuj osobnika
    Wprowadź i-tego osobnika do populacji potomnej
Zwróć rekordowego osobnika
```

Algorytm 1: Zarys algorytmu ewolucyjnego wykorzystywanego w metodzie RKS.

za wynik.

W algorytmie zakłada się minimalizację funkcji przystosowania, która uwzględnia dwie składowe: kryterium właściwe danego problemu J oraz karę za rozmiar rozwiązania[1]:

$$F(G,\Gamma) = (J(G) + r)(1 + r(2n + m + |\Gamma| + 0.5\tau), \tag{3.1}$$

gdzie G i Γ oznaczają fenotyp i genotyp ocenianej sieci, $r \in (0.0001; 0.01)$ jest współczynnikiem kary, n i m oznaczają liczbę węzłów i połączeń w sieci, natomiast $|\Gamma|$ i τ to odpowiednio rozmiar genotypu liczony w instrukcjach oraz liczba kroków rozwoju. Kara za rozmiar uwzględnia kilka aspektów sieci i jest naliczana proporcjonalnie do kryterium właściwego. Tak więc podczas ewolucji stosowana jest **regularyzacja**, w teorii AE zwana również „zwięzłością”

[1]Przy omawianiu wyników, jednakże, rozmiar rozwiązania z reguły nie jest rozpatrywany i termin „przystosowanie” odnosi się do samego kryterium właściwego.

(*parsimony*). Jeśli kryterium właściwe dla danego problemu wymaga maksymalizacji, bądź posiada nietypowy przedział wartości, jest ono przekształcane do postaci $[0; b]$, gdzie b najczęściej wynosi 1. Powyższa funkcja została przyjęta po wielu mniej lub bardziej systematycznych próbach z różnymi problemami, niemniej, jest to wybór w dużym stopniu arbitralny; co jednocześnie oznacza, że wybór innej funkcji nie powinien znacząco wpłynąć na przeciętne wyniki działania algorytmu. Ta zasada dotyczy wielu innych detali algorytmu.

Techniką często stosowaną (uaktywnianą) w RKS jest mechanizm regulacji — a właściwie promocji — różnorodności w populacji. Zasadniczo, **promocja różnorodności** polega na karaniu — czyli modyfikacji funkcji przystosowania na niekorzyść — tych osobników, które są do siebie w pewien sposób podobne. Oczywiście istnieje wiele miar podobieństwa między osobnikami i sposobów naliczania kar. Podobieństwo można obliczać na poziomie funkcji przystosowania (technika współdzielenia przystosowania *fitness sharing*), a także na poziomie genetycznym, fenotypowym oraz behawioralnym — czyli tzw. niszowanie (*niching*), specjacja (*speciation*) lub stłaczanie (*crowding*), (patrz Bäck et al., 1997). W badaniach symulacyjnych stosuje się dwie techniki: współdzielenie przystosowania oraz niszowanie na podstawie fenotypu sieci.

Współdzielenie przystosowania polega na modyfikacji funkcji przystosowania:

$$F_i' = F_i \exp(p_i), \tag{3.2}$$

gdzie F_i oznacza wartość przystosowania i-tego osobnika, zaś p_i proporcję osobników mających wartość przystosowania $\lfloor c_F F_i \rfloor$, czyli zaokrągloną w dół wartość przystosowania przemnożoną przez „współczynnik kwantyzacji" $c_F \in (0.1; 20)$.

Druga technika jest podobna, przy czym cechą na podstawie której ocenia się podobieństwo jest strukturalny „odcisk" sieci zamiast wartości przystosowania:

$$F_i' = F_i(1 + c_G\, p_i), \tag{3.3}$$

gdzie $c_G \in (0.001; 0.01)$ jest współczynnikiem, natomiast p_i oznacza proporcję osobników z takim odciskiem jak odcisk i-tego osobnika. Niszowanie na podstawie odcisku wymaga nieco większych nakładów obliczeniowych i implementacyjnych, lecz pozwala lepiej rozróżniać osobniki — sieci o identycznej wartości przystosowania mogą mieć różną strukturę, lecz (zazwyczaj) nie na odwrót. Nie będziemy szczegółowo omawiać algorytmu pobierania „odcisku" sieci, gdyż może on być zrealizowany na wiele sposobów. W skrócie chodzi o wzajemnie jednoznaczne przekształcenie $\mathbb{G} \to \mathbb{Z}$, czyli takie, aby dwie strukturalnie (i przez to również funkcjonalnie) identyczne sieci dawały w rezultacie tę samą wartość całkowitą, natomiast dwie różne sieci wartość różną. Ponieważ przestrzeń fenotypów sieci jest ogromna, a dziedzina odcisków w praktyce ograniczona, nie sposób uzyskać funkcji doskonale wzajemnie jednoznacznej; można jednak uzyskać taką, gdzie tzw. konflikty występują wystarczająco rzadko.

Ostatnim istotnym aspektem algorytmu wykorzystywanego w metodzie jest **struktura populacji**. Oprócz standardowej, pojedynczej populacji panmiktycznej, tj. takiej, w której brak jest restrykcji na selekcję osobników do krzyżowania, wykorzystuje się jeszcze dwa warianty:

- Model wyspowy (*island model*), w którym wiele populacji (demów) ewoluuje równolegle, rozwiązując to samo lub różne zadania i najczęściej komunikując się ze sobą. Możliwość wykorzystania modelu wyspowego przejawia się w 16. kroku algorytmu 1, w którym dokonuje się komunikacja pomiędzy demami.

- Ewolucja przyrostowa, gdzie kilka populacji rozwiązuje sekwencyjnie problem o wzrastającej złożoności, tak że rozwiązanie dla prostszych przypadków może ułatwić poszukiwanie rozwiązań dla kolejnych, bardziej złożonych przypadków.

Model wyspowy i ewolucję przyrostową stosuje się zwłaszcza w badaniach nad skalowalnością i modularnością RKS.

3.2 Fenotyp sieci

W teorii grafów, sieć jest to graf skierowany z krawędziami ważonymi. Przyjmując, że krawędzie nieważone to przypadek szczególny krawędzi ważonych (z wagą 1), w pracy konsekwentnie posługujemy się terminem sieć. Ponadto, mówimy o węzłach i połączeniach, jako wierzchołkach i krawędziach. Krawędzie nieważone, nazywamy połączeniami prostymi.

Sieci ewoluowane przez RKS mają następującą postać:

$$G = (\mathbf{V}, E) = ([v_1 \ \dots \ v_n], \{e_{ij}\}) \ , \tag{3.4}$$

gdzie $\mathbf{V}$ oznacza wektor węzłów, zaś E jest zbiorem połączeń. Sieć G nazywamy fenotypem, dla odróżnienia od genotypu sieci, który jest programem budowy sieci. Z powyższej definicji wynika, iż pomiędzy dwoma węzłami może istnieć tylko jedno połączenie e_{ij}, a więc mamy do czynienia z siecią prostą (lub właściwą). Węzły i połączenia mogą posiadać pewną liczbę atrybutów, takich na przykład jak symbol czy parametr. Niektóre z atrybutów pełnią rolę podczas rozwoju, inne zaś w czasie obliczeń. Węzły posiadają zazwyczaj dwa atrybuty: symbol, który pozwala identyfikować węzły w czasie rozwoju i nadaje węzłowi określoną funkcję przejścia, oraz wartość wyjściową, która ma oczywiste znaczenie podczas obliczeń. Węzeł v można więc rozumieć jako obiekt składający się z dwóch atrybutów: $v = (X, y)$. W przypadku połączeń, atrybutów może nie być w ogóle, lub być ich jeszcze więcej, w zależności od rodzaju połączenia. Przykładowo, połączenia proste nie posiadają atrybutów, połączenia ważone

posiadają atrybut 'waga', zaś połączenia plastyczne mogą posiadać dodatkowo cały zbiór parametrów.

Przyjęta reprezentacja sieci bliska jest reprezentacji standardowej, znanej z teorii grafów, jednak różni się pod kilkoma względami. Przede wszystkim, w odróżnieniu od reprezentacji standardowej, $\mathbf{V}$ nie jest zbiorem, a wektorem, a zatem strukturą uporządkowaną. Ponadto, w teorii grafów atrybut jest zapisywany jako przyporządkowanie wartości z dziedziny atrybutu do węzłów (lub połączeń), a więc stanowi pewną funkcję $V \to A$; tutaj przyjmuje się notację zorientowaną obiektowo, a mianowicie, że węzły oraz połączenia są obiektami złożonymi, zawierającymi pewną ilość wartości atrybutów, w ogólności:

$$v \ni (a_1, \ldots, a_k) \quad \text{oraz} \quad e \ni (b_1, \ldots, b_l) \ . \tag{3.5}$$

Wektor $\mathbf{V}$ dzieli się na 3 zwarte podwektory: wejściowy $\mathbf{V}_u$, ukryty $\mathbf{V}_h$ oraz wyjściowy $\mathbf{V}_y$. Wektor wejściowy i wyjściowy stanowią interfejs sieci i ich długość jest zależna od problemu, lecz stała w czasie rozwoju sieci. Wektor ukryty natomiast, może zmieniać swą długość w czasie rozwoju, kiedy to tworzone i usuwane są węzły ukryte. Przyjmuje się, że węzły wejściowe nie posiadają połączeń przychodzących, co zresztą jest uniemożliwione przez mechanizmy kodowania i dekodowania.

Rozwinięta sieć składa się z węzłów, do których poprzez symbol X przypisana jest pewna funkcja f, nazwijmy ją funkcją przejścia. Algorytm obliczeniowy sieci przedstawiony jest w algorytmie 2. Procedura obliczeniowa rozpoczyna się od inicjalizacji sieci, podczas której zerowane są wartości wyjściowe węzłów, jak również ustawiane do wartości początkowych są wszelkie zmienne stanu, takie jak na przykład wagi połączeń plastycznych. Wartości początkowe mogą mieć charakter stochastyczny. Następnie, dla każdego wektora wejściowego $\mathbf{u}$ sieć wykonuje obliczenia. Po ustaleniu wejść, wykonywane są obliczenia dla każdego węzła ukrytego i wyjściowego, w ścisłej kolejności wyznaczonej przez wektory $\mathbf{V}_h$ i $\mathbf{V}_y$. Obliczenia te są definiowane przez funkcję przejścia f, przypisaną do danego węzła v_j, w skrócie f_j. Wyrażenie $e_{ij}(y_i)$ oznacza operację wykonywaną przez połączenie (np. ważenie) i jednocześnie jedną z wartości wejściowych do węzła v_j. Jeśli w sieci występują połączenia plastyczne, są one aktualizowane według funkcji g związanej z rodzajem połączenia, gdzie B oznacza pewien podzbiór parametrów połączenia, które biorą udział w aktualizacji. Krok obliczeniowy kończy się zwróceniem wektora wyjściowego $\mathbf{V}_y$.

Funkcja przejścia może być zdefiniowana w swobodny sposób, jej stałą cechą jest tylko to, iż jest to funkcja $\mathbb{R}^k \mapsto \mathbb{R}^1$, gdzie $k \in \mathbb{N}_0$ jest liczbą wejść do węzła i w ogólności jest zróżnicowana nawet w ramach pojedynczej sieci. Tak więc f musi być dobrze określona dla dowolnej liczby wejść, włącznie z 0. Zazwyczaj f przyjmuje banalną postać sumy, ewentualnie zmodyfikowanej dodatkowo przez pewną nieliniową funkcję aktywacji, np. sigmoidę. Niemniej, bardziej ogólna definicja umożliwia zastosowanie innych operatorów agregacji,

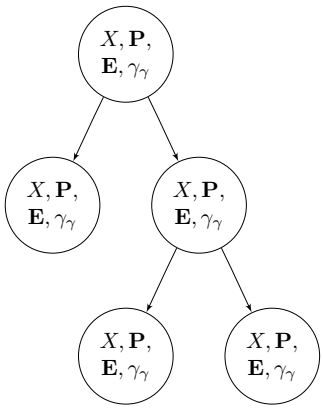
1: Inicjalizuj węzły i połączenia
2: **Dla** każdego wektora wejściowego **u wykonuj**
3: Przypisz wejścia $\mathbf{V}_u := \mathbf{u}$
4: **Dla** każdego $v_j \in [\mathbf{V}_h\ \mathbf{V}_y]$ **wykonuj**
5: Oblicz wartość wyjściową: $y_j := f_j(\{e_{ij}(y_i)\}_{i=1,\ldots,k})$
6: **Dla** każdego plastycznego połączenia e_{ij} **wykonuj**
7: Aktualizuj połączenie $e_{ij} : B = g(B, y_i, y_j)$
8: Zwróć $\mathbf{V}_y$,

Algorytm 2: Algorytm obliczeniowy sieci.

takich jak maximum czy średnia. Interesująca może być również możliwość rozszerzenia dziedziny funkcji o operacje nie-agregujące, czyli biorące pod uwagę porządek lub rodzaje połączeń wejściowych.

3.3 Genotyp sieci

Reprezentacja genetyczna sieci wraz z operatorami genetycznymi stanowi jądro metody RKS. Genotyp Γ jest programem budowy sieci. Program ten ma strukturę drzewa, którego węzły odpowiadają procedurom (rys. 3.2). Korzeń drzewa odpowiada procedurze głównej. Jest ona wystarczająca, aby rozwiązać wiele problemów testowych postawionych w pracy. Każda procedura ma tę samą strukturę wewnętrzną: $\gamma = (X, \mathbf{P}, \mathbf{E}, \gamma_\gamma)$, gdzie X stanowi identyfikator procedury, $\mathbf{P}$ jest listą instrukcji zwaną „Prolog”, $\mathbf{E}$ jest listą instrukcji terminalnych zwaną „Epilog”, natomiast γ_γ jest zbiorem podprocedur danej procedury.



Rysunek 3.2: Drzewiasta struktura programu Γ. Węzły drzewa to procedury γ.

Główną składową procedury jest Prolog. Składa się on z instrukcji operujących na sieci i rozwijających ją. Program Γ rozwija sieć z jej stanu początkowego w stan końcowy, tj. $G_\tau = \Gamma(G_0)$. Tak więc instrukcje λ mogą być

postrzegane jako operacje przekształcające sieć: $G_{t+1} = \lambda(G_t)$. Oto krótka charakterystyka najważniejszych instrukcji:

1. Con X Y C: łączy węzły X i Y połączeniem C.
2. Cut X Y: przerywa połączenie pomiędzy X i Y.
3. Cone x_* y_* C E: łączy węzły x_* i y_* spełniające warunek E połączeniem C.
4. Cute x_* y_* E: przerywa połączenie pomiędzy x_* i y_* spełniającymi warunek E.
5. Divp X Y: dzieli węzeł X równolegle, tj. duplikuje węzeł wraz z jego połączeniami i wstawia kopię za oryginałem w wektorze $\mathbf{V}$.
6. Divs X Y: jak wyżej, przy czym podział jest szeregowy, tj. kopia przejmuje połączenia wychodzące węzła oryginalnego, tworzone jest połączenie między oryginałem a kopią, która jest wstawiana za oryginałem w $\mathbf{V}$.
7. Subst X Y: podmienia symbol X na symbol Y.
8. FCall F X: wywołuje podprocedurę F dla każdego węzła X.
9. Term X Y: jak Subst, przy czym Y jest koniecznie symbolem terminalnym, tj. wyznacza funkcję przejścia.

Argumenty instrukcji, X i Y, są symbolami i służą identyfikacji węzłów w wektorze $\mathbf{V}$. W ogólności symbol X składa się z etykiety x oraz indeksu i, czyli $X = x_i$. Ponieważ wiele węzłów może mieć przyporządkowaną etykietę x, wyznacza ona niejako grupę lub warstwę węzłów. Indeks i pozwala identyfikować węzeł w warstwie.[2] Może on przybierać wartości naturalne lub też wartość specjalną, oznaczaną przez '$*$'. W takim przypadku, x_* oznacza wszystkie węzły warstwy x i umożliwia operowanie na wielu węzłach za pomocą jednej instrukcji.

Instrukcje wykonywane są dla wszystkich węzłów czy też par węzłów, które pasują do argumentów instrukcji X, Y. Symbol x_i pasuje do symbolu y_j jeśli:

$$x_i \simeq y_j \iff x = y \ \wedge \ (i = j \ \vee \ i = *) \ ; \tag{3.6}$$

tak więc relacja pasowania jest niesymetryczna.

Warunek E jest to wyrażenie algebraiczne czterech zmiennych, przeliczane dla każdej pary węzłów z dwóch wybranych warstw, tj. x_* i y_*. Wartości wejściowe do wyrażenia to: indeksy obydwu węzłów oraz najwyższe indeksy

[2] Choć w zakresie pracy używamy pojedynczego indeksu, możliwe jest uogólnienie tej notacji na wiele indeksów i tworzenie w ten sposób warstw wielowymiarowych.

z obydwu warstw. Warunek jest spełniony, kiedy $E \neq 0$. Operacje wykorzystywane w wyrażeniu są praktycznie dowolne, w pracy stosuje się zazwyczaj operacje podstawowe, takie jak: $+$, $-$, $<$, $*$, abs(), $==$ (porównanie). Wyrażenia mogą być sprawnie reprezentowane za pomocą drzew, dzięki czemu możliwe jest wykorzystanie przekształceń znanych z programowania genetycznego.

Zapis genetyczny połączenia C może przybierać różne formy, w zależności od rodzaju połączenia. Połączenia proste oznaczane są jako '1', połączenia ważone jako 'w:c', gdzie c oznacza wagę, natomiast połączenia plastyczne mogą zawierać większą liczbę parametrów (nie zawsze przedstawioną w całości). Pod względem implementacji, połączenie C jest obiektem referencyjnym, działającym jak szablon, z którego kopiowane są określone parametry dla nowych połączeń. Jest to o tyle istotne, że umożliwia łatwe manipulacje na parametrach połączenia, a więc charakterystyka połączeń również może podlegać ewolucji.

Inicjalizacja i determinacja

Inicjalizacja genotypu polega na wygenerowaniu kilku losowych instrukcji Prologu procedury głównej; choć możliwe jest również utworzenie genotypu pustego. Argumenty losowo wygenerowanych instrukcji są *niezdeterminowane*. Determinacja argumentów jest istotnym mechanizmem kodowania, wpływającym na jego efektywność. Aby wyjaśnić na czym polega determinacja i dlaczego argumenty nie mogą być wygenerowane losowo wraz z instrukcjami, trzeba wyjaśnić czym jest *symbol* i jaka jest jego rola w kodowaniu. Rozróżniamy kilka klas symboli:

1. Wejściowe, $S_u \sim \mathbf{V}_u \subset \{x_i : x = \mathsf{u}\}$.
2. Ukryte, $S_h \sim \mathbf{V}_h \subset \{x_i : x \in \{\mathsf{A}, \ldots, \mathsf{Z}\}\} \cup S_t$.
3. Nowe, $S_n \sim \{x_i : x \in \{\mathsf{A}, \ldots, \mathsf{Z}\} \ \wedge \ i = *\}$.
4. Wyjściowe, $S_y \sim \mathbf{V}_y \subset \{x_i : x = \mathsf{y}\}$.
5. Terminalne, $S_t \sim \{x_i : x \in \{+, \ -, \ *, \ \ldots\}\}$ (w zależności od problemu).
6. Identyfikatory procedur, $S_f \sim \gamma_{S_f} \subset \{x_i : x = \mathsf{f}\}$.

Wyrażenie $S_a \sim A \subset B$ oznacza „symbol klasy a jest determinowany poprzez losowanie ze zbioru (lub wektora) A, zawartego w dziedzinie B". Taka interpretacja dotyczy symboli wejściowych, ukrytych, wyjściowych oraz identyfikatorów procedur, gdzie γ_{S_f} oznacza identyfikatory ze zbioru podprocedur danej procedury. W przypadku symboli nowych i terminalnych, determinacja dokonuje się bezpośrednio na dziedzinie danej klasy. Przypomnijmy, że symbole terminalne oznaczają funkcje przejścia, pozostałe symbole są nieterminalne.

Powyższa klasyfikacja służy jasnemu określeniu, jakie są poprawne argumenty poszczególnych instrukcji. Odpowiednie zestawienie przedstawia tabela 3.1, gdzie zamiast symboli ogólnych X i Y użyto poprawnych klas symboli. Zapis S_{ab} lub $S_{a|b}$ oznacza, że dany argument może należeć do klasy S_a lub S_b. W przypadku instrukcji Cone i Cute argumenty wyboru węzłów są oznaczone '*', co oznacza, że symbol pozbawiany jest indeksu i wybór obejmuje całą warstwę węzłów.

Tabela 3.1: Instrukcje i ich poprawne argumenty.

Con S_{uh} S_{hy} C	Cut S_{uh} S_{hy}	Div* S_{uhy} $S_{h\|n}$	Subst S_{hy} $S_{h\|n\|t}$
Cone S^*_{uh} S^*_{hy} C E	Cute S^*_{uh} S^*_{hy} E	FCall S_f S_h	Term S_{hy} S_t

Kiedy nowa instrukcja jest umieszczana w programie, czy to podczas inicjalizacji czy mutacji, jej argumenty są niezdeterminowane. Determinacja argumentu dokonywana jest podczas rozwoju genotypu i polega na wylosowaniu symbolu ze zbiorów odpowiadających klasom symboli poprawnych dla danego argumentu. W przypadku S_{ab} symbol jest losowany z sumy zbiorów odpowiadających klasom a i b. W przypadku $S_{a|b}$ najpierw losowana jest klasa, z której ma pochodzić symbol, tj. S_a lub S_b, a następnie losowany jest symbol ze zbioru wylosowanej klasy. Zdeterminowany argument pozostaje zdeterminowany, dopóki nie zostanie przypadkowo zamieniony w postać niezdeterminowaną podczas mutacji, a wówczas jest ponownie determinowany podczas następnego rozwoju sieci.

Przykładowo, pierwszy argument instrukcji Con należy do klasy S_u lub S_h, a więc jest determinowany poprzez losowanie z podwektorów $\mathbf{V}_u$ i $\mathbf{V}_h$, czyli z wektora $[\mathbf{V}_u\ \mathbf{V}_h]$. Warto przypomnieć, że determinacja dokonywana jest podczas rozwoju dla takiego wektora $[\mathbf{V}_u\ \mathbf{V}_h]$ jaki jest w danym kroku rozwoju. Dlatego właśnie determinacja argumentów nie mogłaby być dokonana przed rozwojem — stan wektora $\mathbf{V}$ byłby nieznany. Kontynuując przykład, drugi argument instrukcji Divp może należeć do klasy S_h lub S_n, lecz jego determinacja odbywa się w ten sposób, że najpierw losowana jest klasa, potem zaś symbol ze zbioru odpowiadającego danej klasie, czyli $\mathbf{V}_h$ lub $\{x_i : x \in \{\mathsf{A}, \ldots, \mathsf{Z}\} \ \wedge \ i = *\}$ — po prostu nowej etykiety, która dla zwiększenia czytelności ma postać wielkiej litery, lecz równie dobrze mogłaby to być liczba całkowita z pewnego zakresu. Powodem, dla którego wprowadzono dwa rodzaje determinacji, tj. 1- i 2-etapowej, jest potrzeba utrzymania pewnej równowagi pomiędzy nowo wprowadzanymi węzłami z symbolami nowymi oraz ukrytymi, tj. tymi już występującymi w sieci.

3.4 Rozwój

Podobnie jak w *Cellular Encoding* (Gruau, 1994a), rozwój sieci rozpoczyna się od sieci początkowej G_0, zawierającej pojedynczy węzeł ukryty połączony z węzłami wejściowymi i wyjściowymi (połączeniami prostymi). Węzeł ukryty ma przyporządkowany symbol A_0. Następnie wykonywane są instrukcje Prologu, przekształcając sieć krok po kroku: $G_{t+1} = \lambda(G_t)$. Jeśli instrukcja nie zmienia stanu sieci, tj. $G_{t+1} = \lambda(G_t) = G_t$, wówczas jest traktowana jako nieefektywna i może być skasowana przy okazji użycia operatora czyszczenia. Sieć wynikowa G_τ otrzymywana jest w wyniku rozwoju, czyli wykonania programu Γ, tj. $G_\tau = \Gamma(G_0)$.

Funkcjonalna poprawność sieci G_τ jest zagwarantowana dzięki Epilogowi oraz operatorowi pokrycia (patrz §3.5). Epilog jest częścią programu wykonywaną po Prologu. Zawiera on instrukcje Term, które podstawiają symbole terminalne za nieterminalne. Jeśli po wykonaniu wszystkich instrukcji Epilogu występują w sieci węzły z nieprzyporządkowanymi funkcjami przejścia, operator pokrycia uzupełnia Epilog dodatkowymi instrukcjami Term.

Rozwój sieci może być przerwany w dowolnym momencie, na przykład w sytuacji przekroczenia maksymalnej liczby kroków rozwoju, a sieć wynikowa będzie funkcjonalnie poprawna, jeśli tylko zostanie zakończona wykonaniem Epilogu z operatorem pokrycia. W pewnej mierze dzięki temu mechanizmowi, RKS jako reprezentacja genetyczna odznacza się zamkniętością, a więc tym, że dowolny genotyp koduje poprawną sieć.

Rysunek 3.3 przedstawia prosty przykład rozwoju sieci od stanu początkowego do końcowego, wraz z genotypem sieci. Węzły bez połączeń wejściowych i mające funkcję przejścia $*$, dają na wyjściu wartość 1, tak więc przedstawiona sieć realizuje logiczną funkcję EQUIV (funkcję przeciwną do XOR, przy założeniu, że wartość wyjściowa poddawana jest binaryzacji z progiem 0.5).

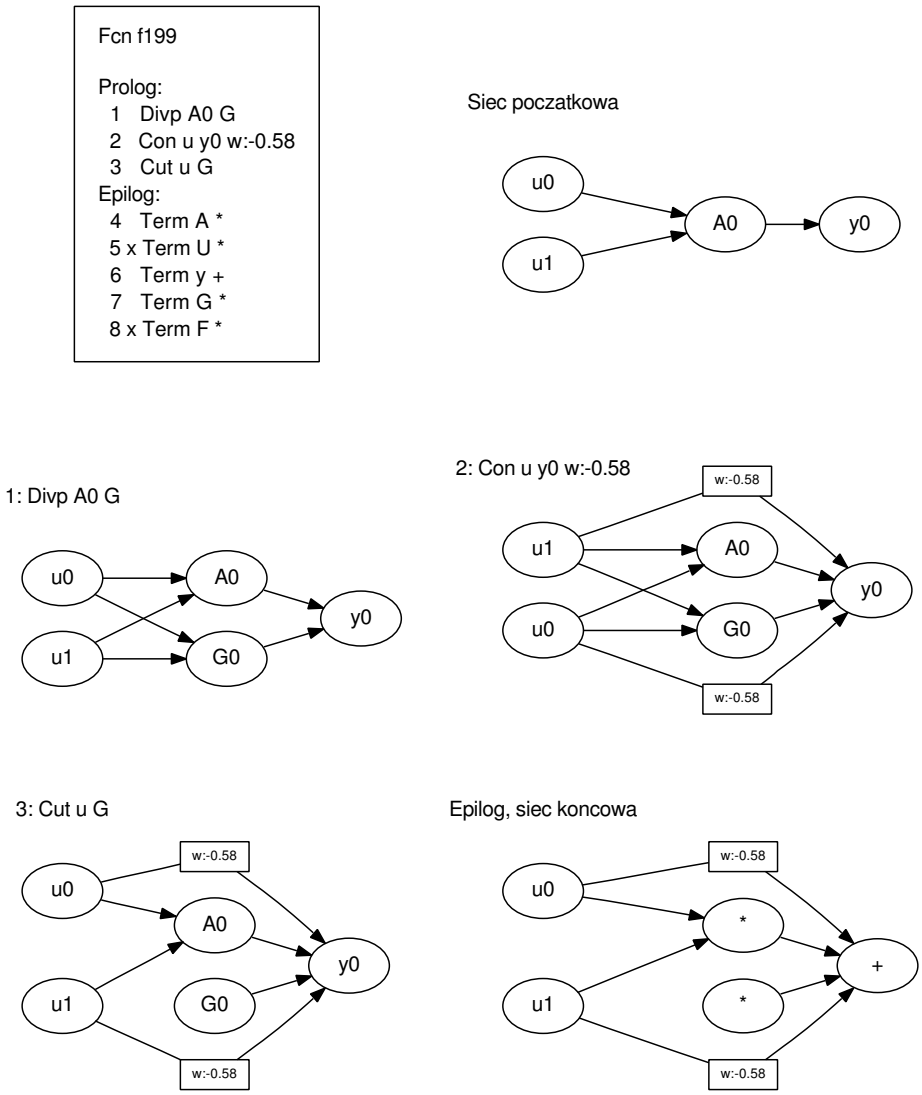
Procedury

Istotną cechą RKS jest zdolność do wielokrotnego wykorzystania kodu w procedurach. Jeśli w zbiorze podprocedur aktualnie wykonywanej procedury znajduje się podprocedura γ' o identyfikatorze F i w Prologu występuje instrukcja FCall F X, wówczas procedura γ' wywoływana jest dla każdego węzła X (tj. pasującego do X), a jej działanie jest następujące:

1: Tworzona jest nowa sieć początkowa G'_0, z jednym węzłem ukrytym A_0, i w której liczba węzłów wejściowych i wyjściowych odpowiada liczbie połączeń przychodzących i wychodzących z węzła X.

2: Procedura γ' jest wykonywana, tj. $G'_\tau = \gamma'(G'_0)$, przy czym zastosowanie operatora pokrycia jest opcjonalne.

3: Sień wynikowa G'_τ jest *składana* z siecią pierwotną G_k rozwijaną przez procedurę γ, z której pochodziło wywołanie.



Rysunek 3.3: Przykład rozwoju sieci. Dany jest genotyp (lewy górny róg) i sieć początkowa. Rozwój przebiega w trzech krokach (instrukcjach) i jest zakończony Epilogiem, który przyporządkowuje węzłom poprawne funkcje przejścia. Instrukcje oznaczone 'x' są nieefektywne.

Złożenie dwóch sieci, nazwijmy je nadrzędną i podrzędną, polega na wstawieniu sieci podrzędnej w miejsce wybranego węzła X w sieci nadrzędnej w ten sposób, że węzły wejściowe sieci podrzędnej są zastąpione przez odpowiadające im węzły w sieci nadrzędnej, natomiast węzły wyjściowe sieci podrzędnej są łączone połączeniami prostymi z odpowiadającymi im węzłami w sieci nadrzędnej, i w końcu węzły ukryte sieci podrzędnej zastępują węzeł X. Przy złożeniu stosowana jest korekta, polegająca na usunięciu bezpośrednich połączeń pomiędzy węzłami wejściowymi i wyjściowymi w sieci podrzędnej — ma to na

celu zachowanie lokalności działania procedury, tak aby jej działanie ograniczało się w efekcie do samego węzła X (bez korekty, podprocedura mogłaby wprowadzać połączenia w sieci nadrzędnej pomiędzy węzłami połączonymi z X).

Istotna kwestia wykorzystania procedur polega na tym, jak odwzorować węzły połączone z węzłem X na węzły wejściowe i wyjściowe początkowej sieci podrzędnej G'_0? Zagadnienie to sprowadza się do sposobu porządkowania połączeń dla danego węzła. Aby procedura była uniwersalna i elastyczna w użyciu, musi istnieć sposób dowolnego, genetycznie modyfikowalnego odwzorowania (porządkowania połączeń) dla dowolnego węzła rozwijanego przez podprocedurę. Jednym z możliwych rozwiązań są dwa dodatkowe atrybuty połączenia, które pozwoliłyby niezależnie uporządkować połączenia przychodzące i wychodzące dla każdego węzła. Jednak to rozwiązanie nie jest stosowane w pracy ze względu na stopień skomplikowania. Zamiast tego przyjmujemy ograniczoną, lecz dużo prostszą regułę, iż połączenia są uporządkowane ze względu na kolejność połączonych węzłów w wektorze $\mathbf{V}$. Alternatywnie można przyjąć taką kolejność połączeń, w jakiej były one dodawane do węzła.

3.5 Operatory genetyczne

RKS wykorzystuje 3 operatory działające na genotypie: mutacja, krzyżowanie oraz pokrycie.

Najważniejszym operatorem jest mutacja. Dokonywana jest ona w dwóch etapach: wybór procedury oraz jej modyfikacja. Wybór procedury ma charakter rekurencyjny. Operator mutacji wywołany dla procedury γ wybiera losowo jedną z jej podprocedur z prawdopodobieństwem:

$$p = \frac{c|\gamma_\gamma|}{|\gamma|} \ , \tag{3.7}$$

gdzie $|\gamma|$ oznacza rekursywnie (tzn. wraz z podprocedurami) liczony w instrukcjach rozmiar procedury, $|\gamma_\gamma|$ rekursywny rozmiar wszystkich podprocedur, natomiast $c \in [0; 1]$ jest współczynnikiem „zagłębienia”, który pozwala regulować względną częstość wywołań mutacji dla podprocedur. Współczynnik zagłębienia związany jest z hipotezą odnośnie modularności ewoluowanych sieci, która mówi, iż redukcja częstości mutacji podprocedur wspomaga tworzenie się użytecznych podprocedur.

Jeśli dana procedura nie zawiera podprocedur ($p = 0$) lub gdy losowanie nie spowoduje zagłębienia operatora, mutowana jest aktualna procedura. A konkretnie, losowo wybierana i wykonywana jest na niej jedna z czterech operacji:

1. Usunięcie losowo wybranej instrukcji z Prologu.

2. Mutowanie losowo wybranej instrukcji z Prologu, czyli zmiana jednego z jej argumentów do postaci niezdeterminowanej.

3. Wstawienie nowej, losowej i niezdeterminowanej instrukcji w losowe miejsce Prologu.

4. Duplikacja losowo wybranej instrukcji, jej mutacja i wstawienie w losowe miejsce Prologu.

Ponadto z niewielkim prawdopodobieństwem, rzędu 0.001, modyfikowany jest identyfikator procedury. W końcu z pewnym prawdopodobieństwem $p \in (0.01; 0.2)$ przeprowadzane jest czyszczenie, czyli usunięcie wszystkich nieefektywnych instrukcji z programu. Czyszczenie nie jest istotną częścią wariacji genetycznej, lecz pozwala zwiększyć czytelność genotypów oraz, jak zaobserwowano, poprawia czasem wyniki przebiegów ewolucyjnych.

Operator krzyżowania działa na poziomie całych procedur, bardzo podobnie jak w programowaniu genetycznym, zważywszy, że program Γ stanowi w istocie drzewo. Krzyżowanie polega na wstawieniu kopii losowo wybranej procedury z programu-dawcy w losowo wybraną procedurę z programu-biorcy. Wraz z procedurą przenoszą się oczywiście wszystkie zawarte w niej podprocedury, a więc krzyżowanie przenosi całe poddrzewo. Jeśli podprocedura o danym identyfikatorze istnieje już w procedurze docelowej, jest ona zastępowana. Warto podkreślić, że program-dawca nie jest w żaden sposób modyfikowany podczas tej operacji. Aby uniknąć niekontrolowanego rozrastania się drzewa, istnieje możliwość narzucenia pewnych ograniczeń co do maksymalnej głębokości drzewa, jak również liczby węzłów potomnych (podprocedur) dla pojedynczej procedury. W przypadku narzucenia takich ograniczeń, losowanie procedur w programach dawcy i biorcy odbywa się tak, aby te ograniczenia zachować.

Trzecim operatorem jest wymieniane już wcześniej pokrycie. W odróżnieniu od mutacji i krzyżowania, pokrycie jest dokonywane podczas rozwoju programu, a ściślej pod koniec Epilogu. Dla każdego węzła w sieci, czyli w wektorze $\mathbf{V}$, oznaczonego nieterminalnym symbolem X, operator pokrycia wstawia do Epilogu instrukcję $\mathsf{Term}\ X\ T$, gdzie T jest symbolem terminalnym, a następnie wykonuje ją. Pokrycie może być aplikowane dla każdej wykonywanej procedury, lub też tylko dla procedury głównej. W obydwu przypadkach gwarantuje ono funkcjonalną poprawność rozwiniętej sieci.

3.6 Właściwości metody

Balakrishnan and Honavar (2001b) określają szereg właściwości odnośnie genetycznych reprezentacji sieci neuronowych, które oczywiście odnoszą się również do sieci obliczeniowych. Są to:

1. Kompletność — reprezentacja jest kompletna, jeśli dowolna sieć może być za jej pomocą wyrażona.

2. Zamkniętość — reprezentacja jest zamknięta, jeśli każdy genotyp koduje poprawną sieć.

3. Skalowalność, rozumiana jako przyrost rozmiaru genotypu względem przyrostu rozmiaru fenotypu (skalowalność przestrzenna), bądź przyrost czasu rozwoju względem przyrostu rozmiaru fenotypu (skalowalność czasowa).

4. Wielokrotność (*multiplicity*). Reprezentacja wykazuje wielokrotność genotypową, jeśli różne genotypy mogą kodować identyczną sieć. Analogicznie wielokrotność fenotypowa zachodzi, kiedy identyczne genotypy w efekcie dekodowania dają różne sieci, ze względu na stochastyczność procesu, lub pod wpływem czynników środowiskowych.

5. Plastyczność ontogeniczna — zachodzi gdy proces rozwoju zależny jest od czynników środowiskowych lub procesu uczenia (zachodzącego podczas rozwoju).

6. Modularność, w sensie zaproponowanym przez Gruau'a (Gruau, 1994a), ma miejsce wówczas, gdy możliwe jest wielokrotne wykorzystanie podsieci w fenotypie, która w genotypie zakodowana jest jednokrotnie.

7. Nadmiarowość (*redundancy*). Genotypowa — jeśli genotyp może zawierać powielone instrukcje lub procedury, bądź fenotypowa — jeśli fenotyp może zawierać powielone węzły lub połączenia, bądź też funkcjonalnie podobne podsieci.

Niektóre z powyższych cech są jasno zdefiniowane (1, 2, 4), inne niezbyt precyzyjnie. Niektóre mają charakter „tak lub nie", inne mają charakter stopniowy, a jeszcze inne wymagają uzupełnienia. Odnośnie niektórych cech nie ma wątpliwości, że są korzystne i pożądane (1, 2, 3, 6), lecz nie można tego powiedzieć o innych, gdyż że ze względu na złożoność omawianych metod trudno jest ocenić znaczenie poszczególnych cech i ich wpływ na efektywność AE w praktyce. Niemniej, warto jest scharakteryzować reprezentację wykorzystywaną w metodzie RKS pod względem przytoczonych cech. Skalowalność i modularność jako cechy szczególnie istotne zostały omówione w osobnym punkcie §2.3.

Reprezentacja genetyczna wykorzystywana w metodzie RKS odznacza się **kompletnością**, pod warunkiem, że nieograniczony jest zarówno rozmiar genotypu jak i zbiór symboli nieterminalnych. Dowolna sieć z przestrzeni fenotypów może być wygenerowana za pomocą instrukcji Divs (lub Divp), Con, Cut oraz Term w następujący sposób. Instrukcja Divs dzieli węzły tak długo, aż otrzymana jest pożądana liczba węzłów, przy czym w ogólności każdy węzeł

ma nadany inny symbol. Dowolne dwa węzły mogą być rozłączone za pomocą instrukcji Cut i połączone ze sobą za pomocą instrukcji Con. Oczywiście niezbędna jest terminacja rozwoju i przyporządkowanie węzłom odpowiednich funkcji przejścia za pomocą instrukcji Term. Tym samym dowolna sieć rekurencyjna z przyjętej przestrzeni rozważań może być wygenerowana.

Reprezentacja odznacza się również **zamkniętością**, pod warunkiem, że w końcowej fazie rozwoju wykonywany jest Epilog wraz z operatorem pokrycia. Zamkniętość kodowania wynika stąd, iż poczynając od poprawnej sieci początkowej, program zawarty w genotypie przekształca ją krok po kroku zgodnie z instrukcjami, a nie istnieje taka instrukcja, która w jakikolwiek sposób mogłaby niepoprawnie przekształcić sieć. Instrukcje co najwyżej mogą być nieefektywne, jeśli np. jej argumenty nie pasują do żadnych węzłów w sieci, lub nie istnieje wywoływana procedura itp. Sieć może być nieprawidłowa pod jednym tylko względem: może być niezakończona, tj. może zawierać węzły o nieokreślonych funkcjach przejścia. Wykonanie Epilogu wraz operatorem pokrycia gwarantuje zakończenie sieci.

Cechą reprezentacji jest również **wielokrotność** genotypowa oraz w pewnym stopniu wielokrotność fenotypowa. Jeden genotyp może dać w efekcie różne fenotypy, jeśli dane są różne sieci początkowe w procesie rozwoju — i na tym właśnie fakcie zasadza się **plastyczność ontogeniczna**. Dekodowanie ma jednak charakter deterministyczny i dla zadanego genotypu i sieci początkowej w efekcie rozwoju otrzymywany jest zawsze ten sam fenotyp.

Cechą reprezentacji jest również **nadmiarowość**. Genotyp może zawierać dowolną ilość powielonych lub nieefektywnych instrukcji czy procedur (choć procedury muszą mieć różne identyfikatory). Połączenia wielokrotne są w fenotypie niedozwolone, lecz powielone podsieci są jak najbardziej możliwe i zostały wielokrotnie zaobserwowane w eksperymentach.

Ważną cechą metody, nie ujętą w powyższej klasyfikacji, jest obiektowy model sieci i wynikająca z niego swoboda w definiowaniu takich elementów sieci jak rodzaje połączeń, funkcje przejścia, uaktualnienia (tj. reguły uczenia) i ich parametry. Co więcej, elementy te mogą również podlegać adaptacji ewolucyjnej, choć możliwość tą w pracy zrealizowano jedynie odnośnie funkcji aktualizacji wag połączeń. W ogólności dowolna funkcja może być zapisana jako wyrażenie algebraiczne w postaci drzewa i może podlegać ewolucji równolegle z siecią, z tym że na „mikro-poziomie”.

Właściwości obliczeniowe

Zgodnie z twierdzeniem o uniwersalnej aproksymacji, jednokierunkowa sieć wielowarstwowa, z jedną warstwą ukrytą i wykorzystująca ciągłe, monotoniczne funkcje przejścia, jest wystarczająca do aproksymacji dowolnej funkcji z n-wymiarowej przestrzeni jednostkowej. Sieć taka zdolna jest do nauczenia się aproksymacji dowolnego odwzorowania, zadanego przez zbiór n-wymiarowych

wektorów wejściowych $\mathbf{x}$ i docelowe wartości wyjściowe $f(\mathbf{x})$ (patrz Haykin, 1998).

Twierdzenie Siegelmann'a i Sontag'a z kolei, mówi o tym, że dowolna maszyna Turing'a może być emulowana przez pewną sieć rekurencyjną o sigmoidalnych funkcjach przejścia (patrz Haykin, 1998).

Z powyższych dwóch twierdzeń oraz kompletności reprezentacji wykorzystywanej w metodzie RKS — która mówi, że metoda ta teoretycznie pozwala wygenerować dowolną sieć rekurencyjną — wynika, że RKS umożliwia wygenerowanie sieci aproksymującej dowolną funkcję, a także sieci równoważnej dowolnej maszynie Turinga, a więc dowolnemu programowi. Oznacza to, że metoda RKS teoretycznie umożliwia rozwiązanie dowolnego problemu obliczalnego w sensie Turing'a, i to w najlepszy możliwy sposób. Naturalnie znacznie mniej można powiedzieć o tym, jak powyższe zdolności metody wyglądają w praktyce.

Właściwości praktyczne

Metoda RKS jest bardzo elastyczna i pozwala na ewoluowanie szerokiego zakresu różnych rodzajów sieci:

- ważonych i nieważonych
- jednokierunkowych i rekurencyjnych
- o połączeniach ustalonych, adaptacyjnych i modulowanych
- o połączeniach, których funkcje przejścia i uczenia mogą podlegać ewolucji
- o dowolnie określonych rodzajach węzłów
- o węzłach, których funkcje przejścia mogą podlegać ewolucji
- o dowolnie określonym interfejsie
- o swobodnie zadawanych ograniczeniach.

Metoda pozwala wykorzystywać dowolnie zdefiniowane rodzaje węzłów i połączeń. Rodzaj węzła lub połączenia jest zakodowany symbolicznie w genotypie i przyporządkowywany węzłowi podczas rozwoju. Funkcja przejścia lub uczenia może być dowolnie zdefiniowana *a priori*, bądź też podlegać może ewolucji, jako np. drzewiaste wyrażenie algebraiczne. Funkcja może być dowolna, lecz oczywiście zbiór argumentów takiej funkcji jest ściśle określony. W przypadku węzłów jest to wektor wartości wejściowych, wartość wyjściowa z poprzedniego kroku oraz parametry lokalne. W przypadku połączeń jest podobnie, przy czym tutaj istnieje tylko jedna wartość wejściowa. Jedynie połączenia modulowane, oprócz sygnału z węzła wejściowego, mogą otrzymywać dodatkowe sygnały z węzła modulującego (lub w ogólności z wielu węzłów), co

pozwala rozszerzyć wachlarz funkcji możliwych do zrealizowania. W pewnym wariancie algorytmu ewaluacji sieci (patrz §4.4), sygnały mogą być również propagowane wstecz, co umożliwia wykorzystanie wstecznej propagacji błędu do uczenia sieci.

Rozdział 4

Badania symulacyjne

W niniejszym rozdziale przedstawione są badania symulacyjne nad metodą RKS. Badania mają charakter eksperymentalny i mają na celu wykazanie określonych właściwości metody RKS. Przede wszystkim jednak, mają na celu wykazanie prawdziwości dwóch tez pracy:

2. Metoda umożliwia syntezę adaptacyjnych sieci odznaczających się skalowalnością i modularnością.

4. Istnieją takie problemy, dla których metoda pozwala generować sieci uczące się efektywniej aniżeli standardowe sieci neuronowe.

Przeprowadzony dowód ma charakter empiryczny i polega na zademonstrowaniu sieci wyewoluowanych za pomocą metody RKS, które potwierdzą powyższe stwierdzenia. Zdolność metody do generowania sieci skalowalnych i modularnych przedstawiona jest w §4.1. W szczególności interesujący jest problem uczenia nadzorowanego w zadaniu klasyfikacji (§4.1.6), zwanym w skrócie problemem „perceptron", dla którego wyewoluowano sieci, których skalowalność wynika w dużym stopniu z ich modularności; na tym problemie zademonstrowano również możliwość wyewoluowania algorytmu uczenia sieci. Zwiększona efektywność sieci ewolucyjnych w stosunku do generycznych SSN została wykazana w następujących problemach testowych: w problemie perceptron (§4.1.6), w problemie spiral (§4.4.2) oraz w generycznym problemie klasyfikacji (§4.6).

Oprócz empirycznych dowodów postawionych tez, w rozdziale zademonstrowano kilka innych właściwości i praktycznych możliwości metody, m.in. możliwość wykorzystania wstecznej propagacji do uczenia sieci (§4.4), wykorzystanie neuromodulacji w sieciach (§4.5), a także korzyści płynące z wykorzystania procedur w równoległym rozwiązywaniu wielu problemów (§4.3). Właściwość skalowalności metody pokazano również na interesującej klasie problemów, polegających na ewolucji topologii spełniającej określone kryteria, jak np. bezskalowość, regularność (w ścisłym sensie teorii grafów), czy modularność według miary Freeman'a (§4.2). Na końcu przedstawiono rozwiązania dla kilku innych problemów (§4.7), demonstrując tym samym zdolność metody do rozwiązywania również problemów niezbyt regularnych.

Istotnym wynikiem pracy, który byłby interesujący, lecz którego realizacja wykroczyła poza zakres pracy, byłoby porównanie efektywności metody RKS z innymi metodami ewolucji sieci. Takie badania jednak są trudne do wykonania ze względu na złożoność metod. Wielość parametrów, szczegółów implementacji, niezgodności w założeniach i wykorzystywanych modelach sieci, i wynikająca stąd mnogość arbitralnych decyzji, które musiałyby zostać podjęte przez eksperymentatora powodują, że takie porównanie wymagałoby wielkich nakładów pracy, a uzyskany obraz porównywanych metod byłby znacznie zniekształcony, jeśli w ogóle niósłby ze sobą wartościową informację. Co więcej, ze względu na wymagania obliczeniowe, porównanie musiałoby się ograniczyć do kilku zaledwie problemów testowych, co jeszcze bardziej obniżyłoby jego wartość poznawczą. W końcu, implementacje metod nie zawsze są dostępne, a są one zbyt złożone, aby mogły być łatwo zaimplementowane. Dlatego w badaniach symulacyjnych rezygnujemy z systematycznego porównywania efektywności metody RKS z pokrewnymi metodami. Jedynie w §4.1.3 oraz §4.1.4, na problemie siatkówki oraz topologii docelowej, przedstawiamy pewne porównanie z metodą HyperNEAT. Wyniki porównania wypadają korzystnie dla metody RKS, lecz na podstawie tak ograniczonych badań nie można uogólniać wniosków o względnej efektywności obydwu metod.

4.1 Skalowalność i modularność

4.1.1 Problem symetrii

Problem symetrii polega na znalezieniu sieci obliczającej logiczną funkcję symetrii n zmiennych. Funkcja zwraca 1, jeśli dla każdej pary wejść (u_i, u_{n-i-1}) ich wartości są równe, tj.:

$$f(\mathbf{u}) = \begin{cases} 1 & \text{jeśli } \forall_{i=0,\ldots,n-1}\ u_i = u_{n-i-1}\ , \\ 0 & \text{w innym przypadku.} \end{cases} \tag{4.1}$$

W eksperymencie wykażemy, że za pomocą RKS możliwe jest znalezienie rozwiązania doskonale skalowalnego. W tym celu wykorzystujemy przyrostowy

wariant algorytmu ewolucyjnego. Początkowo, pojedyncza populacja rozwiązuje problem symetrii dla $n = 2$ wejść. Gdy rozwiązanie zostanie znalezione, tworzona jest nowa populacja, której przydzielone są 2 instancje problemu, dla $n = 2$ i $n = 3$, itd, aż ostatecznie tworzona jest populacja, która rozwiązuje 10 instancji problemu, dla $n = 2, \ldots, 11$. Pomiędzy populacjami umożliwiona jest migracja — w każdym pokoleniu i dla każdej pary populacji następuje wymiana 1% osobników.

Zakłada się, że sieci są jednokierunkowe i wykorzystują dwa rodzaje funkcji aktywacji: n-wejściowe funkcje EQUIV (=) oraz AND (&), zdefiniowane jako:

$$\mathrm{EQ}(\mathbf{x}) = \begin{cases} 0 & \text{jeśli } |\mathbf{x}| = 0 \ , \\ x_0 & \text{jeśli } |\mathbf{x}| = 1 \ , \\ 1 & \text{jeśli } |\mathbf{x}| > 1 \wedge \forall_{i \neq j}\ x_i = x_j \ , \\ 0 & \text{w innym przypadku,} \end{cases} \tag{4.2}$$

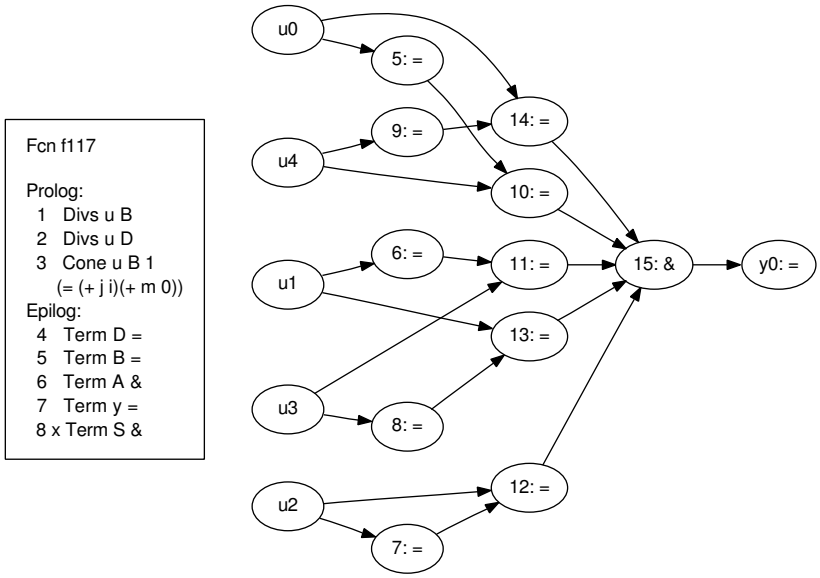
$$\mathrm{AND}(\mathbf{x}) = \begin{cases} 0 & \text{jeśli } \exists_i\ x_i = 0 \ , \\ 1 & \text{w innym przypadku,} \end{cases} \tag{4.3}$$

gdzie $|\mathbf{x}|$ oznacza liczbę elementów w wektorze. Warto dodać, że funkcja AND zwraca 1 również wtedy, gdy $|\mathbf{x}| = 0$.

Rysunek 4.1 przedstawia wybrane wyewoluowane rozwiązanie. Choć było ono selekcjonowane i sprawdzane na podstawie ograniczonego zakresu wymiarowości problemu, jego analiza wskazuje, że jest to rozwiązanie ogólne, dla dowolnego n. Genotyp koduje sieć, która podczas rozwoju automatycznie skaluje się do zadanej wymiarowości problemu. Odbywa się to w trzech krokach: i) warstwa wejściowa dzieli się sekwencyjnie, dając w efekcie warstwę B (węzły 10–14); ii) kolejny podział warstwy wejściowej generuje warstwę D (5–9); iii) tworzone są połączenia proste pomiędzy warstwą wejściową i warstwą B zgodnie z wyrażeniem w instrukcji Cone: $i + j = m + 0$, gdzie i, j oznaczają indeksy węzłów, zaś m przyjmuje wartość najwyższego indeksu w pierwszej warstwie (źródłowej). W końcu, w Epilogu przyporządkowywane są odpowiednie funkcje przejścia. Ukryte węzły sieci są ponumerowane zgodnie z ich pozycją w wektorze $\mathbf{V}$. Przedstawiony genotyp generuje sieci o rozmiarze $|G(n)| = 2n + 2$, podczas gdy minimalne rozwiązanie miałoby rozmiar $\lfloor n/2 \rfloor + 1$. Niemniej, rozmiar genotypu jest stały dla dowolnego n, a więc rozwiązanie jest skalowalne również według wskaźników skalowalności (2.4, 2.5, 2.6).

4.1.2 Problem parzystości

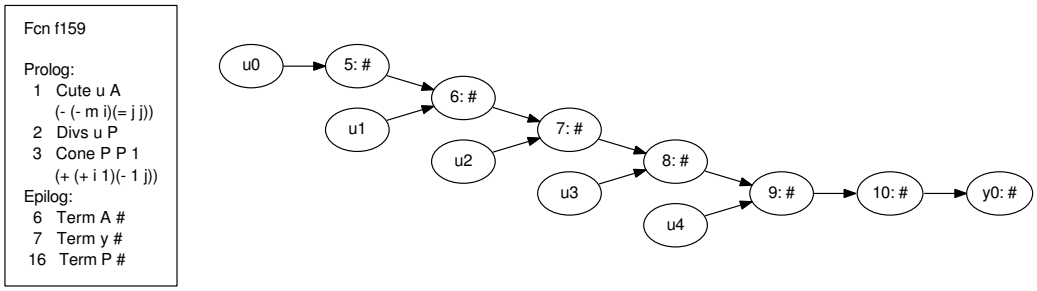
Problem parzystości jest kolejnym dobrze znanym problemem testowym w dziedzinie uczenia maszynowego. Celem jest znalezienie sieci realizującej logiczną funkcję parzystości n wejść, czyli obliczającej dla każdego z 2^n możliwych wektorów wejściowych czy dany wektor $\mathbf{u}$ zawiera parzystą liczbę wartości 1. Podobnie jak w problemie symetrii stosujemy przyrostowy wariant algorytmu



Rysunek 4.1: Doskonale skalowalne rozwiązanie problemu symetrii (genotyp). Przedstawiony fenotyp został rozwinięty dla $n = 5$ wejść.

ewolucyjnego, gdzie $n = 2, \ldots, 11$. Zakłada się, że ewoluowane sieci są jednokierunkowe i wykorzystują tylko jeden rodzaj funkcji aktywacji — NEQ (#), która jest zaprzeczeniem funkcji EQ.

Rysunek 4.2 przedstawia wyewoluowane doskonale skalowalne rozwiązanie postawionego problemu. Analiza genotypu pozwala stwierdzić, że jest to rozwiązanie całkowicie ogólne ze względu na n. Genotyp generuje rozwiązania prawie minimalne, tj. zawierające zawsze tylko 3 węzły więcej.



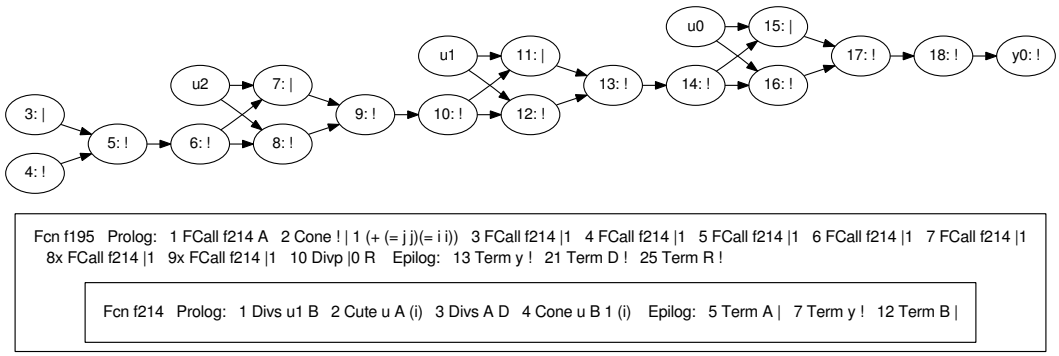
Rysunek 4.2: Doskonale skalowalne rozwiązanie problemu parzystości (genotyp). Przedstawiony fenotyp został rozwinięty dla $n = 5$ wejść.

Modularność

Na przykładzie problemu parzystości zademonstrujemy również zdolność RKS do znajdowania rozwiązań modularnych. W tym celu ponownie wykorzystujemy sekwencyjny wariant AE. Zakładamy, że sieci są jednokierunkowe, jednak inaczej niż poprzednio, do dyspozycji są mniej odpowiednie funkcje aktywacji, a mianowicie: NAND (!), AND (&) oraz OR (|), gdzie NAND jest zaprzeczeniem funkcji AND (4.4), natomiast OR jest zdefiniowany analogicznie jak AND:

$$\mathrm{OR}(\mathbf{x}) = \begin{cases} 1 & \text{jeśli } \exists_i\, x_i = 1\ , \\ 0 & \text{w innym przypadku.} \end{cases} \tag{4.4}$$

Rysunek 4.3 przedstawia modularne rozwiązanie postawionego problemu. Rozwiązanie to jest doskonale skalowalne, choć tylko na ograniczonym zakresie n, a dokładniej dla $n = 2, \ldots, 9$. Dla wyższych wymiarowości problemu genotyp nie generuje poprawnego rozwiązania. Procedura f214 wykorzystywana jest podczas rozwoju do wygenerowania łańcucha podsieci obliczających 2-wejściową funkcję NEQ (czyli XOR). Przedstawiona sieć zawiera 3 takie podsieci, każda z nich połączona z jednym z wejść (węzły 6–9 z u2, 10–13 z u1 oraz 14–17 z u0). Sposób działania procedury jest dość zawiły, m.in. dlatego że generuje ona różne podsieci w zależności od liczby połączeń dochodzących do węzła dla którego jest wywoływana.



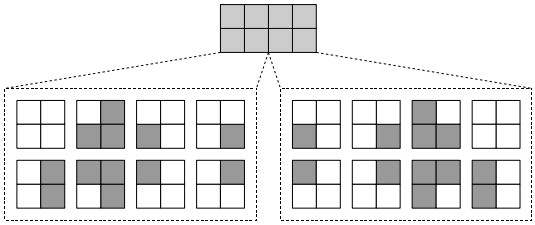
Rysunek 4.3: Modularne, doskonale skalowalne, częściowo ogólne rozwiązanie problemu parzystości (genotyp). Fenotyp został wygenerowany dla przypadku $n = 3$.

Oprócz tego, że rozwiązanie jest modularne, ilustruje również różne aspekty skalowalności. Rozwiązanie jest doskonale skalowalne dla $n = 3, \ldots, 9$ w ścisłym sensie, ponieważ rozmiar genotypu nie wzrasta wraz z rozmiarem fenotypu. W szerszym sensie natomiast, rozwiązanie jest skalowalne, ponieważ dzięki procedurom została uchwycona pewna regularność w problemie. Rozwiązania dla $n > 9$ wymagają jedynie dołożenia kolejnych wywołań procedury f214.

4.1.3 Problem siatkówki

Metoda HyperNEAT jest niewątpliwie zdolna do generowania topologii regularnych, jednak nie jest jasne czy potrafi generować topologie modularne. Badania nad tym przedstawiono w (Clune et al., 2010). Modularność sieci jest tutaj rozumiana jako „lokalizacja funkcji wewnątrz zamkniętej jednostki, która w sieci ma postać grupy węzłów silnie połączonych pomiędzy sobą i słabo połączonych z węzłami spoza tej grupy". Autorzy badania wykorzystali problem siatkówki (*retina problem*), sformułowany oryginalnie w pracy (Kashtan and Alon, 2005), jako odpowiedni test do wykrycia zdolności do ewoluowania rozwiązań modularnych przez kodowanie genetyczne.

Problem siatkówki polega na znalezieniu sieci dla rozpoznawania wzorców wyświetlanych na prostym modelu siatkówki. Składa się ona z dwóch bloków pikseli, lewego i prawego, o wymiarach 2×2 piksela. Spośród 16 możliwych wzorców dla jednego bloku, połowa jest określona jako pozytywne. Pozytywne wzorce są symetryczne dla obydwu części siatkówki. Ilustruje to rysunek 4.4.

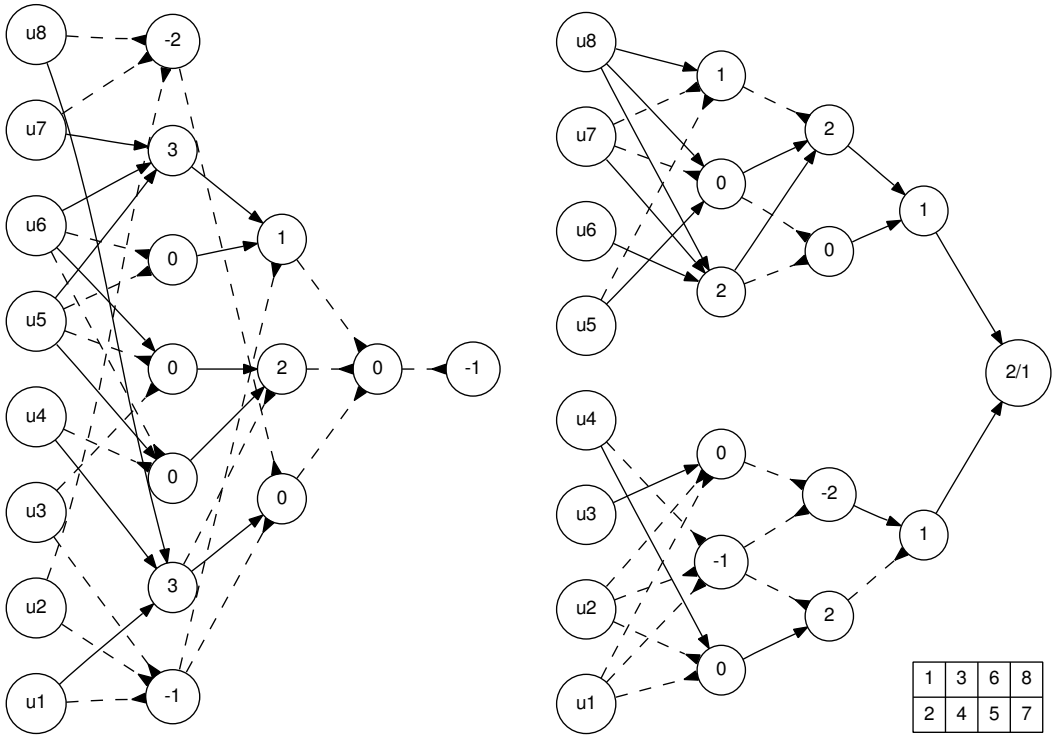


Rysunek 4.4: „Siatkówka" i pozytywne wzorce dla lewej i prawej części.

Dla wszystkich 256 kombinacji wzorców na siatkówce, oceniana sieć musi zdecydować czy którykolwiek blok zawiera wzorzec pozytywny (cel 'OR'), bądź też czy obydwa bloki zawierają wzorce pozytywne (cel 'AND'). Jeśli cel jest ustalony i niezmienny w czasie ewolucji, nie zachodzi nic szczególnego — algorytm znajduje sieci rozwiązujące problem „jakkolwiek", najczęściej o dość zawiłej topologii, w której przetwarzanie wejść z obydwu części siatkówki jest pomieszane. Jeśli jednak cel jest periodycznie zmienny, przykładowo co kilkadziesiąt pokoleń, tego rodzaju pomieszane rozwiązanie, które nie uwzględnia struktury problemu jest nieefektywne — w tym sensie, że zmiana celu pociągałaby za sobą konieczność zmiany w wielu fragmentach sieci. Struktura problemu jest taka, że w pierwszej fazie przetwarzania obrazu siatkówki sieć powinna niezależnie zdecydować czy pozytywne wzorce znajdują się po obu stronach siatkówki — ta faza jest wspólna dla obydwu celów. W drugiej fazie, w zależności od celu, sieć powinna wykonać operację sumy bądź iloczynu logicznego. Tak więc zmiany strukturalne sieci konieczne do realizacji zmiennego wariantu zadania są stosunkowo niewielkie, pod warunkiem, że struktura sieci właściwie odzwierciedla problem. A jest to według podanej definicji struktura modularna, gdzie przetwarzanie lewej i prawej strony siatkówki przebiega

niezależnie od siebie, i co więcej, pomiędzy stronami zachodzi symetria, która może być wykorzystana. Hipoteza stojąca za problemem siatkówki jest taka, że tylko kodowanie zdolne do generowania sieci modularnych może rozwiązać zmienny wariant zadania w sposób efektywny. W rzeczy samej, Kashtan and Alon (2005) ustalili, iż „modularnie zmienny cel" (*modularly varying goal*) zadania, nie tylko wymusił rozwiązania modularne, ale właściwie pozwolił rozwiązać problem efektywniej.

Rysunek 4.5 przedstawia sieci stanowiące rozwiązanie stałego i zmiennego wariantu problemu otrzymane w badaniach Kashtan and Alon (2005). Jak można zauważyć, sieć realizująca stały cel AND przetwarza wejścia łącznie, bez wyraźnej struktury. Dla odmiany sieć mająca do zrealizowania naprzemiennie cele OR i AND wyraźnie rozdziela przetwarzanie wejść z obydwu części siatkówki i łączy je dopiero w węźle wyjściowym. Cała różnica pomiędzy siecią realizującą cel OR i AND zawiera się w progu aktywacji węzła wyjściowego, który wynosi 2 dla celu AND i 1 dla celu OR. Tak więc aby przestawić sieć z jednego celu na drugi, wymagana jest minimalna zaledwie modyfikacja sieci. Struktura sieci jest modularna, choć symetria pomiędzy stronami siatkówki nie została tu wykorzystana ani odzwierciedlona.

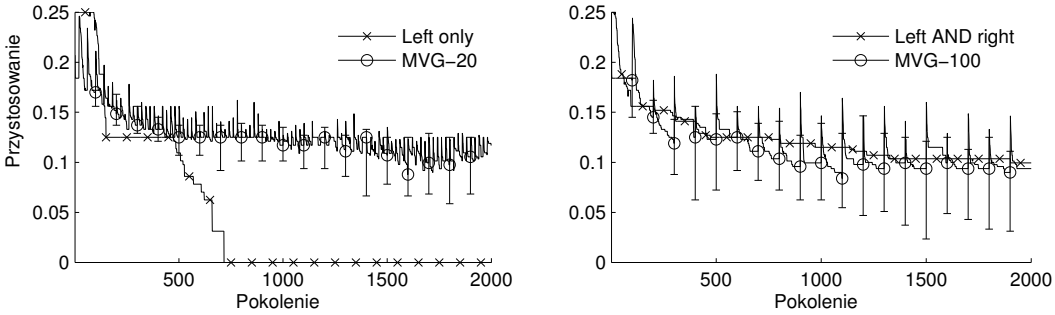


Rysunek 4.5: Sieć stanowiąca rozwiązanie dla wariantu AND (z lewej) oraz rozwiązanie dla wariantu zmiennego (z prawej); układ wejść na siatkówce (z prawej u dołu). Połączenia oznaczone linią ciągłą mają wagi dodatnie, linią przerywaną ujemne; liczby w węzłach ukrytych i wyjściowym oznaczają progi aktywacji.

W sformułowanym problemie, przystosowanie jest proporcjonalne do liczby poprawnie sklasyfikowanych wzorców i mieści się z reguły pomiędzy 0.75 a 1.0. Sieć jest uznawana za rozwiązanie, jeśli klasyfikuje poprawnie 95% wzorców. Badanie przeprowadzone w (Clune et al., 2010) pokazało, że metoda HyperNEAT ma trudności z rozwiązaniem zadania (szczegóły i parametry zadania w artykule). Spośród 20 przebiegów ewolucyjnych, mediana przystosowania dla stałego celu OR wyniosła około 0.85, dla celu AND 0.83; cel zmienny w okresie 20 pokoleń ('MVG-20') dał wynik nie przekraczający 0.78, zaś cel zmienny co 100 pokoleń ('MVG-100') dał medianę nie przekraczającą 0.81. W najłatwiejszym scenariuszu — rozpoznawanie wzorców na jednej części siatkówki — mediana przystosowania wyniosła około 0.88. Wyniki te nie uległy poprawie nawet w przypadku znaczącego wydłużenia przebiegów ewolucyjnych do 30 000 pokoleń. Co więcej, nie wspomniano o żadnych rozwiązaniach dla tego problemu.

Podobny eksperyment przeprowadzono również z kodowaniem RKS, gdzie użyto podobnego rozmiaru populacji jak w przypadku HyperNEAT (500), progowych funkcji transferu i połączeń ważonych. Najważniejsza różnica pomiędzy ustawieniami obydwu systemów dotyczy prawdopodobnie rozmiaru i topologii sieci, które były ustalone w przypadku metody HyperNEAT i ewoluowalne w przypadku RKS. Jest to różnica istotna i zarazem dość fundamentalna pomiędzy tymi metodami. Rysunek 4.4 przestawia medianę przystosowania z 20 przebiegów, trwających 2000 pokoleń, dla różnych wariantów zadania. Ponieważ w stosowanym przez nas algorytmie przystosowanie jest zawsze minimalizowane, liczone jest jako proporcja błędnych, zamiast poprawnych klasyfikacji. Rozpoznawanie wzorców tylko z lewej strony siatkówki nie było trudnym zadaniem dla RKS, jako że doskonałe rozwiązanie zostało znalezione w większości przypadków. Dla celu stałego AND oraz zmiennych MVG-20 i MVG-100 wyniki były podobne i dochodziły do 0.1, co jest znacznie lepszym wynikiem aniżeli w przypadku HyperNEAT, choć trudno jest to potwierdzić statystycznie ze względu na niewielką ilość danych. Inaczej niż w przypadku HyperNEAT, cele zmienne dały nieco lepsze rezultaty, choć margines jest tutaj niewielki.

Tabela 4.1 przedstawia wyniki symulacji, jako liczbę rozwiązań otrzymanych w 20 przebiegach ewolucyjnych. Klasyfikacja pozytywnych wzorców pochodzących z jednej tylko strony siatkówki jest stosunkowo najłatwiejsza ('tylko lewa'). Dla celów OR i AND w wariancie MVG-100 uzyskano rozwiązania standardowe (przystosowanie < 0.05) w 8 przebiegach. Co ważniejsze jednak, w 7 przebiegach otrzymano rozwiązania dla obydwu tych celów w co najmniej dwóch kolejnych okresach zmienności celu (pozycja 'cons.'). Oznacza to, iż struktura rozwiązań była na tyle modularna, że w ciągu jednego okresu możliwa była modyfikacja rozwiązania jednego celu na rozwiązanie dla drugiego. To wskazuje, że RKS odznacza się zdolnością do generowania tego rodzaju modularnych rozwiązań, jakie są niezbędne do rozwiązania problemu siatkówki przy zmiennym celu.



Rysunek 4.6: Mediana przystosowania uzyskana dla kilku wariantów zadania; w przypadku celów zmiennych (MVG) zaznaczono również dolny i górny kwartyl wyników.

Tabela 4.1: Liczba rozwiązań bezbłędnych ($f = 0$), rozwiązań standardowych ($f < 0.05$) oraz rozwiązań „słabych” ($f < 0.1$) otrzymana w 20 przebiegach.

	0	< 0.05	< 0.1		0	< 0.05	< 0.1
Tylko lewa	12	12	12	AND	1	3	10
MVG-20 OR	1	4	12	MVG-100 OR	1	8	13
MVG-20 AND	2	7	11	MVG-100 AND	1	8	15
MVG-20 cons.	1	3	10	MVG-100 cons.	0	7	11

4.1.4 Problem topologii docelowej

Zarówno RKS jak i HyperNEAT mają na celu generowanie skalowalnych rozwiązań dla problemów regularnych. Jednakże regularność jest cechą stopniową. Złożone problemy odznaczające się pewną regularnością, zazwyczaj odznaczają się również pewną dozą nieregularności, np. zawierają wyjątki od reguły. Clune et al. (2008) przeprowadził szereg eksperymentów, mających na celu zbadanie zdolności metody HyperNEAT do reprezentacji wyjątków w regularnej topologii. Problem nazwany *bit mirroring* polegał na wyewoluowaniu pewnego z góry ustalonego wzorca połączeń pomiędzy dwoma 2-wymiarowymi warstwami węzłów. Stopień regularności zadanego wzorca mógł być regulowany poprzez stopień losowości wyboru węzła docelowego. Od topologii najbardziej regularnej, gdzie węzłowi (i, j) w warstwie wejściowej odpowiadał węzeł (i, j) w warstwie wyjściowej; przez regularność tylko w jednym z wymiarów, gdzie dla k węzłów (i, j) losowano węzły docelowe z i-tego wiersza (bądź j-tej kolumny); aż po zupełnie swobodną, gdzie k węzłom losowano węzeł docelowy z całej warstwy. Przystosowanie sieci było proporcjonalne do liczby prawidłowo połączonych węzłów docelowych. Eksperyment pokazał, że choć HyperNEAT skutecznie generował topologie regularne, jego skuteczność gwałtownie spadała, jak tylko zadanie stawało się mniej regularne.

W niniejszym punkcie przedstawiono podobne doświadczenie, w którym

zadanie polega na ewolucji docelowych wzorców połączeń pomiędzy dwoma warstwami 1-wymiarowymi. Zadanie jest sparametryzowane następująco: rozmiar warstwy n_u (tzn. liczba węzłów wejściowych i wyjściowych), offset n_o oraz liczba podmian n_s. Podstawowy wzorzec połączeń łączy węzły wejściowe u_i i węzły wyjściowe y_i spełniające warunek $i = j$. Wprowadzenie offsetu uogólnia ten warunek do:

$$j = (i + n_o) \mod n_u, \;\; i = 0, \ldots, n_u - 1 \;. \tag{4.5}$$

W końcu, połączenia pomiędzy pewnymi węzłami są podmienione. Dokonuje się tego poprzez zamianę węzłów docelowych dla n_s losowo wybranych par węzłów wejściowych (bez powtórzeń). Podobnie jak w sformułowaniu oryginalnym, przystosowanie sieci proporcjonalne jest do liczby węzłów docelowych połączonych zgodnie z zadanym wzorcem.

W eksperymencie wykorzystano implementację HyperNEAT v3.0 napisaną w języku C++ przez J. Gauci.[1] Parametry algorytmu zostały pozostawione przy wartościach domyślnych.[2] Seria symulacji została wykonana dla każdej kombinacji następujących wartości parametrów: $n_u = \{5, 10, 15\}$, $n_o = \{0, \ldots, 29\}$ oraz $n_s = \{0, 1, 2\}$ — w sumie 270 przebiegów dla każdej z metod, tj. RKS i HyperNEAT. Rozmiar populacji wynosił 500. Należy podkreślić, że implementacja problemu była dla obydwu systemów identyczna i co więcej, nie wymagała ona wykonywania żadnych obliczeń na fenotypie. Funkcja przystosowania mogła być obliczona bezpośrednio na macierzy połączeń sieci, która w przypadku HyperNEAT podlegała binaryzacji.

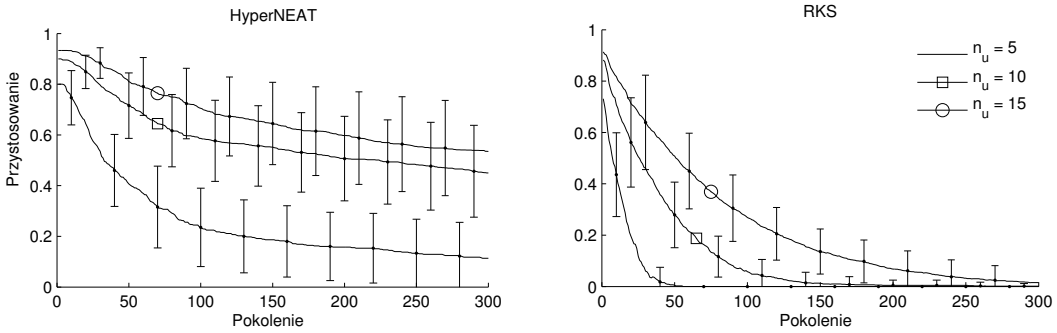
Na rysunku 4.7 wykreślono średnie najlepsze przystosowanie uzyskane dla obydwu metod i trzech rozmiarów warstw. Jak widać trudność problemu wzrasta wraz z rozmiarem warstwy, lecz RKS generuje wyraźnie lepsze rozwiązania aniżeli HyperNEAT.

Nieco trudniej porównać jest skalowalność obydwu metod na podstawie tak relatywnie skromnych danych. Jeśli za koszt obliczeniowy przyjmiemy liczbę pokoleń potrzebną do osiągnięcia poziomu średniego najlepszego przystosowania θ, przy czym ze względu na wyniki bezwzględne $\theta = 0.6$ dla metody HyperNEAT oraz $\theta = 0.2$ dla RKS, wówczas można skorzystać ze wskaźnika S_2 ze wzoru 2.6 i obliczyć koszt obliczeniowy względem rozmiaru fenotypu, czyli równocześnie rozmiarem problemu, czyli rozmiarem warstwy n_u. Dla metody HyperNEAT $S_2 = \{4.6, 8.8, 13.07\}$, zaś dla RKS $S_2 = \{4, 6.4, 8.2\}$. Ilustruje to również rysunek 4.8, na którym koszt obliczeniowy względem rozmiaru problemu wzrasta wolniej w przypadku RKS.

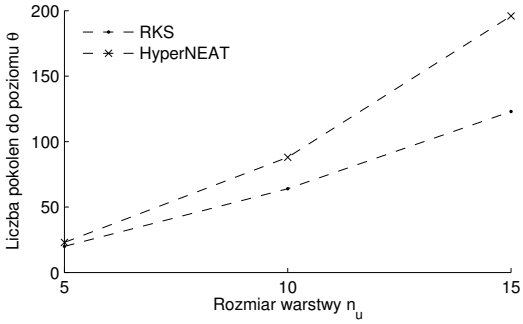
Jeszcze innym sposobem oszacowania skalowalności jest obliczenie o ile większy był koszt przy przejściu z problemu $n_u = 10$ do $n_u = 15$ w stosunku

[1] Oprogramowanie dostępne na stronie http://eplex.cs.ucf.edu/hyperNEATpage/ w tym czasie.

[2] Autorzy metody nie zasugerowali żadnych zmian parametrów podczas konsultacji.



Rysunek 4.7: Średnie najlepsze przystosowanie i odchylenie standardowe dla obydwu metod i różnych rozmiarów warstwy n_u.



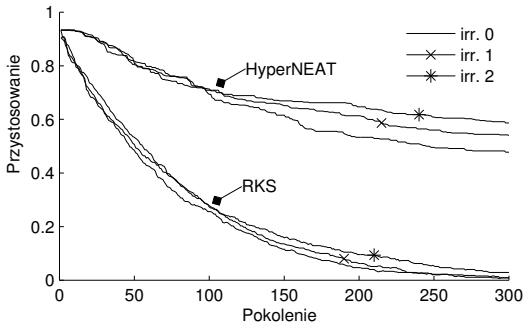
Rysunek 4.8: Skalowalność metody HyperNEAT i RKS.

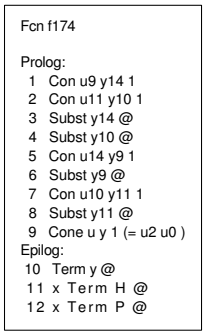
do przejścia z problemu $n_u = 5$ do $n_u = 10$. W przypadku RKS, osiągnięcie poziomu $\theta = 0.2$ zajęło odpowiednio 20, 64 i 123 pokolenia dla wzrastającego n_u, a więc stosunek przyrostu kosztu wynosi (123-64)/(64-20) = 1.34. Analogiczna kalkulacja dla HyperNEAT daje współczynnik (196-88)/(88-23) = 1.66. Obliczenia te wskazują, że metoda RKS wypada lepiej pod względem skalowalności, choć trudno jest te wyniki zweryfikować statystycznie.

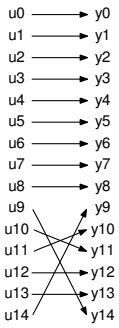
Kolejne istotne pytanie dotyczy skuteczności metod przy wzrastającej nieregularności problemu. Rysunek 4.9 (z lewej) przedstawia wyniki symulacji dla trzech wartości parametru n_s w najbardziej interesującym przypadku $n_u = 15$. Łatwo zauważyć, że skuteczność metody HyperNEAT staje się wyraźnie gorsza wraz ze wzrostem liczby podmian. Inaczej jest w przypadku RKS, na które nieregularność ma niewielki wpływ. Można postawić zarzut, że taki wynik wskazuje, że RKS nie generuje wzorców w regularny sposób, ale łączy węzły pojedynczo, podobnie jak by to było w przypadku kodowania bezpośredniego (patrz Clune et al., 2008). Jednak zarzut ten nie jest do końca prawdziwy.

Rysunek 4.9 (z prawej) przedstawia przykład wyewoluowanego rozwiązania dla instancji $n_u = 15$, $n_o = 0$, $n_s = 2$, gdzie podmienione zostały węzły z indeksami 14 i 9 oraz 11 i 10. Rozwiązanie jest niemal doskonałe, jako że algorytm miał w tym przypadku (zerowy offset stanowi przypadek najłatwiej-

szy) duży zapas "wolnych" pokoleń, aby je udoskonalić, tj. zminimalizować rozmiar genotypu. W pierwszym kroku rozwoju, ustanawiane jest połączenie pomiędzy u9 oraz y14 i następnie pomiędzy u11 i y10. W trzecim i czwartym kroku, węzły y14 oraz y10 mają etykiety zastąpione symbolem terminalnym @ (nie ma on określonego znaczenia). Następnie podobna sekwencja instrukcji jest wykonana dla y9 i y11. W końcu wykonywana jest instrukcja Cone, łącząca węzły wejściowe ui i wyjściowe yi spełniające warunek i = j (symbole u0 i u2 w argumencie instrukcji oznaczają indeksy węzłów odpowiednio warstwy źródłowej i docelowej). Warto zwrócić uwagę, że węzły wyjściowe z zastąpionymi etykietami nie należą do warstwy y i nie są brane pod uwagę podczas tej operacji. W ten sposób zakodowany został regularny wzorzec połączeń z czterema wyjątkami. Przedstawione rozwiązanie nie jest pod względem wykorzystania instrukcji Cone wyjątkiem, jako że rozwiązania wielu innych instancji również wykorzystują tę instrukcję (wraz z Cute). Choć było również wiele takich, które odnajdywały zadany wzorzec łącząc węzły indywidualnie, tj. wyłącznie poprzez instrukcje Con, w sposób charakterystyczny dla kodowania bezpośredniego. Niemniej, wyniki pokazują, że proponowane kodowanie zdolne jest do reprezentowania regularnych wzorców połączeń przy uwzględnieniu wyjątków wyraźnie skuteczniej niż metoda HyperNEAT. Jest tak, ponieważ RKS łączy indywidualne i grupowe mechanizmy tworzenia połączeń sieci.







Rysunek 4.9: Średnie najlepsze przystosowanie dla $n_u = 15$ i wzrastającej nieregularności problemu (z lewej); przykład rozwiązania wyewoluowanego przez RKS dla instancji $n_u = 15$, $n_o = 0$, $n_s = 2$ (z prawej).

Przeprowadzony eksperyment nie był tak wyczerpujący jak można by oczekiwać. W szczególności interesujące byłoby zbadanie zachowania metod dla jeszcze większych rozmiarów warstwy. Wymagania obliczeniowe metody HyperNEAT okazały się jednak dość wysokie, zwłaszcza że koszt oceny przystosowania na podstawie macierzy połączeń był niemal pomijalny — zdecydowaną większość obliczeń pochłaniał rozwój sieci i reprodukcja. 1.2e+11 tyknięć zegara zabrały symulacje z metodą HyperNEAT (36 godzin na komputerze osobistym, bez zbędnych opcji oprogramowania), natomiast 9.6e+9 tyknięć

zegara w przypadku RKS (3 godziny). Choć nie jest to właściwy sposób mierzenia złożoności algorytmicznej, liczby te nie są również bez znaczenia, tym bardziej, że obydwa systemy zaimplementowane zostały w tym samym języku i miały identyczne zadanie do rozwiązania.

4.1.5 Dyskryminacja wizualna

Percepcja wizualna jest odpowiednim obszarem zadań do testowania zdolności kodowania do uchwycenia regularności topologicznych i skalowalności sieci. Układ percepcji wzrokowej, a w szczególności pierwszorzędowa kora wzrokowa u ssaków ma regularną, warstwową strukturę. Przykładowo, detektory orientacji linii są rozmieszczone regularnie w korze wzrokowej, tworząc lokalne, powtarzalne wzorce połączeń. Kora wzrokowa odznacza się wieloma interesującymi cechami funkcjonalnymi. Jedną z nich jest to, iż postrzeganie lub rozpoznawanie bodźców jest w dużym stopniu niezależne od miejsca ich odbioru na siatkówce, co również wskazuje, iż powtarzające się wzory połączeń są w miarę równomiernie rozłożone, począwszy od siatkówki poprzez kolejne warstwy kory wzrokowej.

W badaniach nad metodą HyperNEAT (Gauci and Stanley, 2007), zaproponowano problem „dyskryminacji wzrokowej", mający na celu przetestowanie zdolności metody do generowania sieci dokonujących określonej dyskryminacji dla zadawanych „obrazów" i dla zadanej rozdzielczości. Problem został sformułowany następująco. Dane są dwie, 2-wymiarowe, jednakowe warstwy neuronów — wejściowa i wyjściowa. Geometryczny rozkład neuronów na warstwach jest równomierny i mieści się w kwadracie o wymiarach 2×2. Liczba neuronów, a więc zarazem ich gęstość, zależy od zadanej rozdzielczości. Warstwa wejściowa imituje siatkówkę oka, natomiast wyjściowa odpowiada pierwszej warstwie kory wzrokowej. Na siatkówce wyświetlane są, w pewnej odległości od siebie, 2 obiekty w postaci kwadratów, przy czym jeden z nich ma bok 3 razy dłuższy. Zadanie polega na wygenerowaniu takiego wzorca połączeń pomiędzy warstwami, aby na warstwie wyjściowej największą wartość przyjmował neuron odpowiadający wejściu wyświetlającemu środek większego obiektu — a więc chodzi o wskazanie większego kwadratu. Kwadraty pojawiają się na warstwie wejściowej w losowych miejscach i w pewnym odstępie od siebie.

W eksperymencie podejmujemy podobny problem, z tym że w wersji 1-wymiarowej. Warstwa wejściowa i wyjściowa liczą sobie jednakową liczbę n węzłów, która związana jest z „rozdzielczością" sieci. Rozmiar małego obiektu d_1 jest jednostajnie losowy ze zbioru $\{1, 2, \ldots, [n/10]\}$, gdzie $[n/10]$ oznacza wartość zaokrągloną do najbliższej liczby naturalnej (bez zera). Rozmiar większego obiektu $d_2 = 3d_1 - ((d_1 + 1) \mod 2)$, a więc jest 3-krotnie większy, z tym że w przypadku gdy d_1 jest nieparzyste, d_2 jest pomniejszany o 1 do rozmiaru nieparzystego, aby uniknąć dwuznaczności rozwiązania. Odległość pomiędzy obiektami wynosi minimum $[n/10]$ węzłów warstwy wejściowej. Kryterium

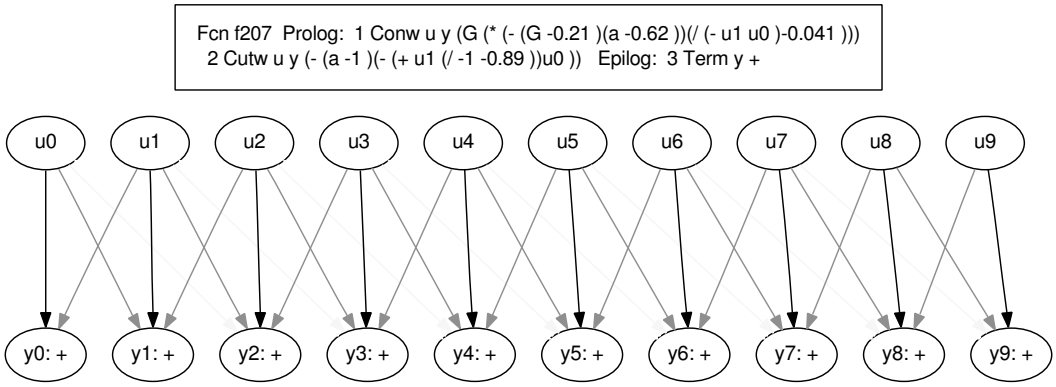
przystosowania składa się z 30 losowych prób, tj. wyświetleń. Na kryterium testowe składa się 600 prób.

Aby wykonać poprawnie zadanie należy znaleźć odpowiedni wzorzec połączeń pomiędzy warstwą wejściową i wyjściową. Do tego wystarczająca jest sieć jednokierunkowa, bez węzłów ukrytych, a więc dla uproszczenia przyjmujemy, że sieć początkowa nie zawiera żadnego węzła ukrytego ani połączeń. Nie są też wykorzystywane instrukcje podziału węzłów. Dwie wykorzystywane tu instrukcje to Conw X Y E oraz Cutw X Y E, będące zmodyfikowanymi wersjami instrukcji Con i Cut. Argumenty X i Y określają węzły warstwy źródłowej i docelowej, zaś E stanowi wyrażenie algebraiczne w formie drzewa o dwóch argumentach wejściowych. Wyrażenie E obliczane dla każdej pary węzłów $x \in X$ i $y \in Y$, gdzie jako zmienne wejściowe przyjmuje się indeksy węzłów w warstwie znormalizowane do przedziału $[0; 1]$. Wyrażenie E wykorzystuje takie operacje jak $+$, $-$, $*$, $/$, abs(), czy — bardzo przydatna w tym przypadku — funkcja Gauss'a $G()$. W przypadku instrukcji Conw, wartość zwrócona przez wyrażenie E określa wagę połączenia, natomiast w przypadku Cutw determinuje, czy połączenie ma być zerwane (dla $E(x, y) > 0$).

Odpowiedni dobór parametrów metody, a wręcz nowych odmian instrukcji, niewątpliwie łamie zasadę „minimalnej wiedzy o problemie", która jest niezbędna do rozwiązania problemu metodą ewolucyjną. Trzeba jednak podkreślić, że niniejszy problem znakomicie wpisuje się w możliwości i ograniczenia metody HyperNEAT, co może wywoływać wrażenie, że problem został specjalnie dobrany dla tej metody. Tak więc choć metoda RKS nie byłaby w rozsądnym czasie rozwiązać postawionego problemu w sposób skalowalny bez instrukcji Conw, ostatecznie rozwiązała go w sposób podobny do metody HyperNEAT. Warto dodać, że konstrukcja i działanie instrukcji Conw nie odbiega zasadniczo od ogólnej koncepcji kodowania i rozwoju metody RKS. Interesujące, że koncepcja metody HyperNEAT jest niemal identyczna z działaniem tej instrukcji, która poprzez wyrażenie algebraiczne generuje pewien wzorzec połączeń pomiędzy dwoma warstwami.

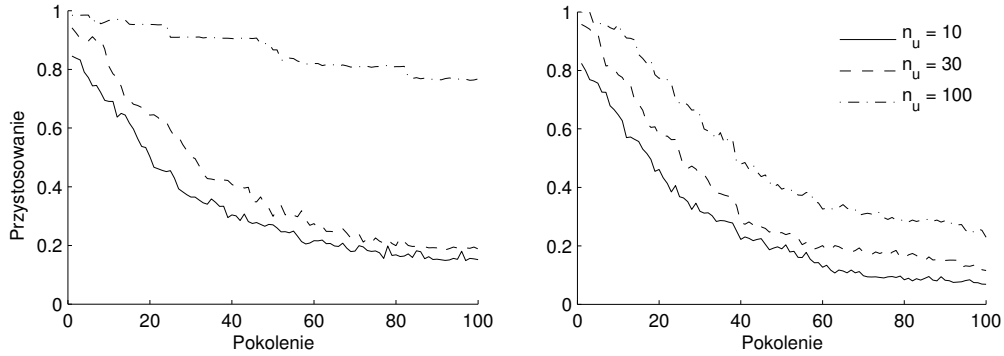
Rysunek 4.10 przedstawia przykład wyewoluowanego rozwiązania dla rozmiaru warstwy $n = 10$. Wagi połączeń są tutaj oznaczone za pomocą odcieni szarości; połączenia pomiędzy odpowiadającymi sobie węzłami mają wagi największe, podczas gdy połączenia z węzłów sąsiednich nieco mniejsze. W sieci występuje też kilka innych połączeń o wagach tak niewielkich, że są niewidoczne.

Rysunek 4.11 przedstawia najlepsze przystosowanie, uśrednione z 20 przebiegów algorytmu wykorzystującego model wyspowy. Populacja składa się tutaj z trzech podpopulacji (demów), z których każda rozwiązuje problem dyskryminacji o innej wymiarowości, $n_u \in \{10, 30, 100\}$. Wykres po lewej przedstawia wyniki wariantu, w którym komunikacja pomiędzy demami nie była umożliwiona. Wykres po prawej dotyczy sytuacji, kiedy pomiędzy demami zachodziła migracja na poziomie 2% populacji na pokolenie. Można zaobserwo-



Rysunek 4.10: Przykład wyewoluowanego rozwiązania zadania dyskryminacji obiektów dla instancji $n = 10$.

wać, że choć wyniki dla mniejszych wymiarowości są podobne, najtrudniejsza instancja problemu (dla $n_u = 100$ wejść) jest rozwiązywana istotnie skuteczniej gdy komunikacja pomiędzy demami jest możliwa (na poziomie istotności $p_{\text{val}} = 0.0004$). Wynik na poziomie 0.23 oznacza, iż przeciętnie najlepszy osobnik z ostatniego pokolenia popełniał błąd w 23% prób. Należy dodać, że wśród 20 przebiegów, dla każdej instancji problemu — zarówno z komunikacją jak i bez — znaleziono rozwiązanie odznaczające się błędem 0.



Rysunek 4.11: Średnie najlepsze przystosowanie z przebiegów algorytmu wykorzystującego model wyspowy z migracją (z prawej) i bez (z lewej).

Tabela 4.2 przedstawia średni rozmiar genotypu ($|\bar{\Gamma}|$) oraz średnią liczbę połączeń ($\bar{m}$) w uzyskanych rozwiązaniach dla rozpatrywanych instancji problemu i obydwu wariantów algorytmu. Choć dane pochodzą z niewielu symulacji i nie są wiążące statystycznie, można zaobserwować — zwłaszcza dla wariantu z migracją — że w przybliżeniu stały rozmiar genotypu koduje sieć z kwadratowo (wobec liczby wejść) wzrastającą liczbą połączeń.

Problem dyskryminacji wizualnej odznacza się regularnością, bez względu na zadaną wymiarowość tego problemu. Jeśli regularność ta zostanie uchwy-

Tabela 4.2: Średni rozmiar genotypu i liczba połączeń w rozwiązaniach.

	n_u, brak migracji			n_u, migracja 2%		
	10	30	100	10	30	100
$\lvert\bar{\Gamma}\rvert$	4.00	2.70	7.00	3.86	3.93	3.16
$\bar{m}$	84	826	6304	71.6	801	8785

cona dla jednej instancji problemu może być przeniesiona na inne instancje. Fakt, iż wykorzystanie modelu wyspowego z komunikacją pozwoliło znacząco zwiększyć efektywność rozwiązywania wielkowymiarowych instancji problemu oznacza, iż kodowanie było zdolne do uchwycenia występującej w problemie regularności. Ponadto, kodowanie okazało się skalowalne ze względu na rozmiar genotypu w stosunku do fenotypu.

4.1.6 Problem uczenia nadzorowanego w klasyfikacji

Ważnym zagadnieniem w badaniach nad ewolucyjnymi sieciami neuronowymi jest kwestia algorytmu uczenia takich sieci. Najczęściej przyjmuje się, że ewolucji podlega architektura sieci, natomiast algorytm uczenia jest zadany z góry, jako na przykład algorytm wstecznej propagacji błędu. Takie rozwiązanie znacznie redukuje przestrzeń poszukiwań, co z jednej strony często przyspiesza znalezienie satysfakcjonującego rozwiązania, lecz z drugiej strony ogranicza możliwości znalezienia potencjalnie lepszych rozwiązań, tj. sieci uczących się w sposób efektywniejszy dla danego problemu aniżeli poprzez algorytm wstecznej propagacji błędu. W niniejszym punkcie zademonstrujemy, iż za pomocą RKS możliwe jest wyewoluowanie sieci wraz z algorytmem uczenia. Uczenie sieci będzie wynikało z jej wewnętrznej struktury i dynamiki, a nie z narzuconego algorytmu czy zadanych reguł uczenia. Ewolucja „autonomicznego” algorytmu uczenia zostanie zademonstrowana na przykładzie liniowo separowalnego problemu klasyfikacji, lub inaczej problemu emulacji pojedynczego neuronu typu perceptron, przy założeniu elementarnych funkcji przejścia $(+, *)$ oraz połączeń prostych.

Postawiony w poniższym eksperymencie problem klasyfikacji stanowi również znakomity sprawdzian właściwości skalowalności i modularności kodowania.

Niech $\mathbf{x} \in \mathbb{R}^n$ będzie punktem (wektorem) wylosowanym z wielowymiarowego rozkładu normalnego o średniej 0 i odchyleniu standardowym 1 ($N(\mathbf{0}, I)$). Niech $X = \{\mathbf{x}_i;\ i = 1, \ldots, 20\}$ oznacza zbiór uczący 20 punktów. Niech $\mathbf{w}$ oznacza n-wymiarowy wektor z podobnego rozkładu $N(\mathbf{0}, I)$, który będzie działał jak hiperpłaszczyzna dzieląca przestrzeń na dwie klasy. Następnie, dla każdego wektora $\mathbf{x}_i$ obliczana jest jego klasa następująco:

$$y_i = (\mathbf{w} \cdot \mathbf{x}_i > 0) \ , \tag{4.6}$$

przypisując 0 do punktów leżących po jednej stronie hiperpłaszczyzny i 1 do tych po stronie drugiej (wektor etykiet oznaczmy jako Y). Następnie losujemy wektor przesunięcia $\mathbf{z}$, ponownie z rozkładu $N(\mathbf{0}, I)$, i przemieszczamy wszystkie punkty ze zbioru X o ten wektor. W rezultacie otrzymujemy zbiór 20 losowo oznaczonych, a jednocześnie liniowo separowalnych punktów, rozmieszczonych „normalnie” wokół pewnego punktu w pobliżu początku układu współrzędnych. Zadaniem sieci jest nauczenie się poprawnej klasyfikacji punktów ze zbioru X.

Ewoluujące sieci oceniane są według następującego algorytmu:

Niech kryterium $J := 0$
Dla 10 prób **wykonuj**
  Odtwórz zadanie, tj. zbiory X i Y oraz wektory $\mathbf{w}i\mathbf{z}$
  Dla 8 epochs **wykonuj**
    Permutuj punkty w X oraz odpowiednio etykiety w $\mathbf{Y}$
    Ustal błąd $e := 0$ i błąd skumulowany $e_{\text{cum}} := 0$
    Dla każdego punktu $\mathbf{x}_i \in X$ **wykonuj**
      Oblicz wyjście sieci $y := \phi([e\ \mathbf{x}_i])$
      Oblicz błąd $e := y_i - (y > 0.5)$ (który wynosi -1, 0, lub 1)
      $e_{\text{cum}} := e_{\text{cum}} + |e|$
    Jeśli $e_{\text{cum}} = 0$ **wtedy**
      Przejdź do następnej próby
  $J := J + e_{\text{cum}}/(10 * 8)$

Powyższy algorytm przewiduje 10 prób. Wartość przystosowania równa jest średniej liczbie błędnych zaklasyfikowań z ostatnich epok ze wszystkich prób. Jeśli w pewnej epoce sieć klasyfikuje wszystkie punkty poprawnie, próba jest przerywana z błędem 0. W każdym pokoleniu najlepszy osobnik w populacji testowany jest nie na 10, lecz na 300 próbach. Warto zwrócić uwagę, że w zadaniu nie przeprowadza się żadnej walidacji klasyfikacji, ponieważ celem zadania jest wykształcenie zdolności sieci do uczenia się, a więc wystarczające dla celów zadania jest zaobserwowanie uczenia na zbiorze trenującym. W celu uniknięcia obciążenia selekcyjnego, rekordowy osobnik ze wszystkich pokoleń testowany jest ponownie i ten wynik brany jest jako wynik ostateczny przebiegu ewolucyjnego.

W eksperymencie stosujemy ewolucję przyrostową. Na początku występuje pojedyncza populacja (o rozmiarze $M = 2048$) rozwiązująca problem dla $n = 1$. Kiedy tylko znalezione jest rozwiązanie dla przypadku n-wymiarowego, tworzona jest nowa populacja rozwiązująca problem o wymiarowości $n+1$. Celem zaoszczędzenia nakładów obliczeniowych, liczność populacji $1, \ldots, n$ jest w każdym takim kroku redukowana o połowę. W każdym pokoleniu, każda populacja wymienia 1% osobników z inną, losowo wybraną populacją.

Ze względu na losowość danych oraz ograniczony czas uczenia, uzyskanie

Tabela 4.3: Porównanie średnich (wiersz μ i σ dla odchylenia standardowego) i najlepszych (wiersz Min) wyników dla standardowego perceptronu i sieci wyewoluowanych. Podano również średnią liczbę pokoleń pomiędzy kolejnymi etapami w ewolucji przyrostowej.

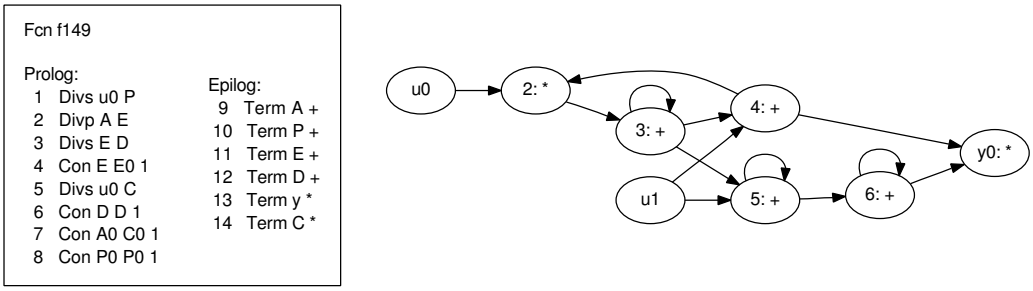
		n					
		1	2	3	4	5	6
	Min	.010	.024	.037	.041	.044	.045
	μ	.013	.033	.045	.052	.053	.053
Perceptron	σ	.002	.005	.006	.005	.005	.005
	Min	.000	.012	.026	.047	.051	(.054)
	μ	.012	.040	.050	.060	.071	(.098)
Sieci ewolucyjne	σ	.007	.010	.012	.013	.016	(.023)
	μ	264	777	126	77	170	106
Liczba pokoleń	σ	346	1627	82	82	321	95

optymalnego wyniku w zadaniu jest dosyć trudne. Nawet neuron typu perceptron uczony za pomocą reguły delty zawodzi (implementacja z *Neural Network Toolbox* dla *Matlab*'a). Dlatego przyjmuje się, że sieć jest zdolna do uczenia, i tym samym rozwiązanie jest satysfakcjonujące, jeśli kryterium $J < 0.1$, tj. jeśli klasyfikuje poprawnie średnio co najmniej 90% punktów. Tabela 4.3 przedstawia wyniki wyewoluowanych sieci w porównaniu ze standardowym perceptronem uczonym za pomocą reguły delty. W przypadku $n = 6$ wyniki są na poziomie rozwiązania satysfakcjonującego, gdyż przebieg był przerywany jak tylko takie rozwiązanie było znajdowane. Średnie wyniki pochodzą z 20 przebiegów ewolucyjnych.

We wszystkich 20 przebiegach ewolucyjnych rozwiązania dla wszystkich przypadków $n = 1, \ldots, 6$ zostały znalezione w przeciągu 8000 pokoleń. W Tabeli 4.3 podano również średnią liczbę pokoleń jaka upływa pomiędzy znalezieniem rozwiązania dla przypadku $n - 1$ i n. Można zaobserwować, że najtrudniejsze jest znalezienie rozwiązania dla $n = 2$, dysponując rozwiązaniem dla $n = 1$. Stosunkowo trudne okazało się również znalezienie rozwiązania dla $n = 1$. Istotne jest, że nakład obliczeniowy niezbędny do znalezienia rozwiązań dla $n > 2$ nie wzrasta wraz z wymiarowością problemu, a więc pod tym względem kodowanie jest skalowalne.

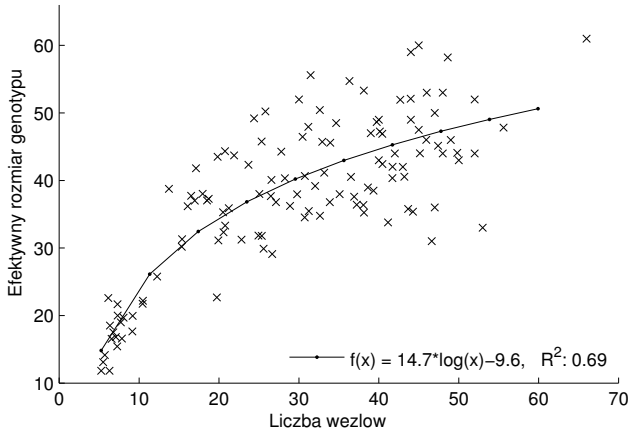
Wyniki uzyskiwane przez sieci ewolucyjne podobne są do tych uzyskiwanych przez standardowy perceptron uczony za pomocą reguły delty. Dla małych n, najlepsze sieci ewolucyjne wypadają nawet lepiej. W osobnym eksperymencie, porównano skuteczność najlepszej wyewoluowanej sieci z perceptronem standardowym dla instancji $n = 1$. Dla 2000 prób wyewoluowana sieć uzyskała optymalny wynik, tj. $J = 0$ przy czym średnia liczba epok uczenia wyniosła 2.3. Perceptron standardowy nie zdołał nauczyć się bezbłędnej klasyfikacji dla 14% zbiorów uczących, a średnia liczba epok uczenia wyniosła 3.3.

Znaczące były również różnice w rzeczywistym czasie trwania eksperymentu: około 400 sekund w przypadku Matlab'owej implementacji perceptronu oraz 0.07 sekundy w przypadku wyewoluowanej sieci rozpisanej jako układ równań różnicowych. Rysunek 4.12 przedstawia jedną z takich wyewoluowanych sieci.



Rysunek 4.12: Wyewoluowana sieć, uzyskująca znacząco lepsze rezultaty od standardowego perceptronu dla instancji $n = 1$.

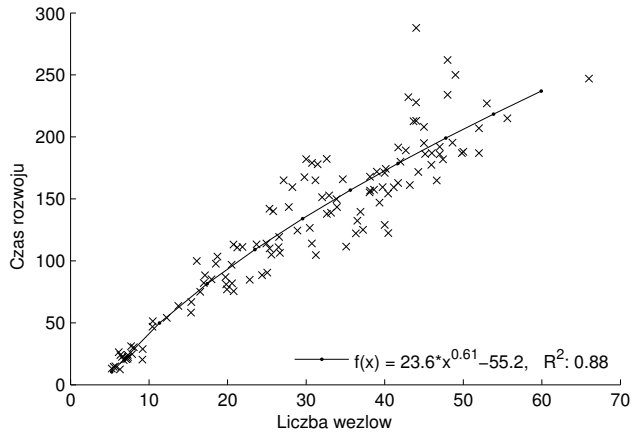
Analiza przebiegów ewolucyjnych na problemie klasyfikacji pozwala również zaobserwować skalowalność RKS. Rysunek 4.13 przedstawia wzrost rozmiaru genotypu (liczba instrukcji) względem rozmiaru fenotypu (liczba węzłów) . Punkty reprezentują tutaj najlepsze osobniki ze wszystkich pokoleń i wszystkich przebiegów ewolucyjnych, stanowiące satysfakcjonujące rozwiązania. Jak widać, do rozkładu można dopasować krzywą logarytmiczną, co oznacza, iż rozmiar genotypu wzrasta znacznie wolniej aniżeli rozmiar fenotypu.



Rysunek 4.13: Logarytmiczny wzrost rozmiaru genotypu względem rozmiaru fenotypu.

Na rysunku 4.14 z kolei przedstawiono wzrost rozmiaru fenotypu względem liczby kroków rozwoju. Tym razem do rozkładu przyłożono krzywą potęgową, przy czym wykładnik wynosi 0.61, w więc krzywa jest w przybliżeniu

„pierwiastkowa”. Obydwa wykresy pozwalają lepiej wyjaśnić źródła skalowalności RKS, a są to procedury oraz „grupowanie” (kiedy pojedyńcza instrukcja operuje na wielu węzłach sieci). Wyjaśnienie jest następujące. Wykorzystanie procedur pociąga za sobą wydłużenie czasu rozwoju, natomiast nie wpływa na rozmiar genotypu. Grupowanie z kolei, nie wpływa ani na czas rozwoju ani na rozmiar genotypu. Ponieważ wzrost rozmiaru genotypu (logarytmiczny) jest wolniejszy aniżeli wzrost czasu rozwoju („pierwiastkowy”), który z kolei jest wolniejszy niż liniowy, można przypuszczać, że skalowalność jest po części efektem obydwu czynników.



Rysunek 4.14: Potęgowy wzrost czasu rozwoju względem rozmiaru fenotypu.

Wykorzystanie procedur okazuje się kluczowe dla rozwiązania problemu klasyfikacji dla $n > 1$. W osobnym eksperymencie wykonano serię symulacji, w których wykorzystanie procedur było zablokowane. Wyniki przebiegów były znacznie gorsze, gdyż nie znaleziono żadnego rozwiązania dla instancji $n > 1$. Można więc wywnioskować, że procedury oraz możliwość ich wymiany pomiędzy osobnikami (operator krzyżowania), znacząco przyczyniły się do efektywnego rozwiązania problemu klasyfikacji.

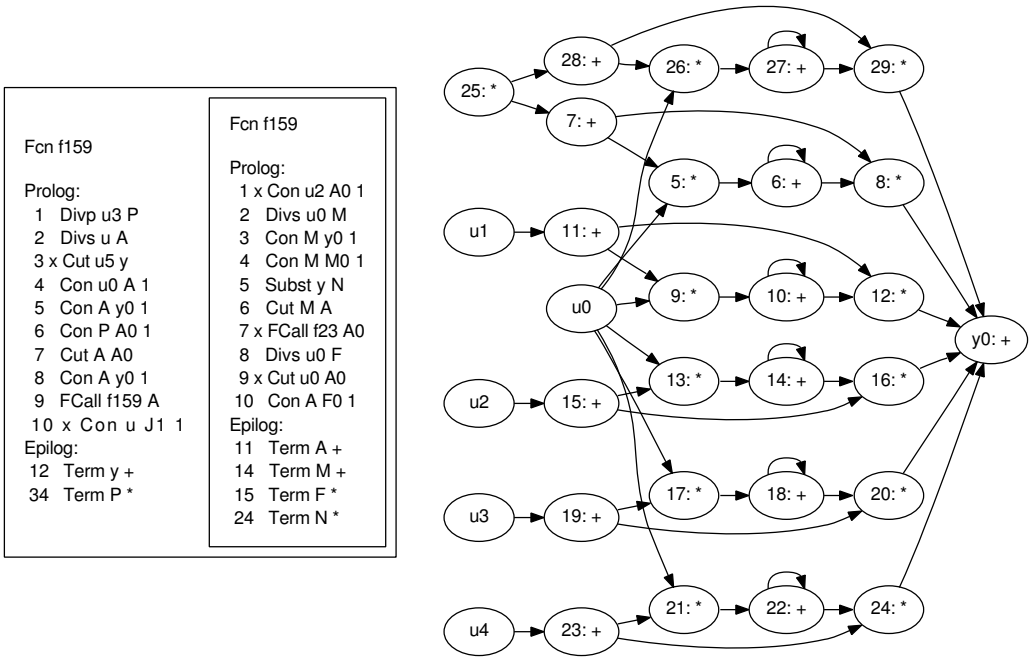
Rysunek 4.15 przedstawia wyewoluowaną sieć i jej genotyp dla instancji problemu $n = 4$ (dodatkowe wejście `u0` reprezentuje błąd klasyfikacji w poprzedniej próbki). Charakterystyczną cechą sieci jest podsieć, powielona dla każdego wejścia i powiązana z wejściem błędu. Dodatkowa, „bezwejściowa” podsieć stanowi odpowiednik biasu w perceptronie (przypomnijmy, że węzeł realizujący funkcję `*` bez wejść daje na wyjściu stałą wartość 1). Co również istotne, powtarzająca się podsieć generowana jest przez jedną i tę samą procedurę `f159`, która jest wykonana sześciokrotnie podczas rozwoju (jedno wywołanie dla 6 węzłów `A`). Kolejne etapy rozwoju sieci wyglądają następująco. Po pierwsze, węzeł `u3` dzieli się równolegle, dając początek podsieci bezwejściowej. Następnie warstwa wejściowa dzieli się szeregowo, dając w efekcie 6 węzłów z symbolem `A` (5 + 1 początkowy). W końcu, po utworzeniu pewnych połączeń,

wywoływana jest procedura dla wszystkich węzłów A, generując powielone podsieci. Co ciekawe, jednakowy identyfikator procedury głównej i podprocedury wskazuje, że wyewoluowały one z tej samej procedury. Warto również zwrócić uwagę na trzecią instrukcję w procedurze głównej, która wskazuje, że pochodzi ona z populacji rozwiązującej instancję problemu dla $n > 4$.

Analiza funkcjonowania sieci pozwala stwierdzić, że każda z podsieci wykonuje operacje odpowiadające uczeniu za pomocą reguły delty w perceptronie. Jeśli przyjmiemy węzeł z połączeniem auto-rekurencyjnym (np. węzeł 14) jako odpowiednik wagi, w, wówczas wyjście v (np. węzeł 16) każdej podsieci związanej z poszczególnym wejściem u wykonuje następujące obliczenia:

$$w(t) := w(t) + \eta u(t-1)u_0(t) \quad \text{i} \quad v(t) := u(t)w(t) \ , \tag{4.7}$$

gdzie $u(t-1)$ stanowi wartość wejściową z poprzedniego kroku (należy zwrócić uwagę na kolejność węzłów w wektorze $\mathbf{V}$ i przez to kolejność ich ewaluacji).



Rysunek 4.15: Modularne rozwiązanie problemu klasyfikacji dla $n = 4$ wejść.

4.2 Ewolucja skalowalnych i modularnych topologii

Zdolność metody do znajdowania skalowalnych i modularnych rozwiązań można badać wykorzystując odpowiednio dobrane problemy testowe, takie jak problem parzystości czy generyczne zadanie klasyfikacji w przestrzeni wielowymiarowej. Jednak możliwe jest również bardziej bezpośrednie podejście, w którym za problem do rozwiązania przyjmuje się znalezienie topologii odznaczającej się zadanymi parametrami. W tym celu wystarczy uwzględnić pożądane parametry topologii w funkcji przystosowania. W tym punkcie przedstawimy eksperymenty, które pokazują, że RKS umożliwia generowanie topologii skalowalnych i modularnych.

We wszystkich eksperymentach tego punktu przyjmujemy, że sieci są nieskierowane. A ściślej algorytm ewolucyjny i parametry są takie jak przy ewolucji sieci jednokierunkowych, lecz przy analizie i ocenie sieci, połączenia są traktowane jako dwukierunkowe (lub niekierunkowe). Tym samym w sieciach nie występują połączenia auto-rekurencyjne.

4.2.1 Sieci bezskalowe

Sieć bezskalowa (*scale-free network*) jest to sieć, której rozkład stopni węzłów jest potęgowy, tj. odsetek węzłów z k połączeniami

$$P(k) \propto k^{-\gamma}, \tag{4.8}$$

gdzie γ oznacza współczynnik rozkładu i najczęściej zawiera się w przedziale $(2; 3)$. Sieci bezskalowe są szczególnie interesujące, ponieważ występują powszechnie w przyrodzie, jak również w technice, czyli w ogólnie rzeczywistości. Sieć relacji towarzyskich, sieć internetowa, interakcje białkowe w organizmach biologicznych czy sieci semantyczne — wszystkie one w większym lub mniejszym stopniu charakteryzują się potęgowym rozkładem stopnia węzłów, a więc należą do klasy sieci bezskalowych (Lewis, 2009).

Problem jaki stawiamy przed RKS jest następujący: znaleźć genotyp sieci, który dla wzrastającej liczby wejść będzie generował sieć bezskalową o liniowo wzrastającej liczbie węzłów. Niech liczba wejść $n_u = |\mathbf{u}| = 0, \ldots, 6$, zaś pożądana liczba węzłów sieci wynosi $n_e = 10 + 2n_u^2$.[3] Dodatkowo wymaga się, aby sieć była spójna, tj. aby istniała ścieżka pomiędzy dwoma dowolnymi węzłami. Biorąc pod uwagę powyższe wymagania, przyjmuje się następującą funkcję przystosowania, liczoną dla fenotypu sieci rozwiniętej dla zadanej liczby wejść n_u:

$$f(G, n_u) = \sqrt{1 - e^{-d_B(P,p)}} + |1 - n/n_e(n_u)| + (n_c - 1) \ , \tag{4.9}$$

[3]Przypomnijmy, że różne wartości liczby wejść stanowią różne warunki początkowe w procesie rozwoju.

gdzie d_B oznacza odległość pomiędzy rozkładem stopni węzłów w sieci a potęgowym rozkładem odniesienia według miary Bhattacharyya, n oznacza liczbę węzłów, zaś n_c liczbę komponentów sieci (spójnych składowych). Odległość Bhattacharyya pomiędzy dyskretnymi rozkładami p i q oblicza się następująco:

$$d_B(p,q) = -\ln \sum_{x \in X} \sqrt{p(x)q(x)} \ . \tag{4.10}$$

Ostateczna wartość przystosowania jest średnią przystosowania dla wszystkich warunków początkowych, tj.:

$$F(G) = \frac{1}{7} \sum_{n_u=0}^{6} f(G, n_u) \ . \tag{4.11}$$

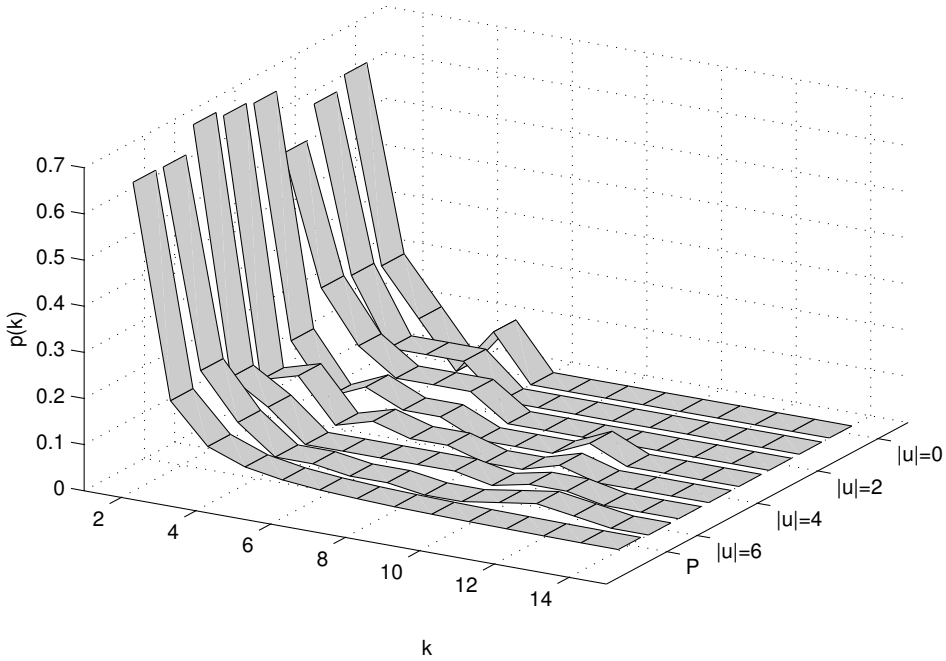
Postawiony problem okazał się niezbyt trudny dla RKS. Przyjęto, iż rozwiązanie jest satysfakcjonujące, jeśli jego przystosowanie $F(G) < 0.27$. Koszt obliczeniowy uzyskania satysfakcjonujących rozwiązań to około 30 000 osobników. W tabeli 4.4 przedstawiono parametry takiego rozwiązania, spełniającego postawione warunki w satysfakcjonującym stopniu. Jak widać, rozwinięta sieć ma dokładnie pożądaną liczbę węzłów, za wyjątkiem przypadków $n_u = 1$ i $n_u = 4$, gdzie liczba węzłów nieznacznie odbiega od zadanej. Niewielkie są również odległości rozkładów stopni węzłów od zadanego rozkładu potęgowego. Rysunek 4.16 przedstawia zadany rozkład potęgowy oraz rozkłady uzyskane po rozwinięciu sieci z odpowiednich sieci początkowych. Przy ocenie podobieństwa uzyskanych rozkładów do rozkładu wzorcowego należy pamiętać, że ze względu na dyskretny charakter rozkładu, możliwość odwzorowania rozkładu zadanego jest ograniczona przy małej liczbie węzłów w sieci. Przejawia się to również w tabeli 4.4 poprzez zmniejszającą się odległość d_B wraz z ilością węzłów sieci. Na rysunku 4.17 przedstawiono genotyp sieci oraz fenotyp sieci rozwiniętej dla liczby wejść $n_u = 0, \ldots, 6$.

Tabela 4.4: Parametry wybranej sieci rozwiniętej dla liczby wejść n_u.

	n_u						
	0	1	2	3	4	5	6
n_e	10	12	18	28	42	60	82
n	10	14	18	28	43	60	82
d_B	.09	.07	.06	.06	.05	.04	.04

4.2.2 Skalowalne sieci modularne

Zgodnie z najbardziej ogólną definicją, sieć jest modularna, jeśli zawiera podsieci gęsto połączone wewnętrznie i rzadko połączone pomiędzy sobą. Jest

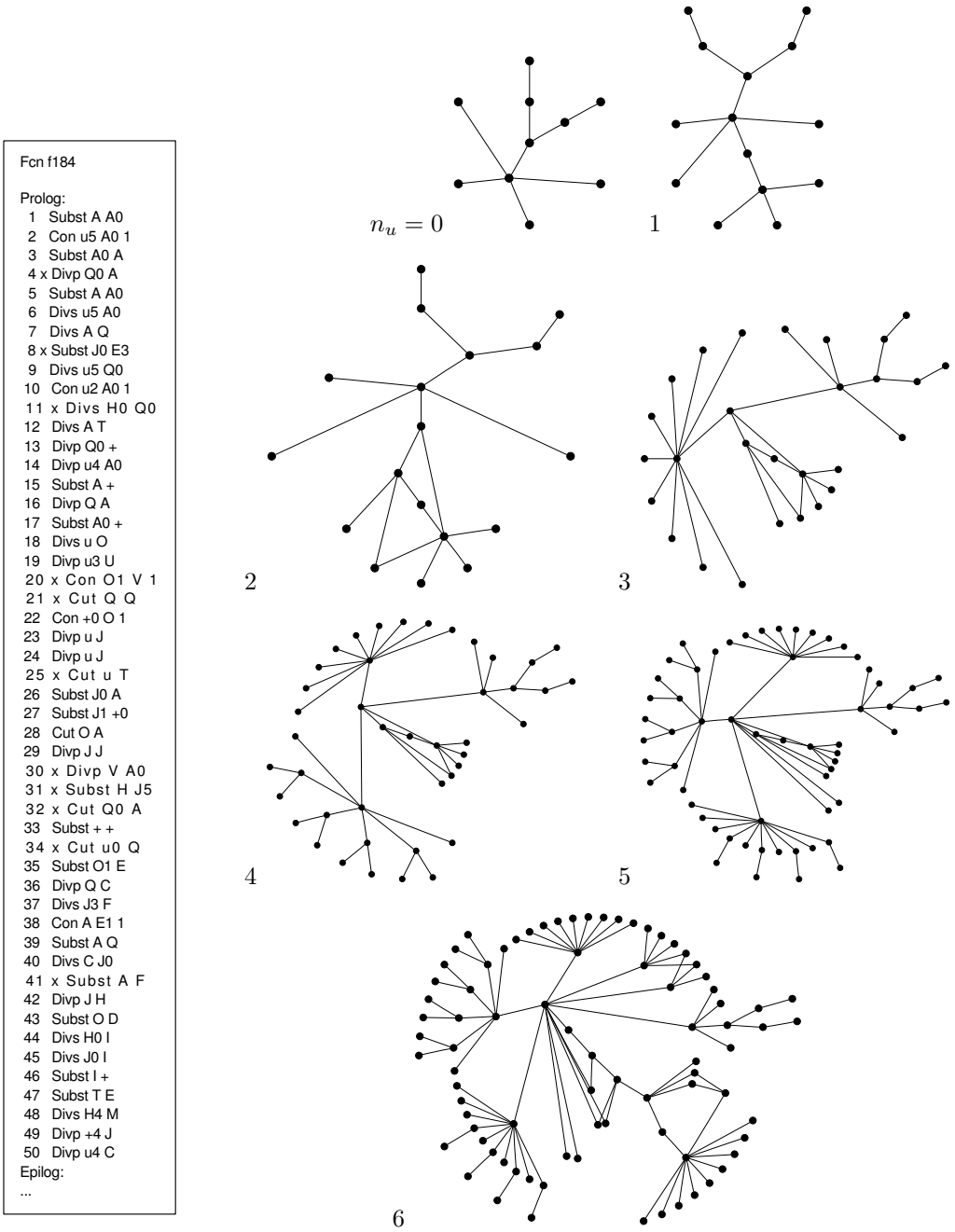


Rysunek 4.16: Potęgowy rozkład stopni węzłów dla $\gamma = 2$ (P) oraz rozkłady sieci rozwiniętych z wyewoluowanego genotypu dla liczby wejść $n_u = 0, \ldots, 6$.

to ujęcie z perspektywy samej topologii, bez uwzględnienia jakiejkolwiek funkcji realizowanej przez sieć czy przez moduły. W ogólnej teorii sieci rozpatruje się wiele miar modularności sieci, jak również algorytmów podziału sieci na moduły. Ze względu na prostotę i dobre efekty, przyjęto tutaj miarę modularności zaproponowaną w (Newman, 2004; Newman and Girvan, 2004). Niech e_{ij} oznacza odsetek połączeń sieci pomiędzy i-tą a j-tą grupą węzłów oraz niech $a_i = \sum_j e_{ij}$. Wówczas

$$Q = \sum_i (e_{ii} - a_i)^2 \ , \tag{4.12}$$

jest częścią wszystkich połączeń wewnątrz modułów po odjęciu oczekiwanej wartości odsetka wszystkich połączeń wewnątrz modułów gdyby połączenia w sieci były rozłożone całkowicie przypadkowo. Innymi słowy, wartość Q wskazuje jak bardzo dana sieć różni się od sieci losowej pod względem struktury czy też koncentracji połączeń. Dla sieci całkowicie losowej Q wynosi około 0, wartości powyżej 0.3 wskazują na istnienie struktury modularnej. Obliczenie wskaźnika modularności wymaga — nieco paradoksalnie —podziału sieci na moduły, inaczej nie sposób określić wartości e_{ij}. W ogólnym przypadku, należałoby obliczyć Q dla wszystkich możliwych partycji i za wskaźnik modularności przyjąć maksymalną wartość Q, co jest jednak obliczeniowo zbyt złożone aby mogło być zrealizowane. Newman (2004) proponuje algorytm zachłanny, który nie gwarantuje znalezienia maksymalnego Q, lecz na ogół działa zadowalająco.



Rysunek 4.17: Genotyp wybranej sieci (z lewej) oraz sieć rozwinięta dla liczby wejść $n_u = 0, \ldots, 6$ (z prawej).

Tabela 4.5: Parametry wybranej sieci rozwiniętej dla zadanej liczby wejść n_u.

	n_u				
	0	1	2	3	4
n_e	10	20	30	40	50
n	17	24	31	39	46
Q	.52	.62	.68	.72	.75

Poczynając od podziału, w którym każdy węzeł stanowi osobny moduł, algorytm iteracyjnie łączy ze sobą te moduły, które dają maksymalny przyrost wartości Q. Złożoność obliczeniowa tego algorytmu wynosi $O((m+n)n)$, gdzie n oznacza liczbę węzłów, zaś m liczbę połączeń w sieci.

Problem jaki stawiamy przed RKS, podobny jest do problemu postawionego w §4.2.1, przy czym zamiast zbieżności rozkładu stopni węzłów do rozkładu potęgowego (4.9) wymaga się tutaj maksymalizacji modularności:

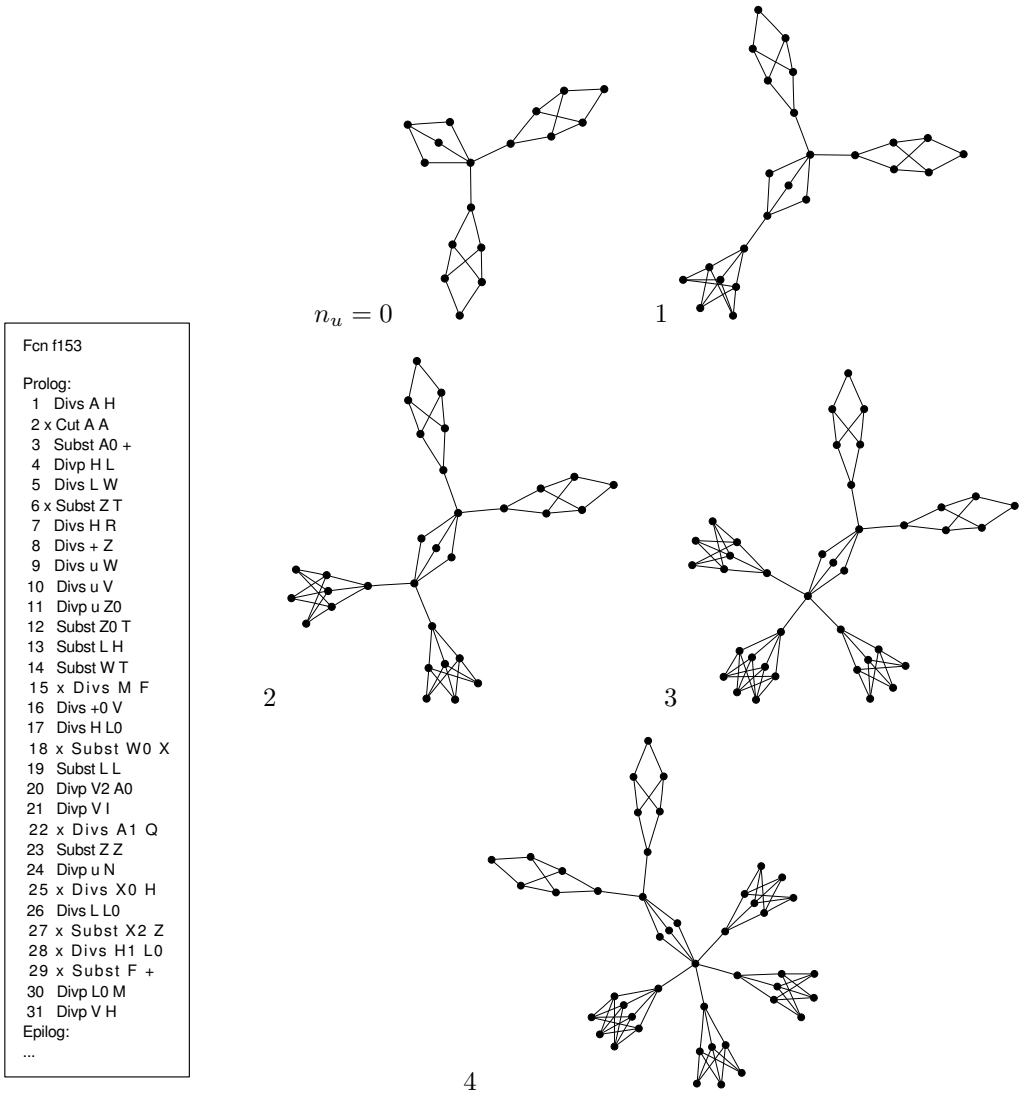
$$f(G, n_u) = (1 - Q) + 0.2|1 - n/n_e(n_u)| + (n_c - 1) \ . \tag{4.13}$$

Podobnie jak poprzednio wymaga się również, aby sieć była spójna oraz aby liczba węzłów sieci wzrastała wraz z liczbą wejść, tym razem jako funkcja liniowa $n_e(n_u) = 10 + 10n_u$, dla $n_u = 0, \ldots, 4$. Ostateczna wartość przystosowania jest średnią z $f(G, n_u)$ dla wszystkich przypadków n_u.

Tabela 4.5 przedstawia wyniki wybranej wyewoluowanej sieci, której przystosowanie wynosi 0.382. Jak widać, uzyskane rozmiary sieci znacznie bardziej odbiegają od zadanych aniżeli w eksperymencie z §4.2.1. Częściowo jest tak dlatego, że tym razem na rozmiar sieci położono mniejszy nacisk w funkcji przystosowania — współczynnik 0.2. Wartość Q wyraźnie wskazuje na modularny charakter sieci, który jest tym wyższy im więcej jest węzłów w sieci. Warto odnotować, iż najwyższa wartość jaka została zaobserwowana przez autorów miary dla jakiejkolwiek sieci wynosi 0.807 (Newman, 2004). Na rysunku 4.18 przedstawiono genotyp i fenotyp wybranej sieci, rozwinięty dla wszystkich przypadków n_u. Można zaobserwować, iż sieć odznacza się nie tylko modularną strukturą, ale również wysoce regularny jest sposób skalowania sieci wraz ze wzrostem liczby wejść. Dodatkowe wejścia dają w efekcie nowe moduły przy zachowaniu dotychczasowych, a całość struktury odznacza się symetrią.

4.2.3 Regularne sieci modularne

W tym punkcie zademonstrujemy elastyczność ewolucyjnej metody w formułowaniu cech i warunków jakie musi spełniać sieć stanowiąca rozwiązanie. Oczywiście trzeba pamiętać, że algorytm ewolucyjny w praktyce nie gwarantuje znalezienia rozwiązania, nawet jeśli ono teoretycznie istnieje. Zbieżność



Rysunek 4.18: Genotyp wybranej sieci (z lewej) oraz sieć rozwinięta dla liczby wejść $n_u = 0, \ldots, 4$ (z prawej).

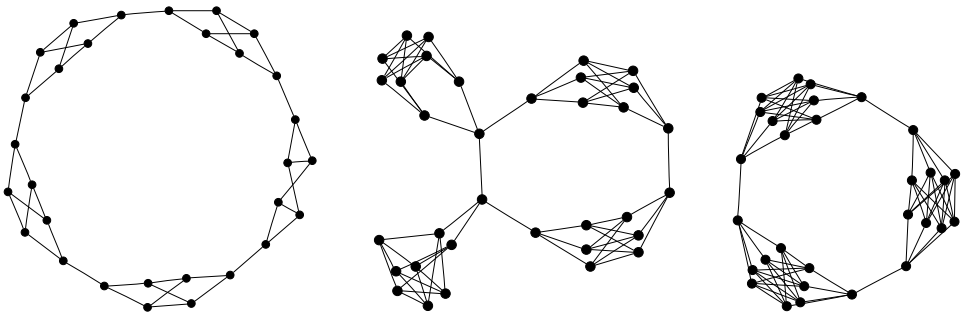
algorytmów ewolucyjnych jest zagwarantowana tylko w przypadku spełnienia określonych warunków przez algorytm i sam problem, które w przypadku metody RKS, ale również większości AE nie są spełnione.

Przedmiotem eksperymenty są sieci regularne w sensie teorii grafów, czyli takie, w których każdy węzeł ma jednakową liczbę połączeń. Należy znaleźć sieć składającą się z około 30 węzłów, w możliwie największym stopniu modularną i jednocześnie k-regularną. Funkcja przystosowania może mieć wówczas postać:

$$f(G) = (1 - Q) + 0.5|1 - n/30| + |d_\mu - k| + |d_\sigma| + (n_c - 1) \ , \qquad (4.14)$$

gdzie d_μ i d_σ oznaczają średnią i odchylenie standardowe stopni węzłów w sieci.

Z pewnością nie jest to jedyna możliwa postać funkcji przystosowania, która prowadzi do celu. Na rysunku 4.19 przedstawiono fenotypy trzech otrzymanych rozwiązań, dla trzech przypadków $k = 3, 4, 5$. Składają się one odpowiednio z 30, 34 i 30 węzłów, ich wskaźniki modularności Q wynoszą 0.64, 0.64, 0.60 i wszystkie spełniają warunek regularności. Koszt obliczeniowy uzyskania powyższych rozwiązań (tj. każdego z osobna) wynosi w przybliżeniu 300 000 osobników.



Rysunek 4.19: Przykłady wyewoluowanych modularnych sieci 3-, 4- i 5-regularnych (w kolejności).

4.3 Ewolucja równoległa a procedury

Genotyp sieci w kodowaniu RKS ma strukturę drzewa, w którym węzły są procedurami. Jedną z motywacji dla wprowadzenia takiej struktury jest hipoteza, iż możliwość wykorzystywania podprocedur i ich wymiana pomiędzy osobnikami w trakcie ewolucji zwiększy efektywność równoległego rozwiązywania wielu podobnych problemów. W tym punkcie przedstawiony jest eksperyment, którego wyniki wspierają tę hipotezę.

Niech problem polega na znalezieniu sieci jednokierunkowych, realizujących następujące 4-argumentowe funkcje logiczne:

- $f_1 = (u_1 \oplus u_2) \wedge (u_3 \oplus u_4)$
- $f_2 = (u_1 \oplus u_2) \uparrow (u_3 \oplus u_4)$
- $f_3 = (u_1 \oplus u_2) \vee (u_3 \oplus u_4)$

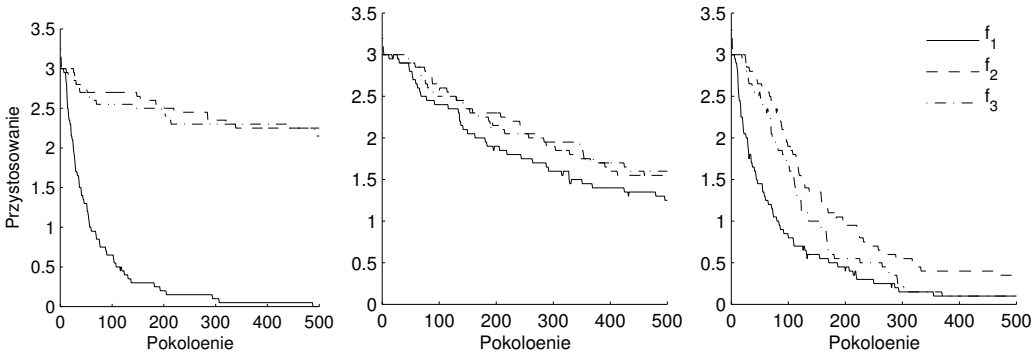
gdzie symbol ! oznacza funkcję NAND. Ograniczenie jest takie, iż jedyna dostępna funkcja aktywacji to wieloargumentowa funkcja NAND:

$$!(\mathbf{x}) = \begin{cases} 0 & \text{, jeśli } \forall_i\ x_i = 1\text{, lub brak wejść} \\ 1 & \text{, w innym przypadku.} \end{cases} \tag{4.15}$$

Jak widać, powyższe funkcje są w dużym stopniu podobne pod względem struktury, ponieważ różnią się tylko jednym z trzech operatorów. Warto przypomnieć, iż aby zrealizować funkcję $\oplus$ niezbędne są minimum 4 operacje !, do realizacji funkcji $\vee$ potrzebne są 3 operacje !, zaś do $\wedge$ 2.

W eksperymencie zastosowano model wyspowy z podziałem populacji na 3 demy, z których każdy rozwiązuje inny podproblem, tj. funkcję. Przygotowano 3 warianty algorytmu. W wariancie 1 osobniki mogą korzystać z podprocedur, lecz brak jest komunikacji pomiędzy demami. W wariancie 2 osobniki nie mogą korzystać z podprocedur, lecz występuje komunikacja pomiędzy demami (migracja 1% liczebności demu na pokolenie). Natomiast w wariancie 3 możliwe jest zarówno korzystanie z podprocedur jak i komunikacja pomiędzy demami. Wszystkie inne parametry algorytmu są identyczne.

Dla każdego z wariantów wykonano 20 przebiegów ewolucyjnych. Rysunek 4.20 przedstawia wyniki symulacji w postaci średniego najlepszego przystosowania w trakcie ewolucji. Jak widać, najlepsze wyniki uzyskano dla wariantu 3, choć w przypadku funkcji f_1 najlepszy okazał się wariant 1. Potwierdza to Tabela 4.6, w której zawarto średnią i odchylenie standardowe przystosowania najlepszego osobnika przebiegu z 20 symulacji. Wariant 3 uzyskał istotnie lepsze wyniki (na poziomie istotności $\alpha = 0.01$, test permutacji) od pozostałych dwóch wariantów, za wyjątkiem funkcji f_1, dla której wariant 1 uzyskał wyniki nieznacznie lepsze.



Rysunek 4.20: Przebieg przystosowania dla trzech wariantów algorytmu: tylko podprocedury (lewy), tylko migracja (środkowy), podprocedury i migracja (prawy).

4.4 Wsteczna propagacja błędu

Algorytm wstecznej propagacji błędu jest jednym z najczęściej stosowanych algorytmów uczenia jednokierunkowych, wielowarstwowych sieci neuronowych (patrz Haykin, 1998). W tym punkcie przedstawiona zostanie możliwość wykorzystania wstecznej propagacji błędu (WPB) w sieciach ewoluowanych za pomocą RKS.

Tabela 4.6: Przystosowanie w ostatnim pokoleniu dla trzech wariantów algorytmu. Wyniki oznaczone ‡ są istotnie gorsze ($p_{\text{val}} < 0.01$) od wyników dla wariantu 3.

	Podproblem		
Wariant	f_1	f_2	f_3
1 (podproc.)	0.00 ± 0.00	2.25 ± 1.07‡	2.15 ± 0.88‡
2 (migracja)	1.25 ± 0.70‡	1.55 ± 0.86‡	1.60 ± 0.86‡
3 (podproc. + mig.)	0.10 ± 0.30	0.30 ± 0.64	0.10 ± 0.30

Wykorzystanie WPB wymaga nieco innego wariantu algorytmu obliczeniowego sieci. Algorytm 3 przedstawia zmodyfikowaną wersję algorytmu 2. Modyfikacja, a raczej rozszerzenie, polega na dodaniu procedury przypisania błędu węzłom wyjściowym, oraz wstecznej propagacji błędu z jednoczesnym uaktualnieniem wag (kroki 2–7). Pozostała część algorytmu pozostaje bez zmian. Wykorzystując algorytm z WPB zakładamy, że ewoluowane sieci są jednokierunkowe.

1: Inicjalizuj węzły i połączenia
2: **Dla** każdego wektora wejściowego **u** i wektora błędu **e** **wykonuj**
3: Przypisz błąd: $\forall\ v_i \in \mathbf{V}_y :\ \delta_i := e_i$
4: **Dla** każdego $v_i \in [\mathbf{V}_u\ \mathbf{V}_h]$ (w odwróconej kolejności) **wykonuj**
5: Oblicz błąd: $\delta_i := \sum_j e_{ij}(\delta_j)$
6: **Dla** każdego połączenia klasy B wychodzącego z v_i **wykonuj**
7: Uaktualnij wagę: $w := w + \eta y_i \delta_j$, bądź ogólniej: $B := h(B, y_i, \delta_j)$
8: Przypisz wejścia $\mathbf{V}_u := \mathbf{u}$
9: **Dla** każdego $v_j \in [\mathbf{V}_h\ \mathbf{V}_y]$ **wykonuj**
10: Oblicz wartość wyjściową: $y_j := f_j(\{e_{ij}(y_i)\}_{i=1,\ldots,k})$
11: **Dla** każdego plastycznego połączenia e_{ij} **wykonuj**
12: Aktualizuj połączenie $e_{ij} : B = g(B, y_i, y_j)$
13: Zwróć $\mathbf{V}_y$,

Algorytm 3: Algorytm obliczeniowy sieci ze wsteczną propagacją błędu.

Zastosowanie WPB wymaga rozszerzenia atrybutów węzłów o parametr δ, czyli $v = (X, y, \delta)$. Istotną różnicą w stosunku do standardowych sieci neuronowych uczonych poprzez WPB jest to, iż sieci ewolucyjne mogą mieć dużo bardziej swobodną strukturę, zawierającą różnego rodzaju połączenia. WPB w sieciach ewolucyjnych dotyczy tylko połączeń klasy 'B'. Najprostsze połączenie klasy B, oznaczone jako `b`, składa się z następujących atrybutów: $e_{ij} = (w, w_0, \eta, h)$, czyli wagi, wagi początkowej, współczynnika uczenia oraz funkcji uaktualnienia wagi, tj. reguły uczenia:

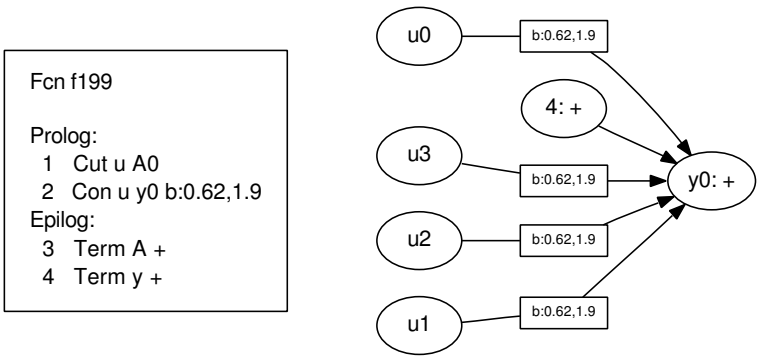
$$h_1(w, \eta, y_i, \delta_j) : \Delta w := \eta y_i \delta_j\ . \tag{4.16}$$

Bardziej elastyczny wariant połączenia (B) reprezentuje funkcję aktualizacji w postaci wyrażenia algebraicznego 3 zmiennych ($h_2(w, y_i, \delta_j)$), która może podlegać ewolucji poprzez mutację argumentu C w instrukcjach Con i Cone. Warto zwrócić uwagę, że gdy parametr η jest bliski zeru, lub też bardziej ogólnie funkcje h_1 i h_2 nie powodują zmian wagi, połączenie staje się nieplastyczne, ale wcale nie bezużyteczne — funkcjonuje ono wówczas jak połączenie ważone, gdzie waga jest wyznaczona przez wagę początkową, tj. $w \approx w_0$.

Algorytm 3 różni się od standardowego algorytmu WPB również brakiem pochodnej funkcji aktywacji w wyrażeniu na aktualizację wag połączeń. Wynika stąd, że te dwa algorytmu nie są sobie równoważne, a więc wszelkie właściwości standardowego algorytmu WPB, zwłaszcza odnośnie zbieżności, nie mogą być przypisane zaproponowanemu algorytmowi. O ile matematyczna analiza zaproponowanego wariantu WPB wykracza poza ramy tego punktu, o tyle możliwe i wskazane jest empiryczne sprawdzenie jak algorytm 3 sprawdza się w praktyce.

4.4.1 Problem klasyfikacji

Prostym przykładem wykorzystanie algorytmu WPB może być rozwiązanie dla problemu klasyfikacji, opisanego w §4.1.6. Wykorzystuje ono połączenia typu b, czyli wykorzystujące regułę uczenia h_1. Na rysunku 4.21 przedstawiono sieć stanowiącą rozwiązanie dla tego problemu, dla przypadku $n = 3$ (wejście u0 zamiast błędu ma przypisaną stałą wartość 1). Na połączeniach typu b zaznaczono ich współczynnik uczenia oraz wagę początkową. Sieć jest bardzo prosta i łatwa do znalezienia przez algorytm ewolucyjny. Co więcej, genotyp sieci koduje rozwiązanie dla dowolnego n, a więc jest ono doskonale skalowalne.



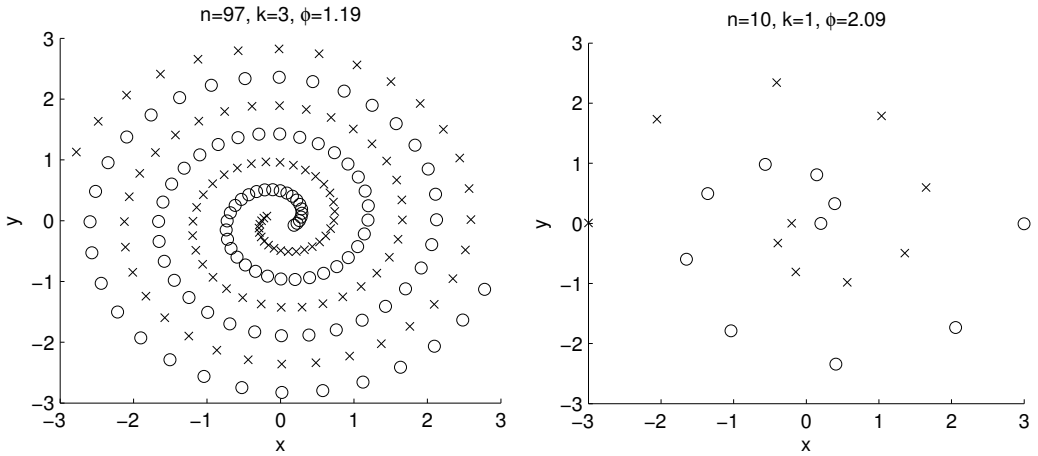
Rysunek 4.21: Ogólne rozwiązanie problemu „perceptron” (genotyp). Fenotyp sieci rozwinięty dla przypadku $n = 3$.

4.4.2 Problem spiral

Problem przeplatających się spiral (*intertwined spirals*) jest dobrze znanym problemem testowym w uczeniu maszynowym. Polega na nauczeniu agenta, w tym przypadku sieci, klasyfikacji punktów leżących wzdłuż dwóch przeplatających się spiral na płaszczyźnie:

$$\begin{aligned} s_1 &= \{(x_i, y_i) : x_i = r_i \cos\theta_i;\ y_i = r_i \sin\theta_i;\ i = 1, \ldots, n\}\ , \\ s_2 &= \{(x_i, y_i) : x_i = r_i \cos\theta_i + \pi;\ y_i = r_i \sin\theta_i + \pi;\ i = 1, \ldots, n\}\ . \end{aligned} \tag{4.17}$$

W biegunowym układzie współrzędnych, punkty są wyznaczone przez promień $r_i = \frac{i}{n}2.8 + 0.2$ i kąt $\theta_i = \frac{i}{n}k2\pi + \phi$, które z kolei uzależnione są od 3 parametrów — liczby punktów na spirali n, liczby obrotów k oraz przesunięcia fazowego $\phi \in [0; 2\pi)$. Przesunięcie ϕ jest zmienną losową, wyznaczaną przy każdym odświeżeniu problemu, a zatem problem jest stochastyczny. Rysunek 4.22 przedstawia dwa przykłady problemu — typowy: $n = 97$, $k = 3$, oraz prosty: $n = 10$, $k = 1$. Ze względu na złożoność obliczeniową, w eksperymentach wykorzystujemy uproszczoną wersję problemu.



Rysunek 4.22: Problem klasyfikacji przeplatających się spiral.

Przeplatające się spirale nie są liniowo separowalne, a więc rozwiązanie zadania wymaga sieci wielowarstwowej, lub w inny sposób bardziej złożonej niż standardowa jednowarstwowa sieć neuronowa. Zakłada się, że sieć ma tylko 3 epoki treningowe, czyli każda z $2n$ próbek uczących jest podawana 3-krotnie na interfejs sieci. W ramach epoki kolejność podawanych punktów jest przypadkowa. Wartość przystosowania sieci jest średnią z 5 prób, w których do nauczenia się jest inna instancja problemu ze względu na zmienną losową ϕ (choć instancje te są identyczne dla całej populacji w danym pokoleniu). Ponieważ zadanie jest stochastyczne, w każdym pokoleniu najlepsza sieć jest testowana nie na 5, a na 250 instancjach problemu.

Ponieważ celem eksperymentu jest zbadanie zdolności do uczenia sieci ewolucyjnych wykorzystujących RKS, obserwowana będzie jedynie zdolność do uczenia się na zbiorze treningowym, bez sprawdzenia zdolności do uogólniania.

Przeprowadzono 2 serie symulacji dla dwóch wariantów wykorzystywanych połączeń klasy B — połączenia typu b z predefiniowaną funkcją aktualizacji wag h_1 (wz. 4.16), oraz typu B z funkcją podlegającą ewolucji — h_2. W każdej serii wykonano 20 przebiegów ewolucyjnych, w których populacja $M = 2048$ i liczba pokoleń $\tau = 500$.

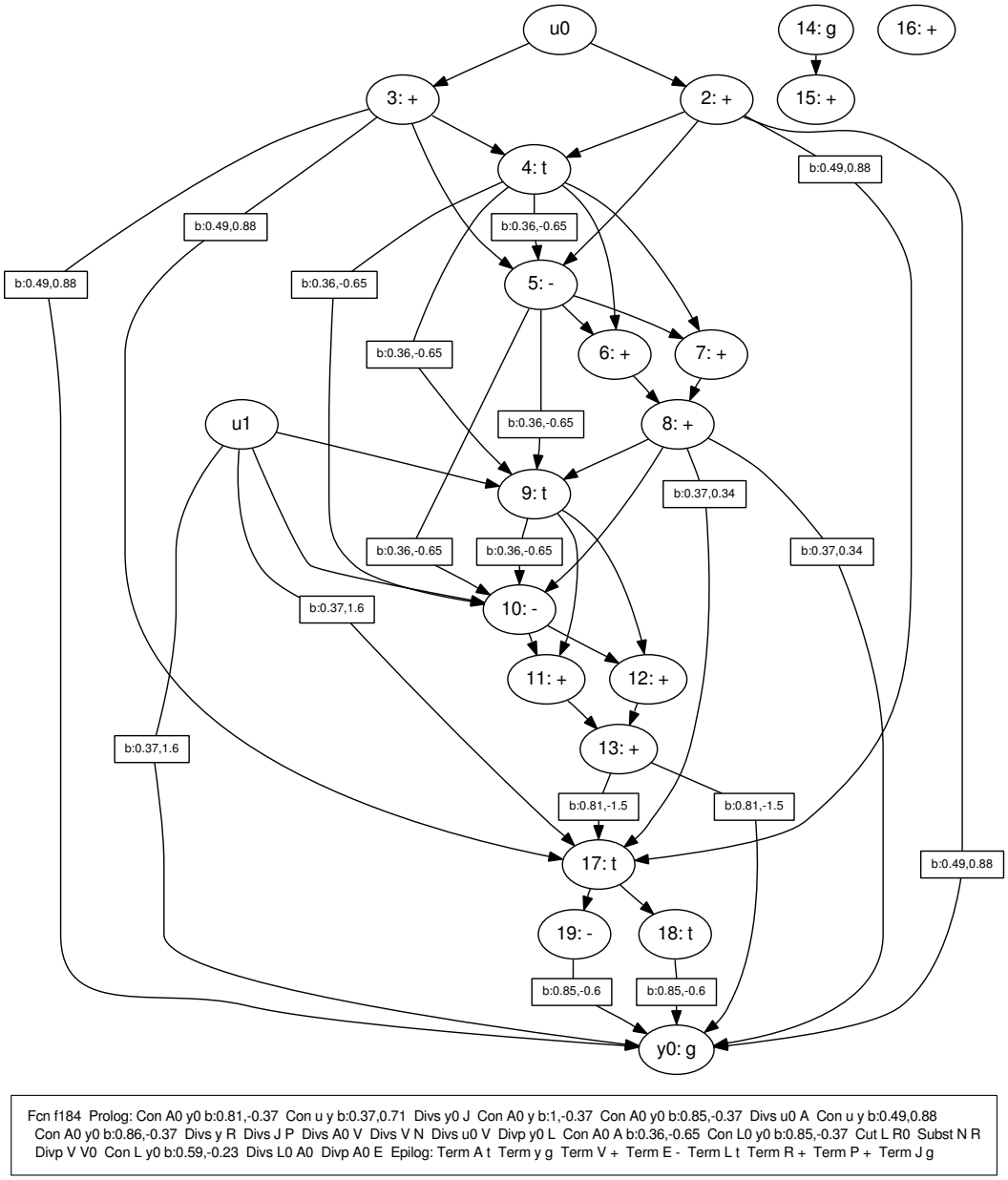
Rysunek 4.23 przedstawia wybraną sieć wykorzystującą połączenia typu b. Składa się ona z 18 węzłów ukrytych, z funkcjami aktywacji $-$, $+$, t (czyli tanh()) oraz funkcją sigmoidalną g w węźle wyjściowym. Struktura sieci wykazuje pewną nadmiarowość — kilka węzłów jest zupełnie niewykorzystanych; natomiast kilka innych (7 i 8 oraz 11 i 12) jest powielonych w tym sensie, że dokonuje tych samych obliczeń i zawsze ma jednakową wartość wyjściową. Nadmiarowość jest często spotykaną cechą wyewoluowanych sieci, którą można redukować bądź metodami analitycznymi (np. algorytmem wykrywania podsieci funkcjonalnie identycznych), lub kontynuując bieg ewolucyjny przy zwiększonej presji na rozmiar sieci — nie jest to jednak przedmiotem badań. Sieć zawiera 18 połączeń typu b, ale tylko 7 z nich faktycznie jest plastycznych — wynika to stąd, iż połączenia proste nie propagują błędu wstecz, a więc dla pozostałych połączeń błąd wynosi 0 i waga połączenia utrzymuje wartość początkową (drugi parametr połączenia na diagramie).

Rysunek 4.24 przedstawia najlepszą uzyskaną sieć wykorzystującą połączenia z funkcją h_2. Składa się ona z 22 węzłów ukrytych, z czego 5 jest zupełnie zbędnych. Sieć wykorzystuje 9 połączeń typu B, lecz tylko 3 są faktycznie plastyczne. Funkcja aktualizacji, jaka wyewoluowała dla owych trzech połączeń, a oznaczona na diagramie jako wyrażenie prefiksowe (- (*u1 u2) R) sprowadza się do:

$$h_2(w, y_i, \delta_j) : \Delta w := y_i \delta_j \ . \tag{4.18}$$

Zmienne u1 i u2 odpowiadają wartościom y_i i δ_j, natomiast R jest częścią mechanizmu pamięciowego dla drzew algebraicznych, który w tym przypadku redukuje się do 0 (patrz Suchorzewski, 2010a). Warto zwrócić uwagę, iż omawiane plastyczne połączenia, które występują w sieci 3-krotnie, w genotypie są zakodowane 1-krotnie. Ten fakt wskazuje na skalowalność kodowania.

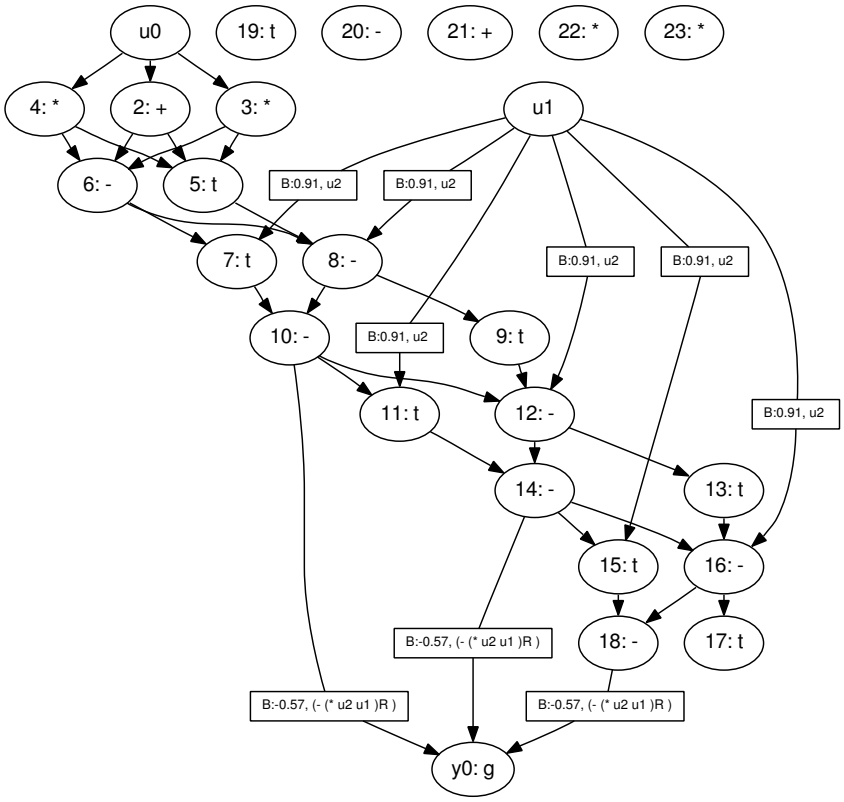
W przedstawionych dotąd sieciach wykorzystywano funkcje aktywacji ze zbioru $S_t = \{\mathsf{g}, \mathsf{h}, \mathsf{t}, +, -, *\}$, gdzie $h(\mathbf{x}) = (\sum_i x_i > 0.5)$ jest progową funkcją aktywacji. Ponadto wykorzystanie funkcji sinusoidalnej $s = \sin(\sum_i x_i)$ znacznie ułatwia znalezienie rozwiązanie zadania. Sieci uzyskujące wynik $J_t < 1$ (patrz niżej) są znajdowane zaledwie po kilkunastu pokoleniach . Rysunek 4.25 przedstawia wybraną sieć, wykorzystującą połączenia h_1 i sinusoidalną funkcję przejścia. Sieć składa się tylko z 9 węzłów ukrytych i ma strukturę zbliżoną



Rysunek 4.23: Rozwiązanie wykorzystujące połączenia typu b.

do warstwowej, lecz zawiera również połączenia bezpośrednie z węzłów wejściowych do wyjściowych i odznacza się pewną separacją pomiędzy węzłami wejściowymi a węzłami w warstwie ukrytej. Trzeba jednak podkreślić, że — jak zresztą zwykle — przedstawiona struktura jest tylko jedną z kilku charakterystycznych struktur otrzymanych w wyniku ewolucji.

Tabela 4.7 przedstawia wyniki testu wyewoluowanych sieci oraz dla porównania wyniki otrzymane dla standardowych SSN. Kryterium J_t oznacza

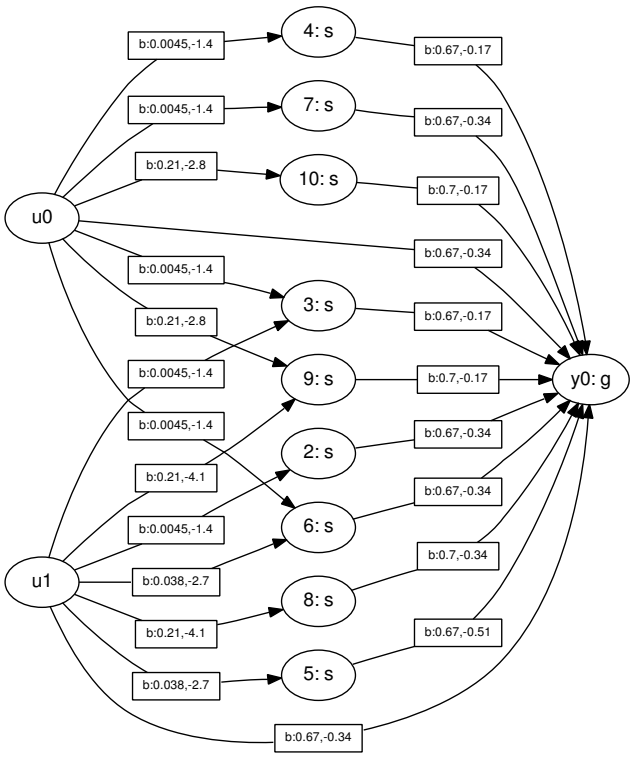


Fcn f9 Prolog: Divs y0 S Subst S0 I Divp I G Cut u1 A Divs u0 A Divs u0 A0 Subst A0 A0 Con u1 A B:0.46,u2 Con A y0 B:-0.57,(- (* u2 u1)R) Con u1 A0 B:0.46,(* (W u0)(* R -0.44)) Divs A0 A Divs y Y0 Subst y0 D Divs u0 A0 Divp A0 E Divs u0 Y0 Cut A0 D Cut A0 A0 Divp Y M Divp M M Epilog: Term A t Term D g Term Y + Term M * Term E - Term I t Term G -

Rysunek 4.24: Rozwiązanie wykorzystujące połączenia typu B.

liczbę błędnie zaklasyfikowanych próbek w ostatniej epoce treningu, uśrednioną dla 250 prób (± odchylenie standardowe). Sieć P_1 składa się z pojedynczej warstwy ukrytej o 20 sigmoidalnych neuronach oraz liniowym neuronie wyjściowym. Sieć jest uczona algorytmem wstecznej propagacji Levenberg-Marquardt'a. Pozostałe parametry są standardowe. W eksperymencie wykorzystano pakiet *Neural Network Toolbox* dla środowiska *Matlab*. Sieć P_2 różni się od P_1 tylko nieco inną architekturą — składa się z dwóch warstw ukrytych po 10 neuronów każda. Jak widać, wyewoluowane sieci radzą sobie znacznie lepiej od sieci generycznych. Sieć wykorzystująca swobodną funkcję aktualizacji uzyskała bezbłędny rezultat. Lepsze wyniki sieci ewolucyjnych są niewątpliwie efektem specjalizacji sieci do szczególnych warunków zadania, przede wszystkim spiralnego charakteru zadania i określonej, niewielkiej liczby epok podczas treningu.

Rysunek 4.26 przedstawia zbieżność procesu uczenia sieci przedstawionej na rysunku 4.25. Jak widać już po pierwszej epoce sieć klasyfikuje próbki poprawnie (przyjmując, że ciemny obszar odpowiada próbkom oznaczonym sym-



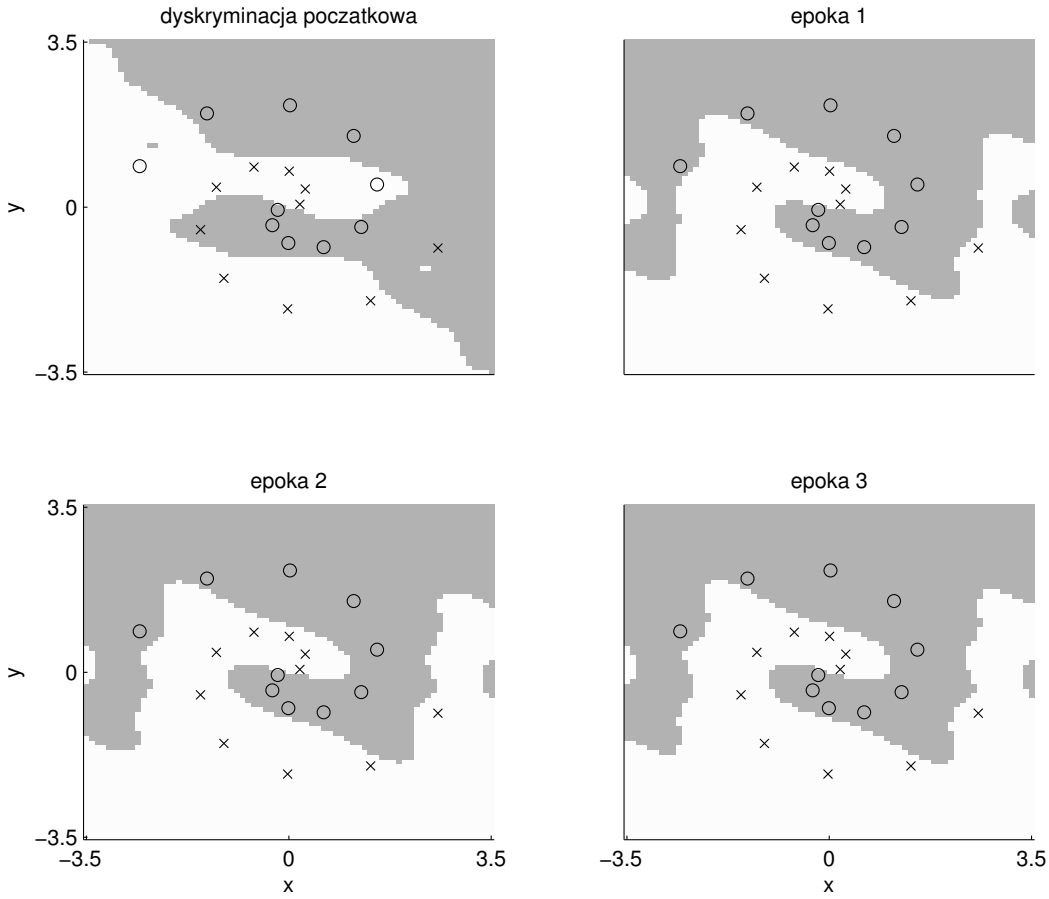
Fcn f156 Prolog: Con u A b:0.21,-1.4 Con u y0 b:0.67,-0.17 Divp u0 A Con A y0 b:0.7,-0.17 Con u1 A b:0.038,-1.3 Divp u0 A0 Cut A A Con u A b:0.0045,-1.4 Subst A0 A0 Divp A0 J Cut u1 J0 Subst A0 A0 Divp A0 F Cut u0 A0 Con u y0 b:0.52,-0.17 Con A0 y0 b:0.5,-0.17 Epilog: Term A s Term y g Term J s Term F s

Rysunek 4.25: Rozwiązanie wykorzystujące sinusoidalną funkcję aktywacji.

bolem $\circ$). Oczywiście nie zawsze zbieżność jest tak szybka. Charakterystyczną cechą przeanalizowanych wykresów zbieżności jest antysymetria dyskryminacji względem początku układu współrzędnych. Ponadto można również zaobserwować, że początkowa funkcja dyskryminacji wykazuje często charakterystyczne „zakręcenie” obszaru dyskryminacji w okolicy punktu $(0, 0)$, co jest również widoczne na przedstawionym wykresie. Wskazuje to na predyspozycję sieci do klasyfikacji spiral, potwierdzając hipotezę odnośnie ewolucyjnej specjalizacji sieci.

Tabela 4.7: Wyniki testu. P_1, P_2 — standardowe sieci neuronowe; b, B — sieci wyewoluowane, wykorzystujące połączenia klasy B; b+s — jak b przy czym sieć wykorzystuje węzły z sinusoidalnymi funkcjami przejścia.

	Sieć				
	P_1	P_2	b	B	b+s
J_t	1.25 ± 1.73	3.03 ± 2.21	0.07 ± 0.32	0.00 ± 0.00	0.01 ± 0.09



Rysunek 4.26: Zbieżność procesu uczenia sieci z rysunku 4.25.

4.5 Neuromodulacja

Instrukcje Con oraz Cone tworzą połączenia pomiędzy węzłami. Wewnętrzna dynamika takich połączeń, zarówno przy ewaluacji jak i uczeniu, może być określona dowolnie, choć w ramach oczywistych ograniczeń: wyrażenia na ewaluację i uczenie mogą wykorzystywać jedynie zmienne lokalne, tj. wartości obydwu połączonych węzłów, oraz lokalne parametry połączenia. Jest to wystarczające do uzyskania różnego rodzaju połączeń — ważonych, połączeń uczonych regułą Hebb'a, czy wsteczną propagacją, jednakże wiele innych interesujących rozwiązań nie jest możliwa w takim układzie. Neuromodulacja polega na dołączenia trzeciego (a w ogólności dowolnej ilości) węzłów do połączenia i przez to pozwala na uzyskanie znacznie bogatszej dynamiki takiego połączenia, czy to w fazie ewaluacji czy też uczenia. Połączenie modulowane łączy dwa węzły podstawowe — wejściowy i wyjściowy, oraz węzeł modulujący.

Dobrym przykładem możliwości uzyskanych dzięki neuromodulacji jest reguła uczenia Hebb'a ze wzmocnieniem (Chen et al., 2007):

$$\Delta w = \eta r x (y - w) \ , \tag{4.19}$$

gdzie w oznacza wagę połączenia, η współczynnik uczenia, x i y wartości aktywacji węzłów podstawowych, zaś r sygnał modulujący, czyli wartość aktywacji węzła modulującego.

Dzięki neuromodulacji możliwe jest wyrażenie wielu innych reguł uczenia. Ogólna postać reguły uczącej jest następująca (Chen et al., 2007):

$$\Delta w = (1 - \epsilon) w + \eta r x \ , \tag{4.20}$$

gdzie ϵ jest współczynnikiem zapominania, zaś x oznacza sygnał węzła wejściowego. W odróżnieniu od połączenia konwencjonalnego, sygnał r może pochodzić z dowolnego innego węzła w sieci. W zależności od tego skąd pochodzi sygnał r, zrealizowane mogą być różne reguły uczenia (Chen et al., 2007):

- Uczenie Hebb'a lub anty-Hebb'a, gdy $r = y$ lub $r = -y$, gdzie y oznacza wartość aktywacji węzła wyjściowego.

- Reguła delty, gdy $r = \delta = y - z$, gdzie z oznacza pożądaną wartość na wyjściu, tj. węźle y.

- Reguła korekcji błędu LMS, gdy $r = \sum_i w_i x_i - z$, gdzie i iteruje po wszystkich neuronach warstwy, zaś z stanowi sygnał uczący. Wagi połączeń w warstwie zbiegają do wartości minimalizujących błąd $(z - \sum_i w_i x_i)^2$.

- Uczenie PCA, gdy $r = \sum_i w_i x_i$, składowa główna macierzy kowariancji $C = \langle \mathbf{x}\mathbf{x}^T \rangle$ może być uzyskana po normalizacji.

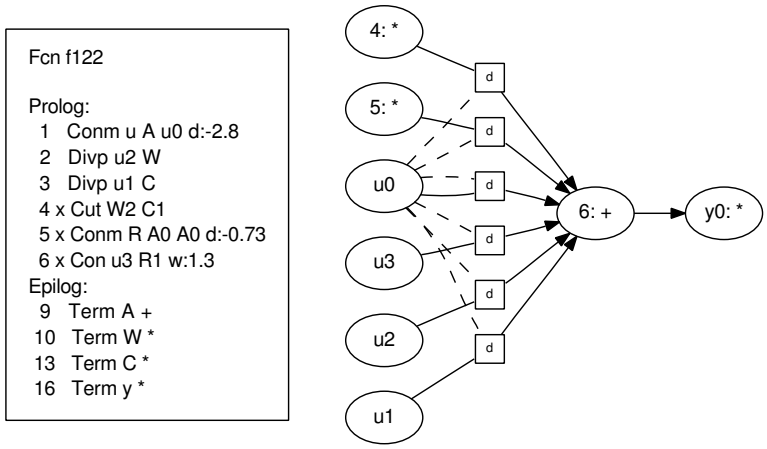
- Uczenie metodą różnic czasowych (*TD learning*), gdy $r = \epsilon$, gdzie ϵ jest błędem TD oszacowywanym metodą TD (zrealizowaną za pomocą pewnej podsieci).

- Uczenie asocjacyjne, gdy $r = x$, tak że wagi połączeń w grupie zbiegają do macierzy autokorelacji: $w_{ij} = 1/T \sum_{t=1}^{T} x_i(t) x_j(t)$, gdzie t oznacza krok czasowy.

- Uczenie asocjacyjne w czasie, gdy $r(t) = x(t+1)$. W tym przypadku wagi połączeń zbiegają do wartości: $w_{ij} = (1/T - 1) \sum_{t=1}^{T-1} x_i(t) x_j(t+1)$, które stanowią macierz kowariancji w czasie.

Rysunek 4.27 przedstawia sieć stanowiącą rozwiązanie dla problemu klasyfikacji (patrz §4.1.6), która wykorzystuje połączenia modulowane. Połączenia są tworzone przez instrukcję Conm, która jest podobna do instrukcji Con, przy czym posiada ona dodatkowy argument wybierający węzły modulujące.

Połączenia tworzone są dla każdej trójki węzłów pasujących do argumentów instrukcji Conm, pod warunkiem, że nie istnieje jeszcze połączenie pomiędzy węzłami podstawowymi — w tej sytuacji połączenie nie jest tworzone; tak więc połączenia wielokrotne pomiędzy tą samą parą węzłów podstawowych są wykluczone, choć jest to oczywiście tylko pewien wariant implementacyjny.



Rysunek 4.27: Sieć stanowiąca rozwiązanie problemu klasyfikacji dla $n = 3$, wykorzystująca neuromodulację.

W przedstawionej sieci, błąd klasyfikacji podawany jest na wejście u0. Połączenie modulowane, oznaczone symbolem d wykonuje zwyczajne ważenie w fazie ewaluacji, natomiast podczas uczenia realizuje następującą regułę:

$$w(t+1) = w(t) + x(t-1)z(t) \ , \tag{4.21}$$

gdzie z jest wejściem modulującym, czyli błędem klasyfikacji dla poprzedniej próbki. Alternatywnie, wejście u0 mogłoby reprezentować wartość pożądaną klasyfikacji i wówczas reguła uczenia miałaby postać:

$$w(t+1) = w(t) + x(t-1)(y(t-1) - z(t)) \ . \tag{4.22}$$

4.6 Sieci ewolucyjne a generyczne

Jedna z tez pracy mówi, iż istnieją takie problemy, dla których metoda RKS pozwala generować sieci uczące się efektywniej aniżeli standardowe sieci neuronowe. Określenie „standardowe sieci neuronowe” oznacza tutaj sieci neuronowe, jakie zostałyby zaproponowane przez eksperta dla rozwiązania danego problemu. Możliwość wyewoluowania sieci uczących się efektywniej niż sieci generyczne została już częściowo pokazana w §4.1.6 i §4.4.2. W niniejszym punkcie przedstawimy eksperyment, który wyraźnie pokaże, że teza jest prawdziwa, a także pozwoli lepiej zrozumieć dla jakich problemów sieci ewolucyjne są szczególnie odpowiednie.

Problem nazwany roboczo „ruchomy pas” jest następujący. W 2-wymiarowej przestrzeni $[-0.5; 0.5]^2$ losowany jednostajnie jest punkt (x_0, y_0). Wyznacza on podprzestrzeń $S_1 = \{(x, y) : x_0 - 0.5 < x < x_0 + 0.5\}$ punktów klasy 1. Pozostałe punkty z przestrzeni zostają przydzielone do klasy 0. Zadanie stojące przed siecią polega na tym, aby zaklasyfikować $m = 100$ kolejno podawanych losowych punktów z przestrzeni jednostkowej $S = [-1; 1]^2$ popełniając jak najmniejszą liczbę błędów.

Inaczej niż w przypadku typowych zadań klasyfikacji, gdzie maszyna ma do nauczenia się klasy próbek na podstawie pojedynczego i często bardzo ograniczonego zbioru uczącego, powyższe zadanie jest generyczne; w tym sensie, że pozwala wygenerować dowolną liczbę opisanych powyżej prób (*trials*), czyli zbiorów danych. Zbiory te odznaczają się zarówno cechami stałymi — szerokość pasa wynosi 1 i jest to pas zorientowany wzdłuż osi y, oraz cechą zmienną — położenie pasa jest dla każdej próby losowe. Jak to było omówione w §2.1, mocną stroną sieci ewolucyjnych jest zdolność do połączenia adaptacji ewolucyjnej, która pozwala na przystosowanie do stałych cech zadania, oraz do uczenia, która pozwala na szybkie uchwycenie cech zmiennych środowiska. Co więcej, sam mechanizm uczenia może być ewolucyjnie przystosowany do nauczenia się „specyficznie zmiennych” cech środowiska, tak jak na przykład w sformułowanym zadaniu.

Wykonano 20 przebiegów ewolucyjnych, dla populacji $M = 4096$ i liczby pokoleń $\tau = 300$. Każdy osobnik oceniany był na podstawie trzech prób, natomiast najlepszy osobnik w pokoleniu był testowany na 90 próbach. Jako kryterium przystosowania przyjęto średnią częstość błędów w próbie. Istotne jest również to, że interfejs sieci rozszerzony był o wejście z wartością błędu klasyfikacji poprzedniej próbki:

$$\mathsf{u0} = e_i = c_{i-1} - f_{i-1} \ , \tag{4.23}$$

gdzie c_i oznacza klasę i-tej próbki, zaś f_i wartość wyjściową sieci; ponadto przyjmuje się $e_1 = 0$.

W przypadku sieci generycznych wypróbowano kilka architektur sieci jednokierunkowych wielowarstwowych. Wykorzystano przy tym pakiet *Neural Network Toolbox* dla środowiska *Matlab*. Sieci uczone były poprzez adaptację sekwencyjną, a dokładnie funkcję ‘adapt’, która jest bardziej odpowiednia dla postawionego zadania, niż trening wsadowy. Adaptacja sekwencyjna pozwala uaktualniać wagi sieci dla każdej nowej próbki, tak więc sieć może uczyć się na bieżąco, popełniając z czasem coraz mniej błędów. Aby zwiększyć skuteczność uczenia, adaptacja była wykonywana nie tylko dla ostatnio podanej, j-tej próbki, ale dla wszystkich dostępnych próbek, czyli całej sekwencji $1, \ldots, j$. Aktualizacja wag dokonywana była na podstawie błędu metodą gradientową. Pozostałe parametry sieci generycznych pozostały domyślne.

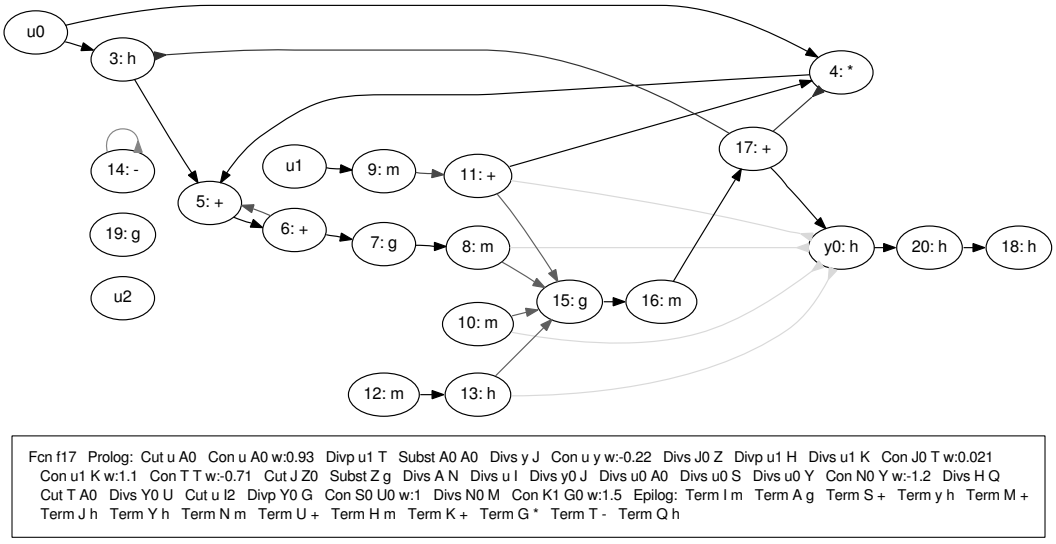
Spośród 20 przebiegów, 6 dało rozwiązania uzyskujące wynik poniżej 20 błędów w próbie. Najlepsze rozwiązanie uzyskało wynik 5.6 błędnych klasy-

Tabela 4.8: Wyniki testu. FF5, FF10, FF55, FF10s — jednokierunkowe standardowe sieci neuronowe z, odpowiednio, 5 i 10 neuronami w warstwie ukrytej, oraz dwoma warstwami po 5 neuronów. E — najlepsza sieć wyewoluowana.

	Sieć			
	FF5	FF10	FF55	E
J_t	24.3 ± 9.4	21.9 ± 7.7	25.3 ± 9.2	5.6 ± 2.4

fikacji w osobnym teście na 90 próbach. Porównanie z wynikami uzyskanymi przez sieci generyczne przedstawia tabela 4.8. Jak widać, sieć ewolucyjna rozwiązuje zadanie znacznie skuteczniej niż sieci generyczne ($p_{\text{val}} < 10^{-6}$, test permutacji).

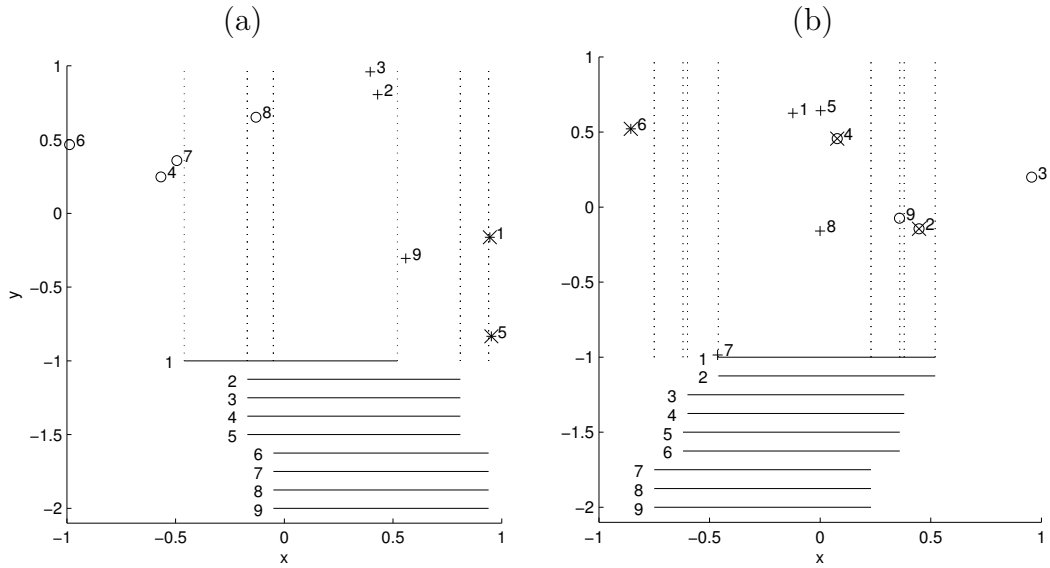
Rysunek 4.28 przedstawia najlepszą wyewoluowaną sieć. Jak widać, wiele węzłów w sieci jest zbędnych lub niewykorzystanych; w szczególności niewykorzystane jest wejście u2, co jest poprawnym odzwierciedleniem warunków zadania. Sieć jest w dużym stopniu jednokierunkowa, występują jedynie 3 połączenia zwrotne, które wpływają na proces uczenia. Podobnie jak w problemie perceptron (§4.1.6), algorytm uczenia sieci jest cechą całkowicie autonomicznie wyewoluowaną i polega na samej strukturze i dynamice działania sieci.



Rysunek 4.28: Wyewoluowana sieć dla zadania „ruchomy pas”. Wagi o małej bezwzględnej wartości oznaczono odcieniami szarości, wagi ujemne oznaczono odwrotnym grotem.

Rysunek 4.29 przedstawia przebieg uczenia sieci na przykładzie dwóch zbiorów danych. Zainicjalizowana sieć zalicza do klasy S_1 punkty leżące w pasie wokół osi y o szerokości około 1, co jest bardzo odpowiednim punktem wyjścia. Przesunięcie granic klasyfikacji następuje w sytuacjach błędnych zaklasyfiko-

wań, kiedy to błąd u0 jest różny od zera. W przypadku wykresu 4.29a przesunięcie zachodzi w kroku 2., tj. po błędnym zaklasyfikowaniu punktu 1, oraz w kroku 6. Warto odnotować, że adaptacja ma miejsce zarówno po błędach pierwszego jak i drugiego rodzaju, tj. błędach w zaklasyfikowaniu punktów obydwu klas. Można jednak zaobserwować, że uczenie nie jest optymalne, widać to zwłaszcza w kroku 7 wykresu (b), gdzie po błędnej klasyfikacji punktu 6, granice się przesuwają, lecz niewystarczająco daleko by objąć ten punkt.



Rysunek 4.29: Początkowy przebieg uczenia sieci na dwóch przykładach.

4.7 Problemy nieregularne

W tym punkcie przedstawiono eksperymenty i rozwiązania dla problemów nieregularnych, wyewoluowane przy użyciu proponowanej metody.

4.7.1 Problem sztucznej mrówki

Problem sztucznej mrówki jest dobrze znanym problemem testowym w obszarze uczenia maszynowego, a zwłaszcza metod ewolucyjnych (patrz Koza, 1992). Polega on na znalezieniu programu dla agenta w sztucznym środowisku, który pozwoli mu zebrać pożywienie rozłożone wzdłuż pewnej ścieżki, zwanej „szlakiem Santa Fe” (rys. 4.30). Sztuczna mrówka ma jeden sensor, przyjmujący wartość 1, gdy na polu przed nią znajduje się pożywienie i 0 w przeciwnym przypadku. Trzy wyjścia odpowiadają trzem możliwym akcjom: obrót w lewo, ruch do przodu, obrót w prawo. Przyjmuje się, że podjęta jest ta akcja, której wyjście jest największe, a w przypadku równych wartości decyduje podana

powyżej kolejność (czyli lewo, do przodu, prawo). Mrówka zbiera pożywienie automatycznie, gdy tylko wejdzie na pożywne pole. Na zebranie wszystkich ziarenek mrówka ma 400 kroków. Tak więc zarówno agent, jak i środowisko są całkowicie deterministyczne. Problem jest w dużym stopniu statyczny, choć nie całkowicie, gdyż w trakcie próby agent wpływa na środowisko. Teoretycznie możliwy jest program generujący sekwencję akcji bez wykorzystania informacji wejściowej agenta, jednak informacja ta z pewnością jest przydatna.

```
+--------------------------------+
|>***                            |
|   *                            |
|   *                     ***    |
|   *                    *    *  |
|   *                    *    *  |
|   **** *****        **        |
|           *                *   |
|           *                    |
|           *       *            |
|           *       *        *   |
|           *       *            |
|           *       *            |
|           *                *   |
|           *       *     ***    |
|           *       *  *         |
|               *                |
|           *   *       *        |
|           *   *          *     |
|           *                    |
|           *   *                |
|           *            *       |
|           *         *          |
|   **  *****    *               |
| *             *                |
| *             *                |
| *      *******                 |
| *     *                        |
|       *                        |
|  ****                          |
+--------------------------------+
```

Rysunek 4.30: Szlak Santa Fe. W środowisku o wymiarach 32×32 rozmieszczonych jest 81 pakietów pożywienia (∗). Sztuczna mrówka (>) startuje z lewego górnego rogu i jest zorientowana „na wschód”.

Przeprowadzono dwie serie po 30 przebiegów dla dwóch wariantów ustawień AE, różniących się zbiorami dozwolonych funkcji przejść i rodzajów połączeń. W pierwszym wariancie wykorzystano funkcje przejść $S_t = \{+, -, *, \mathsf{h}, \mathsf{m}\}$, gdzie funkcja m wyciąga średnią ze wszystkich wejść, tj.:

$$m(\mathbf{x}) = \frac{1}{|\mathbf{x}|} \sum_i x_i \ , \tag{4.24}$$

natomiast przytaczana już wcześniej funkcja progowa $h(\mathbf{x}) = ((\sum_i x_i) > 0.5)$; czyli przyjmuje wartość 1, gdy suma wejść przekracza 0.5, zaś 0 w pozostałych przypadkach. W drugim wariancie zbiór funkcji przejść ograniczono do $S_t = \{+, -, *\}$, ale włączono połączenia ważone.

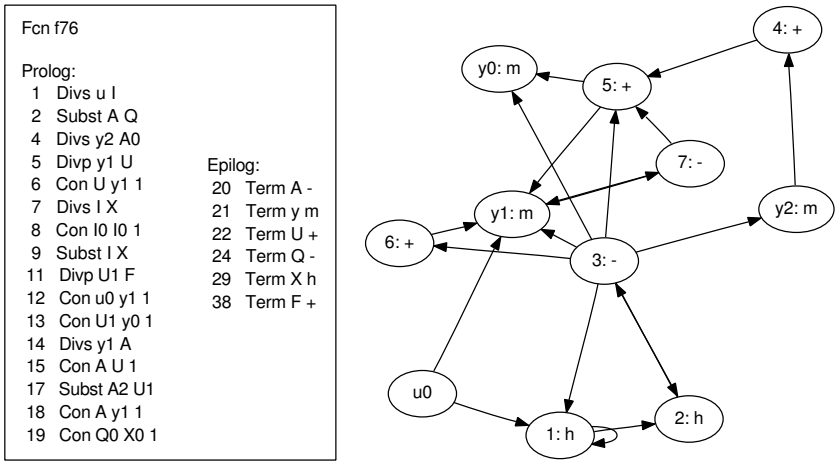
Tabela 4.9 przedstawia wyniki symulacji, w porównaniu z wynikami PG dla tego zadania (Koza, 1992). Kolumna P_S oznacza częstość znajdowania rozwiązania dla określonych parametrów przebiegu, tj. wielkości populacji (M) oraz liczby pokoleń (τ). Kolumna CE natomiast, oznacza koszt obliczeniowy (*computational effort*), liczony tutaj jako liczba osobników jaka musi być oceniona, aby uzyskać rozwiązanie z prawdopodobieństwem 99% w i-tym pokoleniu,

Tabela 4.9: Porównanie RKS i PG pod względem częstości sukcesu (P_S) i kosztu obliczeniowego (CE) dla problemu sztucznej mrówki.

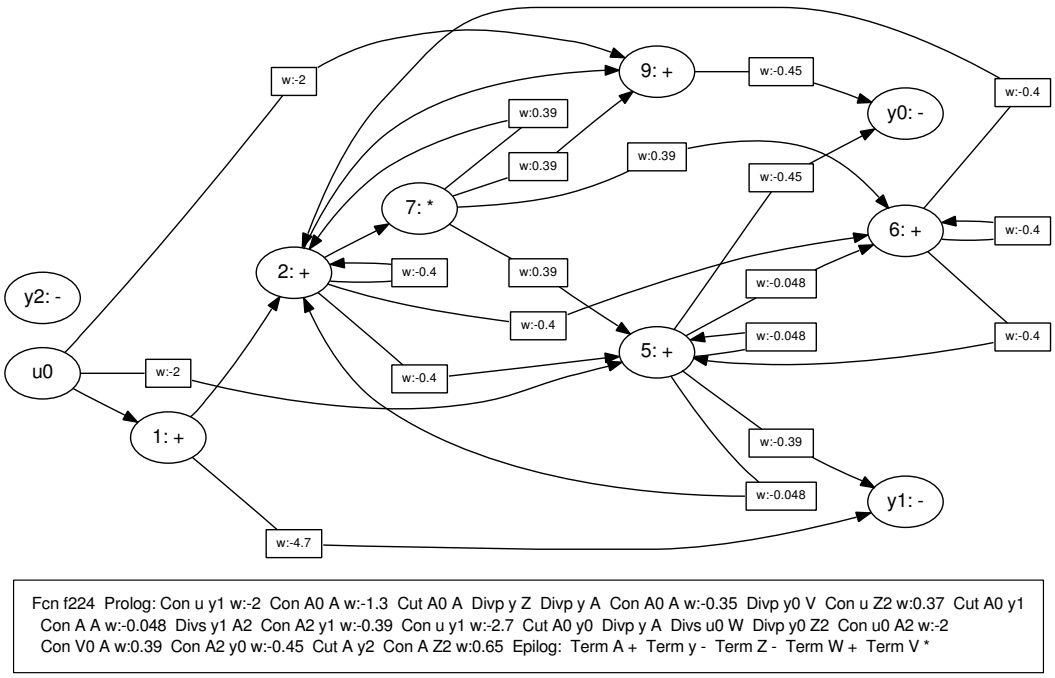
	M	τ	P_S	CE [$\times 10^3$]
PG	500	50	0.16	450
RKS 1	4096	200	0.2	11 750
RKS 2	4096	200	0.3	8 377

dla określonych parametrów algorytmu, a w szczególności rozmiaru populacji M. Indeks pokolenia i, dobrany jest *a posteriori* tak, aby minimalizować koszt obliczeniowy. Szczegółowe omówienie tego wskaźnika można znaleźć np. w (Christensen and Oppacher, 2002). Choć koszt obliczeniowy odznacza się pewnymi niepożądanymi cechami jak brak możliwości oszacowania marginesu niepewności, pozwala zgrubnie oszacować wydajność algorytmu. I tak z tabeli można odczytać, że choć szansa na uzyskanie rozwiązania jest w przypadku PG nieco niższa, znacznie niższy jest koszt obliczeniowy uzyskania rozwiązania. Taki rezultat nie powinien dziwić, zważywszy, że PG stosuje dla tego problemu specjalnie dobrany zestaw operacji, a ponadto problem nie nosi cech regularnego.

Rysunki 4.31 oraz 4.32 przedstawiają najbardziej zwięzłe wyewoluowane rozwiązania dla obydwu wariantów, przy czym w przypadku sieci ważonej usunięto dla przejrzystości nieefektywne fragmenty genotypu i fenotypu.



Rysunek 4.31: Przykład rozwiązania problemu sztucznej mrówki za pomocą sieci nieważonej.



Rysunek 4.32: Przykład rozwiązania problemu sztucznej mrówki za pomocą sieci ważonej.

4.7.2 Problem podwójnego odwróconego wahadła

Kolejnym szeroko wykorzystywanym problemem testowym jest sterowanie podwójnym odwróconym wahadłem. Fizyczny model wahadła jest następujący. Na wózku o masie 1kg, znajdującym się na bieżni o długości 4.8m, zamontowane są obok siebie dwa wahadła o masie 0.1 i 0.01kg oraz długości odpowiednio 0.5 i 0.05m. W stanie początkowym, wózek znajduje się w środku bieżni; krótsze wahadło jest postawione pionowo, zaś dłuższe pod losowo wybranym kątem z przedziału $\theta_1 \in [-7°; 7°]$. Model symulowany jest metodą Runge-Kutta'ego 4. rzędu, gdzie krok symulacji wynosi 0.01s (Gomez and Miikkulainen, 1999). Zadanie polega na sterowaniu siłą przykładaną do wózka $F \in [-10; 10]$N, tak, aby wózek nie wyjechał poza bieżnię, a wahadła utrzymały się w górze, w zakresie $[-36°; 36°]$. Siła obliczana jest co drugi krok symulacji, czyli co 0.02s. Czas trwania próby wynosi $K = 10\ 000$ kroków, a więc 100s czasu symulowanego. Za kryterium przystosowania przyjmuje się $J = \log(K/k)$, gdzie k oznacza liczbę kroków utrzymania układu w pożądanym stanie.

Rozpatrywane są 2 warianty tego zadania. W pierwszym wariancie na wejścia sieci podawane są (w kolejności): wartość stała 1 (bias), położenia wózka, prędkość wózka, wychylenie dłuższego wahadła i jego prędkość obrotowa i w końcu wychylenie drugiego wahadła i jego prędkość — w sumie 7 wejść. Wa-

Tabela 4.10: Parametry AE dla zadania sterowania podwójnym wahadłem.

	Wariant	
	7-wejściowy	4-wejściowy
Populacja	512	2048
Węzły	$+$, $-$, $*$, g (sigmoida)	$+$
Połączenia	ważone	
Topologia	jednokierunkowa	rekurencyjna
Warunek końcowy	$\tau = 200$ pokoleń lub rozwiązanie	

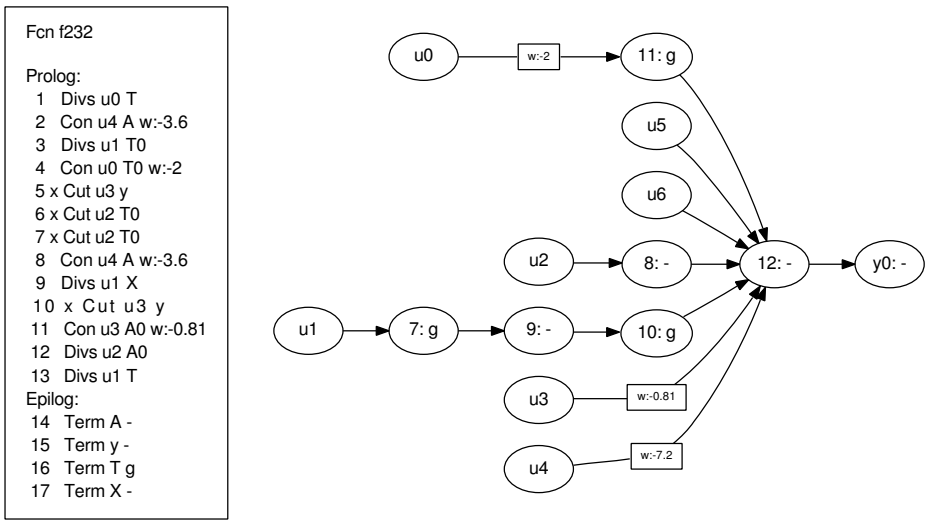
Tabela 4.11: Porównanie metod pod względem średniej liczby ewaluacji do rozwiązania. Odnośnie przytoczonych metod patrz (Stanley and Miikkulainen, 2002).

	Wariant	
	7-wejściowy	4-wejściowy
Programowanie ewolucyjne	307 200	–
Kodowanie bezpośrednie	80 000	–
SANE	12 600	–
Cellular Encoding	–	840 000
ESP	3 800	169 466
NEAT	3 600	33 184
RKS	32 686	270 280

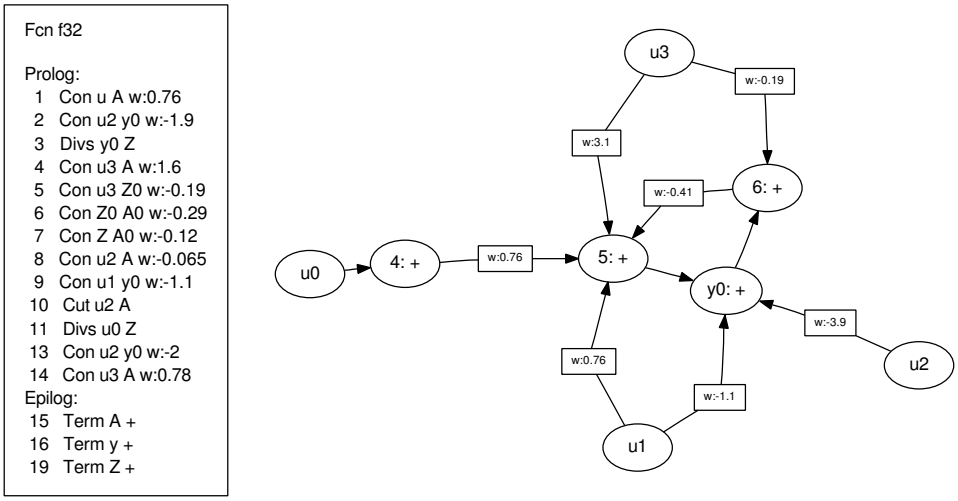
riant ten zwany jest wariantem Markov'a, gdyż o układzie dostępna jest pełna informacja. W drugim wariancie, prędkości są nieobserwowalne, a więc wejścia są tylko 4.

W eksperymencie przeprowadzono po 50 przebiegów dla obydwu wariantów zadania. Najważniejsze parametry algorytmu dla obydwu wariantów przedstawiono w tabeli 4.10. Porównanie uzyskanych wyników z wynikami innych metod pod względem kosztu obliczeniowego, liczonego tym razem jako średnia liczba ewaluacji osobników niezbędna do uzyskania rozwiązania, przedstawiono w tabeli 4.11. W wariancie 7-wejściowym uzyskano rozwiązanie we wszystkich 50 przebiegach, natomiast w wariancie 4-wejściowym w 39 przebiegach; tutaj wynik został obliczony jako suma wszystkich ewaluacji podzielona przez 39. Wyniki dla pozostałych metod zostały zaczerpnięte z (Stanley and Miikkulainen, 2002). Trzeba jednak podkreślić, że porównanie ma charakter orientacyjny, gdyż warunki zadania nie były identyczne we wszystkich eksperymentach.

Na rysunkach 4.33 i 4.34 przedstawiono najbardziej zwięzłe z wyewoluowanych rozwiązań dla obydwu wariantów zadania.



Rysunek 4.33: Sterownik 7-wejściowy dla podwójnego odwróconego wahadła.



Rysunek 4.34: Sterownik 4-wejściowy dla podwójnego odwróconego wahadła.

Rozdział 5

Podsumowanie i wnioski

Ewolucyjne sztuczne sieci neuronowe — lub nieco ogólniej ewolucyjne sieci obliczeniowe — stanowią interesujące i obiecujące podejście do rozwiązywania szerokiego zakresu problemów. Już ponad dekadę temu pokazano, że kombinacja adaptacji ewolucyjnej oraz uczenia — dwóch podstawowych form adaptacji — może prowadzić do znacznie bardziej inteligentnych systemów, aniżeli poleganie na sieciach neuronowych lub algorytmach ewolucyjnych z osobna (Yao, 1999).

W ramach pracy zaproponowano nową metodę ewolucji sieci obliczeniowych, nazwaną Rozwojowe Kodowanie Symboliczne, której główną częścią jest nowatorska reprezentacja genetyczna sieci, czyli sposób kodowania i dekodowania sieci. Zaproponowane kodowanie należy do rodzaju reprezentacji pośrednich, bądź rozwojowych, które odznaczają się tym, że fenotyp — w tym przypadku sieć — są generowane na podstawie genotypu w dość złożonym procesie rozwoju.

W pracy postawiono 4 tezy odnośnie opracowanej metody:

1. Metoda umożliwia wygenerowanie rozwiązania dla dowolnego problemu obliczalnego w sensie Turing'a.

2. Metoda umożliwia syntezę adaptacyjnych sieci odznaczających się skalowalnością i modularnością.

3. Metoda umożliwia syntezę sieci uczących się co najmniej równie efektywnie jak sztuczne sieci neuronowe.

4. Istnieją takie problemy, dla których metoda pozwala generować sieci uczące się efektywniej aniżeli standardowe sieci neuronowe.

Tezy 1 i 3 mają charakter teoretyczny i ich dowód okazuje się bardzo krótki (patrz §1 i §3.6). Dużo więcej pracy zajęło wykazanie prawdziwości tez 2 i 4, których dowody mają charakter empiryczny i polegają na badaniach eksperymentalnych.

W ujęciu niniejszej pracy, skalowalność oznacza zdolność do odzwierciedlenia regularności problemu w fenotypie sieci i uchwycenia tej regularności w jej genotypie, i dzięki temu efektywniejsze rozwiązywanie problemów o wzrastającym rozmiarze. Regularność problemu można tutaj rozumieć jako regularność jego rozwiązania, tj. takich cech sieci obliczeniowej jak symetria, powtarzalność, powtarzalność z wariacją, rekurencja, gładkość czy samo-podobieństwo.

W §4.1.2 wyewoluowano genotyp, generujący sieci stanowiące rozwiązania problemu parzystości dla dowolnej liczby wejść. Podobnie w §4.1.1 otrzymano doskonale skalowalne rozwiązanie dla problemu symetrii. O „doskonałej skalowalności" mówimy, jeśli pojedynczy genotyp zdolny jest do generowania rozwiązań dla różnych instancji problemu, a ściślej dla różnej liczby wejść. Skalowalne, choć niedoskonale, rozwiązania wyewoluowano również dla problemu topologii docelowej (§4.1.4), dyskryminacji wizualnej (§4.1.5) oraz uczenia nadzorowanego w zadaniu klasyfikacji, czyli tzw. problemu „perceptron" (§4.1.6). Skalowalność rozwiązań dla tych problemów nie ma tak mocnego charakteru jak w przypadku parzystości bądź symetrii, tym niemniej jest znacząca. Polega ona na tym, że rozmiar genotypu rozwiązań oraz nakład obliczeniowy potrzebny na ich znalezienie wzrasta wolniej aniżeli rozmiar problemu bądź fenotypu rozwiązań. Przykładowo, dla problemu uczenia nadzorowanego w zadaniu klasyfikacji rozmiar genotypu rozwiązań wzrasta tylko logarytmicznie wraz z rozmiarem fenotypu, zaś koszt obliczeniowy potrzebny dla rozwiązania kolejnych instancji problemu nie wzrasta wraz z jego wymiarowością, tj. liczbą wejść.

Istotną składową skalowalności, zarówno samego kodowania jak i generowanych rozwiązań jest modularność. Mówimy, że reprezentacja genetyczna wspiera modularność, jeśli pozwala generować wiele powtarzających się w fenotypie podsieci (modułów) na podstawie jednego fragmentu kodu genetycznego oraz jeśli umożliwia bądź ułatwia wymianę pomiędzy osobnikami fragmentów genotypu kodujących moduły fenotypu sieci. Blisko związane z modularnością jest pojęcie hierarchii, które oznacza rekursywne złożenie struktury i/lub funkcji. Hierarchia w systemie pozwala konstruować złożone sieci za pomocą kombinacji podsieci prostszych, a te z kolei konstruować z jeszcze bardziej prymitywnych. Modularność i hierarchia są jawnie wspierane przez RKS, gdyż genotyp ma w nim postać drzewa procedur. Procedury te zdolne są do generowania wielu podsieci w fenotypie, a ponadto są łatwo wymienialne pomiędzy osobnikami. To sprawia, że możliwe jest wyciągnięcie korzyści z komunikacji pomiędzy populacjami równolegle rozwiązującymi zestaw zadań.

Wsparcie proponowanego kodowania dla modularności zademonstrowano na przykładzie problemu parzystości (§4.1.2) oraz problemu perceptron (§4.1.6). Wykazano przy tym, że modularność genotypowa, czyli wykorzystanie procedur jest kluczowe dla rozwiązania problemu perceptron dla $n > 1$. W osobnym eksperymencie wykonano serię symulacji, w których wykorzystanie procedur było zablokowane. Wyniki przebiegów były znacznie gorsze, gdyż

nie znaleziono żadnego rozwiązania dla instancji $n > 1$. Można więc wnioskować, że procedury oraz możliwość ich wymiany pomiędzy osobnikami (poprzez operator krzyżowania), znacząco przyczyniły się do efektywnego rozwiązania problemu klasyfikacji. Modularność i płynące z niej korzyści zostały również wyraźnie pokazane na przykładzie problemu ewolucji podobnych do siebie 4-argumentowych funkcji logicznych (§4.3). Spośród trzech wariantów algorytmu, najlepsze rezultaty uzyskał ten wariant, w którym zarówno wykorzystanie procedur jak i migracja osobników pomiędzy populacjami rozwiązującymi różne instancje problemu były dozwolone.

W §4.2 przeprowadzono szereg eksperymentów, które równocześnie wykazały zdolność metody do generowania skalowalnych i modularnych topologii sieci, przy czym modularność była tutaj mierzona według miary (Newman and Girvan, 2004; Newman, 2004). Problem polegał tutaj na znalezieniu genotypów sieci charakteryzujących się określonymi parametrami topologii, przede wszystkim określoną liczbą węzłów w zależności od zadanej liczby wejść sieci, określonym rozkładem stopni węzłów sieci oraz jak największym wskaźnikiem modularności. Wyewoluowane genotypy w dużym stopniu spełniały postawione wymagania.

Zdolność do generowania sieci modularnych, rozpatrywana w nieco innym ujęciu, a mianowicie jako „lokalizacja funkcji wewnątrz zamkniętej jednostki, która w sieci ma postać grupy węzłów silnie połączonych pomiędzy sobą i słabo połączonych z węzłami spoza tej grupy", była testowana w §4.1.3. Za pomocą RKS otrzymano modularne rozwiązania dla problemu siatkówki, dla którego konkurencyjna metoda HyperNEAT okazała się wcześniej mało skuteczna (Clune et al., 2010).

Poważnym wyzwaniem było również wykazanie prawdziwości tezy 4. Choć o prawdziwości tej tezy można wnioskować przyjmując, że standardowe sieci neuronowe nie są optymalnym rozwiązaniem dla wielu problemów uczenia, dla których optymalne rozwiązanie istnieje. Wówczas z tezy 1 wynika, że metoda RKS pozwala znaleźć takie optymalne i lepsze zarazem rozwiązania. Niewątpliwie jednak wykazanie tej tezy doświadczalnie jest dużo bardziej wymowne. Lepsze wyniki sieci ewolucyjnych uzyskano w trzech przypadkach, w których porównywano je do standardowych sieci neuronowych. W problemie perceptron (§4.1.6) sieci ewolucyjne uczyły się istotnie skuteczniej aniżeli perceptron prosty dla instancji problemu z jednym wejściem. Podobnie w problemie spiral (§4.4.2) sieci ewolucyjne uczone na zasadzie wstecznej propagacji błędu uzyskały znacznie lepsze wyniki pod względem efektywności uczenia. W końcu, sieci ewolucyjne uzyskały lepsze wyniki w problemie klasyfikacji „ruchomy pas" (§4.6), opracowanym w celu zademonstrowania przewagi sieci ewolucyjnych i zarazem lepszego zobrazowania, dla jakiego rodzaju problemów taka przewaga może zaistnieć. Trudność tego problemu rozkłada się na dwie składowe — stałą oraz zmienną cechę środowiska. Jak to omówiono w §2.1, właśnie do tego rodzaju problemów ewolucyjne sieci obliczeniowe są szczególnie

odpowiednie, gdyż adaptacja ewolucyjna pozwala znaleźć sieci odpowiednio przystosowane do stałych cech środowiska, a jednocześnie zdolnych do uczenia się w warunkach zmiennych. Co więcej, mechanizmy uczenia również mogą podlegać optymalizacji na drodze ewolucyjnej.

W badaniach symulacyjnych wykazano również dwie inne ważne właściwości i możliwości metody RKS. Na przykładzie problemu perceptron (§4.4.1) oraz problemu spiral (§4.4.2) zademonstrowano możliwość wykorzystania wstecznej propagacji błędu do uczenia sieci. Do tego wymagane są jedynie drobne modyfikacje algorytmu działania sieci oraz wprowadzenie dodatkowych typów połączeń. Ponownie na przykładzie problemu perceptron (§4.5) zademonstrowano możliwość ewolucji sieci z połączeniami modulowanymi, czyli takimi, których funkcja ewaluacji lub uczenia podlega modulacji sygnałami z węzła trzeciego, modulującego. Neuromodulacja jest w obszarze SSN mechanizmem stosunkowo nowym, oferującym potencjalnie szerszy zakres zachowań sieci, szczególnie odnośnie uczenia.

W ostatnim eksperymencie badań symulacyjnych (§4.7) wykazano, że choć metoda RKS została opracowana z myślą o rozwiązywaniu problemów regularnych w sposób skalowalny, to zdolna jest również do rozwiązywania problemów nieregularnych. W §4.7.1 przedstawiono dwa rozwiązania dla standardowego problemu sztucznej mrówki na szlaku Santa Fe, wykorzystujące dwa różne rodzaje sieci. W §4.7.2 natomiast, przedstawiono rozwiązania dla problemu sterowania podwójnym odwróconym wahadłem, dla wariantów z pełną informacją o stanie systemu oraz niepełną. Wydajność metody RKS dla tego problemu nie była jednak tak dobra jak niektórych innych metod, lepiej przystosowanych do rozwiązywania niewielkich problemów nieregularnych.

Pewnym ograniczeniem pracy jest brak eksperymentalnego porównania metody RKS z metodami konkurencyjnymi, głównie *Cellular Encoding* oraz HyperNEAT. Jak to zostało uargumentowane na wstępie rozdziału 4, jednakże, takie porównanie byłoby trudne do przeprowadzenia, a jego wyniki niewiele mogłyby powiedzieć o tego rodzaju złożonych metodach. Niemniej, w badaniach występują odniesienia do innych metod. Oprócz wspomnianych już porównań ze standardowymi sieciami neuronowymi, w §4.1.4 przedstawiono skromne porównanie metody RKS z metodą HyperNEAT na problemie topologii docelowej, którego wyniki wskazały na przewagę metody RKS pod względem absolutnej skuteczności znajdowania rozwiązań oraz w mniejszym stopniu pod względem skalowalności. W porównaniu metod na problemie siatkówki (§4.1.3), wykorzystano dane pochodzące z eksperymentu przeprowadzonego w (Clune et al., 2010), i tu również metoda RKS okazała się nieco lepsza od HyperNEAT. W końcu, w §4.7.2 na problemie sterowania podwójnym wahadłem, efektywność metody RKS została zestawiona z efektywnością innych metod i — zgodnie z przewidywaniami — nie wypadła najlepiej.

Metoda RKS odznacza się znakomitymi właściwościami teoretycznymi (patrz §3.6) — krótko mówiąc zdolnością do wygenerowania optymalnego roz-

wiązania dla dowolnego problemu obliczalnego. Jest to jednak właściwość czysto teoretyczna, podczas gdy w praktyce skuteczność metody bywała różna, nawet przy ograniczeniu badań eksperymentalnych do problemów testowych. Próby rozwiązywania bardziej złożonych problemów rzeczywistych (które nie zostały przedstawione w pracy) z reguły napotykały na poważne trudności. Jedną z przyczyn jest niewątpliwie skromna moc obliczeniowa, jaka była wykorzystywana w ramach pracy (komputer osobisty), ale z pewnością są również inne. Dobór parametrów metody — a w zależności od ujęcia może ich być kilkadziesiąt — dokonywany był na podstawie prób i błędów. Tak więc jeden z przyszłych kierunków badań może dotyczyć możliwości wykorzystania samo-adaptacji parametrów w metodzie. Oprócz parametrów, wiele szczegółów algorytmu, zwłaszcza odnośnie sposobu kodowania, dekodowania oraz wariacji genetycznej sieci również opierało się na arbitralnych decyzjach, które również mogą być optymalizowane. Dalszych badań wymagałyby również bardziej ogólne techniki AE, takie jak regulacja różnorodności populacji, techniki regularyzacji czy wykorzystanie wielo-kryterialnego wariantu algorytmu.

Interesującym kierunkiem badań może być wykorzystanie metody w równoległym i przyrostowym rozwiązywaniu zadań. Podobnie jak złożone organizmy biologiczne ewoluowały z organizmów prostszych, również trudno oczekiwać, aby rozwiązania złożonych problemów wyewoluowały z pojedynczej populacji w ramach pojedynczego przebiegu ewolucyjnego, liczącego kilka tysięcy osobników i kilkaset pokoleń. Wydaje się, że rozwiązywanie problemów złożonych, wymaga ich rozłożenia na wiele prostszych pod-problemów, których rozwiązania ulegałyby złożeniom i rekombinacji, dając w efekcie coraz bardziej złożone rozwiązania. Raz wyewoluowane rozwiązania prostszych problemów mogłyby być składowane w bazach danych i służyć jako moduły w innych rozwiązaniach. Populacje i pokolenia musiałyby być liczone w milionach i miliardach osobników. Jest to wizja o tyle realna, że algorytmy ewolucyjne znakomicie poddają się zrównolegleniu i rozproszeniu. Interesującym przykładem mogą być tutaj algorytmy ewolucyjne wykorzystujące sieci *peer-2-peer* (Laredo et al., 2008). Tym samym, z praktycznego punktu widzenia, nie podlegają one tym ograniczeniom technologicznym co algorytmy typowo „sekwencyjne".

Wyniki pracy w punktach

- Nowatorska metoda ewolucji sieci obliczeniowych.
- Centralną częścią metody jest rozwojowa reprezentacja genetyczna sieci obliczeniowych.
- Metoda umożliwia syntezę sieci odznaczających się skalowalnością i modularnością.
- Metoda umożliwia syntezę sieci odznaczających się regularnością, a przez to pozwala na rozwiązywanie problemów w sposób skalowalny.

- Metoda umożliwia wygenerowanie rozwiązania dla dowolnego problemu obliczalnego w sensie Turing'a.
- Metoda umożliwia syntezę sieci uczących się co najmniej równie efektywnie jak sztuczne sieci neuronowe.
- Istnieją takie problemy, dla których metoda pozwala generować sieci uczące się efektywniej aniżeli standardowe sieci neuronowe.
- Metoda umożliwia synteze sieci o dowolnej topologii oraz dowolnych rodzajach połączeń i węzłów.
- Metoda umożliwia objęcie ewolucją również funkcji przejść i parametrów węzłów oraz funkcji przejść, uczenia i parametrów połączeń.
- Metoda pozwala narzucić dowolne ograniczenia i wymagania projektowe dające się wyrazić poprzez funkcję przystosowania.
- Metoda umożliwia wykorzystanie wstecznej propagacji błędu do uczenia sieci.
- Metoda jest konkurencyjna wobec uznanej i aktualnie rozwijanej metody HyperNEAT.

Słownik pojęć

Agent	Inaczej program, przy czym zaakcentowany jest tutaj kontekst środowiskowy i pewna autonomiczność w działaniu.
Ewolucja przyrostowa	Wariant AE, w którym populacje rozwiązują kolejno coraz trudniejsze instancje problemu.
Genom	Zbiór genów, materiał genetyczny osobnika.
Genotyp	Genetyczna, tj. zakodowana, postać osobnika (sieci).
Fenotyp	Rozwinięta, zdolna do wykonania zadania postać osobnika, czyli sieć w postaci grafu bądź pary $G = (\mathbf{V}, E)$.
Funkcja przystosowania	Funkcja stanowiąca optymalizowane kryterium jakości rozwiązania.
Kodowanie	Inaczej reprezentacja genetyczna.
Model wyspowy	Wariant AE, w którym wiele osobnych populacji ewoluuje równolegle, zwykle komunikując się ze sobą w ograniczonym zakresie (poprzez migracje).
Modularność	W ogólności: funkcjonalna modularność to strukturalna lokalizacja funkcji. W pracy modularność jest rozumiana jako występowanie powtarzających się w fenotypie podsieci (modułów), generowanych na podstawie jednego fragmentu kodu genetycznego, który jest wymienialny pomiędzy osobnikami.
Osobnik	Element populacji, (kandydat na) rozwiązanie, w postaci genotypowej i/lub fenotypowej. W kontekście pracy osobnikami są sieci.
Regularność problemu	Regularność problemu można rozumieć jako regularność jego rozwiązania, a więc w kontekście pracy wszystkich takich cech strukturalnych sieci, które mogą być zwięźle zakodowane (skompresowane), np. symetria, powtarzalność, powtarzalność z wariacją, rekurencja, samo-podobieństwo.

Skalowalność	W ogólności: możliwość dopasowania systemu do różnej skali zadania. W pracy skalowalność jest rozumiana bardziej szczególnie jako: zdolność do uchwycenia regularności problemu (fenotypu) w genotypie, i przez to efektywniejsze rozwiązywanie problemów o wzrastającym rozmiarze.
Przystosowanie	Wartość przystosowania, określająca jakość rozwiązania i brana pod uwagę podczas selekcji w AE.
Regularyzacja	W teorii AE zwana również zwięzłością (*parsimony*). Narzucenie ograniczeń odnośnie rozmiaru rozwiązania, najczęściej w postaci kary wprowadzonej do funkcji przystosowania, wzrastającej wraz z rozmiarem rozwiązania.
Reprezentacja genetyczna	Sposób zapisu osobnika (sieci) w postaci genotypowej.
Rozwój	Przekształcenie genotypu do postaci fenotypu.
Sieć neuronowa	Może odnosić się do biologicznej sieci neuronowej, lecz najczęściej odnosi się do sztucznej sieci neuronowej (SSN).
Sieć neuronowa, generyczna	Jak SSN, przy czym zaakcentowany jest tutaj sposób tworzenia sieci na podstawie pewnego szablonu, wzorca, konkretnego modelu.
Sieć neuronowa, standardowa	Jak SSN, przy czym zaakcentowana jest tutaj powszechność wykorzystania pewnych rodzajów sieci, jak np. wielowarstwowa sieć jednokierunkowa (perceptron wielowarstwowy).
Sieć neuronowa, sztuczna	Zbiór połączonych ze sobą sztucznych neuronów, zamodelowany w mniejszym lub większym stopniu na wzór biologicznych sieci neuronowych.
Sieć obliczeniowa	Jak SSN, przy czym odznacza się większą swobodą w określaniu różnych elementów architektury sieci, w sposób niekoniecznie zgodny z naturalnymi wzorcami.

Bibliografia

A. Abraham. Meta learning evolutionary artificial neural networks. *Neurocomputing*, 56:1–38, 2004.

A. Abraham, L. Jain, and R. Goldberg, eds. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Springer, 2005.

M. Augusteijn and T. Harrington. Evolving transfer functions for artificial neural networks. *Neural Computing and Applications*, 13(1):38–46, 2004.

T. Bäck, D. Fogel, and Z. Michalewicz, eds. *Handbook of Evolutionary Computation*. IOP Publishing, 1997.

K. Balakrishnan and V. Honavar. Evolving neuro-controllers and sensors for artificial agents. In M. Patel, V. Honavar, and K. Balakrishnan, eds., *Advances in the Evolutionary Synthesis of Intelligent Agents*, pp. 109–152. MIT Press, Cambridge, MA, 2001a.

K. Balakrishnan and V. Honavar. Evolutionary and neural synthesis of intelligent agents. In M. Patel, V. Honavar, and K. Balakrishnan, eds., *Advances in the Evolutionary Synthesis of Intelligent Agents*, pp. 1–27. MIT Press, Cambridge, MA, 2001b.

J. M. Baldwin. A new factor in evolution. *The American Naturalist*, 30(354): 441–451, 1896.

S. Bengio, Y. Bengio, and J. Cloutier. Use of genetic programming for the search of a new learning rule for neural networks. In *The 1st IEEE Conference on Evolutionary Computation*, vol. 1, pp. 324–327, Orlando, FL, 1994.

P. Bentley. *From coffee tables to hospitals: Generic evolutionary design*, pp. 405–423. Morgan Kaufmann, 1999.

E. J. W. Boers and I. G. Sprinkhuizen-Kuyper. Combined biological metaphors. In M. Patel, V. Honavar, and K. Balakrishnan, eds., *Advances in the Evolutionary Synthesis of Intelligent Agents*, pp. 153–183. MIT Press, Cambridge, MA, 2001.

J. A. Bullinaria. Understanding the emergence of modularity in neural systems. *Cognitive Science: A Multidisciplinary Journal*, 31(4):673–695, 2007.

D. J. Chalmers. The evolution of learning: An experiment in genetic connectionism. In D. S. Touretsky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, eds., *Proceedings of the 1990 Connectionist Models Summer School*, pp. 81–90, San Mateo, CA, 1990. Morgan Kaufmann.

Z. Chen, S. Haykin, J. Eggermont, and S. Becker. *Correlative Learning: A Basis for Brain and Adaptive Systems*. Wiley-Interscience, 2007.

S. Christensen and F. Oppacher. An analysis of Koza's computational effort statistic for Genetic Programming. In *EuroGP '02: Proceedings of the 5th European Conference on Genetic Programming*, pp. 182–191, London, UK, 2002. Springer-Verlag.

J. Clune, C. Ofria, and R. Pennock. How a generative encoding fares as problem-regularity decreases. *Parallel Problem Solving from Nature–PPSN X*, pp. 358–367, 2008.

J. Clune, B. Beckmann, C. Ofria, and R. Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *IEEE Congress on Evolutionary Computation 2009*, pp. 2764–2771. IEEE, 2009.

J. Clune, B. E. Beckmann, P. K. McKinley, and C. Ofria. Investigating whether HyperNEAT produces modular neural networks. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 635–642. ACM, 2010.

D. D'Ambrosio and K. Stanley. Generative encoding for multiagent learning. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pp. 819–826. ACM, 2008.

D. D'Ambrosio, J. Lehman, S. Risi, and K. Stanley. Evolving policy geometry for scalable multiagent learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems 1*, pp. 731–738. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

J. Drchal, J. Koutnik, and M. Snorek. HyperNEAT controlled robots learn how to drive on roads in simulated environment. In *IEEE Congress on Evolutionary Computation 2009*, pp. 1087–1092. IEEE, 2009.

W. Duch and N. Jankowski. Transfer functions: hidden possibilities for better neural networks. In *9th European Symposium on Artificial Neural Networks*, pp. 81–94, 2001.

D. Floreano and C. Mattiussi. *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press, 2008.

D. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Wiley-IEEE Press, 2006.

J. Gauci and K. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural computation*, 22(7):1860–1898, 2010.

J. Gauci and K. O. Stanley. Generating large-scale neural networks through discovering geometric regularities. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 997–1004, New York, NY, USA, 2007. ACM.

F. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *International Joint Conference on Artificial Intelligence*, vol. 16, pp. 1356–1361, 1999.

F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis, Ecole Normale Supérieure de Lyon, France, 1994a.

F. Gruau. Automatic definition of modular neural networks. *Adaptive Behavior*, 3(2):151–183, 1994b.

F. Gruau and K. Quatramaran. Cellular Encoding for interactive evolutionary robotics. In *Fourth European Conference on Artificial Life*, pp. 368–377. The MIT Press, 1997.

S. Harding and W. Banzhaf. Artificial development. *Organic Computing*, pp. 201–219, 2008.

S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd ed., 1998.

G. S. Hornby. Functional scalability through generative representations: the evolution of table designs. *Environment and Planning B*, 31(4):569–588, 2004.

G. S. Hornby. Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pp. 1729–1736. ACM, 2005.

J. Jung and J. Reggia. Evolutionary design of neural network architectures using a descriptive encoding language. *IEEE Transactions on Evolutionary Computation*, 10(6):676–688, 2006.

N. Kashtan and U. Alon. Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences of the United States of America*, 102(39):13773, 2005.

H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4(4):461–476, 1990.

J. Koza. Human-competitive results produced by Genetic Programming. *Genetic Programming and Evolvable Machines*, pp. 1–34, 2010.

J. Koza, M. Keane, and M. Streeter. What's AI done for me lately? Genetic Programming's human-competitive results. *IEEE Intelligent Systems*, pp. 25–31, 2003.

J. R. Koza. *Genetic Programming.* MIT Press, 1992.

H. Kwaśnicka. *Ewolucyjne projektowanie sieci neuronowych.* Oficyna Wydawnicza Politechniki Wrocławskiej, 2007.

J. Laredo, A. Eiben, M. van Steen, P. Castillo, A. Mora, and J. Merelo. P2P evolutionary algorithms: A suitable approach for tackling large instances in hard optimization problems. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, pp. 622–631. Springer, 2008.

T. Lewis. *Network Science: Theory and Applications.* Wiley Publishing, 2009.

H. Lipson. Principles of modularity, regularity, and hierarchy for scalable systems. *Journal of Biological Physics and Chemistry*, 7(4):125, 2007.

M. L. Littman. *Simulations combining evolution and learning*, pp. 465–477. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

C. Mattiussi. *Evolutionary Synthesis of Analog Networks.* PhD thesis, Ecole Polytechnique Fédérale de Lausanne, Italie, 2005.

C. Mattiussi and D. Floreano. Analog Genetic Encoding for the evolution of circuits and networks. *IEEE Transactions on Evolutionary Computation*, 11 (5):596–607, 2007.

M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6):066133, 2004.

M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):26113, 2004.

S. Nolfi and D. Floreano. Learning and evolution. *Autonomous robots*, 7(1): 89–113, 1999.

R. Poli, W. Langdon, and N. McPhee. *A Field Guide to Genetic Programming.* Lulu Press, 2008.

A. Radi and R. Poli. Discovering efficient learning rules for feedforward neural networks using Genetic Programming. In *Recent advances in intelligent paradigms and applications*, pp. 133–159. Physica-Verlag, Heidelberg, Germany, 2003.

S. Risi and K. Stanley. Indirectly encoding neural plasticity as a pattern of local rules. In *From Animals to Animats 11*, Lecture Notes in Computer Science, pp. 533–543. Springer Berlin / Heidelberg, 2010.

S. Risi, J. Lehman, and K. Stanley. Evolving the placement and density of neurons in the HyperNEAT substrate. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 563–570. ACM, 2010.

M. Rocha, P. Cortez, and J. Neves. Evolution of neural networks for classification and regression. *Neurocomputing*, 70(16-18):2809–2816, 2007.

F. Rothlauf. *Representations for genetic and evolutionary algorithms.* Springer-Verlag, 2nd ed., 2006.

J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik, and K. O. Stanley. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation*, pp. (to appear), 2011.

A. Soltoggio, P. Dürr, C. Mattiussi, and D. Floreano. Evolving neuromodulatory topologies for reinforcement learning-like problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2471–2478, 2007.

A. Soltoggio, J. Bullinaria, C. Mattiussi, P. Dürr, and D. Floreano. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In S. Bullock, J. Noble, R. Watson, and M. A. Bedau, eds., *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, pp. 569–576. MIT Press, Cambridge, MA, 2008.

K. Stanley. Exploiting regularity without development. In *Proceedings of the AAAI Fall Symposium on Developmental Systems*, 2006.

K. Stanley. Compositional Pattern Producing Networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, 2007.

K. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.

K. Stanley, D. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

K. O. Stanley. Generative and developmental systems. In *GECCO '08: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pp. 2849–2864, New York, NY, USA, 2008. ACM.

K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9:653–668, 2005.

M. Suchorzewski. NeuroEvolution of Augmenting Topologies under Linear Genetic Programming framework. *Polish Journal of Environmental Studies*, 17(3b):154–157, 2008a.

M. Suchorzewski. Evolving weighted topologies for neural networks using Genetic Programming. *Metody Informatyki Stosowanej*, 17(4):211–221, 2008b.

M. Suchorzewski. Extending Genetic Programming to evolve perceptron-like learning programs. In L. Rutkowski et al., eds., *ICAISC 2010, Part II*, LNAI 6114, pp. 221–228, Heidelberg, 2010a. Springer.

M. Suchorzewski. Evolving adaptive recurrent networks with Developmental Symbolic Encoding. *Metody Informatyki Stosowanej*, 22(1):175–187, 2010b.

A. Teller. Turing completeness in the language of Genetic Programming with indexed memory. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, vol. 1, pp. 136–141, 1994.

D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing*, 14 (3):347–361, 1990.

D. Wolpert and W. Macready. No free lunch theorems for search. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

J. Woodward and R. Bai. Why evolution is not a good paradigm for program induction: a critique of genetic programming. *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pp. 593–600, 2009.

X. Yao. Evolving Artificial Neural Networks. *Proceedings of the IEEE*, 87(9): 1423–1447, 1999.