Wydział Informatyki i Telekomunikacji
Katedra Automatyki, Mechatroniki i Systemów Sterowania
Politechnika Wrocławska

Seria: PRE nr W04N/2024/P-001

Uczenie maszynowe i analiza krajobrazu przestrzeni w rozwiązywaniu wybranych problemów szeregowania zadań i marszrutyzacji

(rozprawa doktorska)

Teodor Niżyński

PROMOTOR:
Prof. dr hab. Wojciech Bożejko

PROMOTOR POMOCNICZY:
Dr inż. Andrzej Gnatowski

Słowa kluczowe:
– optymalizacja dyskretna
– uczenie maszynowe
– analiza krajobrazu przestrzeni rozwiązań
– szeregowanie zadań
– marszrutyzacja
– sztuczne sieci neuronowe
– uczenie ze wzmocnieniem
– sieć lokalnych optimów

WROCŁAW, luty 2024

Autor dysertacji składa serdeczne podziękowania:

Rodzicom: Monice i Zbigniewowi Niżyńskim
– za nieustające wsparcie, wpojenie wartości nauki oraz pomoc w znajdowaniu drogi rozwoju od lat najmłodszych. Dodatkowo za zapewnianie warunków, które umożliwiły studiowanie, w tym również rozpoczęcie studiów doktorskich.

Prof. dr. hab. Wojciechowi Bożejce
– za pomoc na przestrzeni całego doktoratu – ze szczególnym pokreśleniem znaczenia w ukończeniu pracy oraz anielską cierpliwość do doktoranta, który w swojej ocenie „porwał się z motyką na słońce", co gorsza jednocześnie dużą część czasu „marnując" na pracę komercyjną poza uczelnią.

Dr. inż. Andrzejowi Gnatowskiemu
– za długoletnią przyjaźń, a także bycie niedoścignionym wzorem młodego, utalentowanego naukowca – istnego „tytana pracy" – oraz pomoc wszelaką m.in. wspólne publikacje czy liczne rozmowy stanowiące wyzwanie intelektualne.

Bliskiemu środowisku naukowemu i biznesowemu, w tym:
MEng MIChemE Lucy Kennedy,
Mgr inż. Arturowi Barcikowskiemu,
Dr. inż. Radosławowi Idzikowskiemu,
Mgr inż. Maciejowi Kołczykowi,
Mgr inż. Sewerynowi Zawadzkiemu,
Maciejowi Błędowskiemu,
Mgr inż. Michalinie Kotyli,
Dr. inż. Konradowi Kluwakowi,
– za udowadnianie swoim postępowaniem, mimo licznych przeciwności dni powszednich, że „da się"[1]*, nawet w warunkach, które to aktywnie utrudniają – i co gorsza, prawdopodobnie coraz dalej od „końca historii" Fukuyamy, a bliżej przysłowiowych „ciekawych czasów".*

Spis treści

Lista skrótów

ABC	–	Algorytm kolonii pszczół, z ang. *Artificial Bee Colony*
AGI	–	Silna sztuczna inteligencja, z ang. *Artificial General Intelligence*
AI	–	Sztuczna inteligencja, z ang. *Artificial Intelligence*
ASP	–	Problem wyboru algorytmu, z ang. *Algorithm Selection Problem*
B&B	–	Schemat podziału i ograniczeń, z ang. *Branch and Bound*
BF	–	Podejście siłowe, z ang. *Brute Force*
BP	–	Algorytm propagacji wstecznej, z ang. *Backpropagation*
CJSSP	–	Cykliczny problem gniazdowy, z ang. *Cyclic Jobs Shop Scheduling Problem*
CVRP	–	Problem marszrutyzacji z ograniczeniem pojemności, z ang. *Capacitated Vehicle Routing Problem*
DL	–	Uczenie głębokie, z ang. *Deep Learning*
DP	–	Programowanie dynamiczne, z ang. *Dynamic Programming*
DQN	–	Model uczenia ze wzmocnieniem wykorzystujący głęboką sieć neuronową *Deep Q-Network*
EA	–	Algorytm ewolucyjny, z ang. *Evolution Algorithm*
FL	–	Przestrzeń rozwiązań, z ang. *Fitness Landscape*
FLA	–	Analiza przestrzeni rozwiązań, z ang. *Fitness Landscape Analysis*
FNN	–	Jednokierunkowe sieci neuronowe, z ang. *Feed-forward Neural Network*
FSSP	–	Niepermutacyjny problem przepływowy szeregowania zadań, z ang. *Non-permutation Flow Shop Scheduling Problem*
GA	–	Algorytm genetyczny, z ang. *Genetic Algorithm*
GAN	–	Generatywna Sieć Konkurencyjna, z ang. *Generative Adversarial Network*
JSSP	–	Problem gniazdowy szeregowania zadań, z ang. *Job Shop Scheduling Problem*

LB – Dolne ograniczenie, z ang. *Lower Bound*
LLM – Ogromne modele językowe, z ang. *Large Language Models*
LON – Sieć lokalnych optimów, z ang. *Local Optima Network*
LP – Programowanie liniowe, z ang. *Linear Programming*
LSTM – Architektura sieci neuronowej LSTM od nazwy angielskiej *Long Short-Term Memory*
MDP – Procesy decyzyjne Markowa, z ang. *Markov Decision Processes*
MILP – Programowanie liniowe mieszane całkowitoliczbowe, z ang. *Mixed Integer Linear Programming*
ML – Uczenie maszynowe, z ang. *Machine Learning*
PFSSP – Permutacyjny problem przepływowy szeregowania zadań, z ang. *Permutation Flow Shop Scheduling Problem*
PMSP – Problem szeregowania zadań z równoległymi maszynami, z ang. *Parallel Machine Scheduling Problem*
PN – Model głębokiej sieci neuronowej *Pointer Network*
PSO – Algorytm optymalizacji za pomocą roju cząstek, z ang. *Particle Swarm Optimization*
RL – Uczenie ze wzmocnieniem, z ang. *Reinforcement Learning*
RNN – Rekurencyjne sieci neuronowe, z ang. *Recurrent Neural Network*
SA – Algorytm symulowanego wyżarzania, z ang. *Simulated Annealing*
SL – Uczenie nadzorowane, z ang. *Supervised Learning*
T2IG – Metody generowania obrazów na podstawie dostarczonego opisującego je tekstu, z ang. *Text-to-Image Generation*
TS – Algorytm przeszukiwania z zabronieniami, z ang. *Tabu Search*
TSP – Problem komiwojażera, z ang. *Traveling Salesman Problem*
UB – Górne ograniczenie, z ang. *Upper Bound*
UL – Uczenie nienadzorowane, z ang. *Unsupervised Learning*

Streszczenie

Tematyką niniejszej rozprawy jest uczenie maszynowe i analiza krajobrazu przestrzeni rozwiązań, zastosowane do rozwiązywania wybranych problemów szeregowania zadań i marszrutyzacji. W ramach dysertacji przeprowadzono analizę różnorodnych problemów z zakresu badań operacyjnych, informatyki oraz automatyki – od klasycznego problemu komiwojażera, poprzez cykliczny problem gniazdowy, aż po cykliczny problem przydziału w niepermutacyjnym problemie przepływowym szeregowania zadań. Praca zawiera także przegląd teoretyczny stosowanych metod, obejmujący: uczenie maszynowe, z naciskiem na sztuczne sieci neuronowe, analizę krajobrazu przestrzeni rozwiązań, sieci lokalnych optimów, uczenie głębokie oraz uczenie ze wzmocnieniem. Realizacja badań własnych wymagała licznych eksperymentów obliczeniowych. Dokonano modyfikacji metaheurystyk – klasycznych metod optymalizacji dyskretnej – poprzez użycie mechanizmów neuronowych, a także zastosowanie podejścia opartego o uczenie ze wzmocnieniem. Wykorzystano zaawansowane narzędzia analizy krajobrazu przestrzeni rozwiązań w celu ekstrakcji informacji o cechach instancji rozważanych problemów optymalizacji dyskretnej, co następnie pozwoliło na rozwiązywanie problemu wyboru algorytmu. Aby ocenić przydatność wykorzystania analizy krajobrazu przestrzeni rozwiązań, przeprowadzono badania nad zależnością wybranych metryk od stopnia próbkowania przestrzeni rozwiązań, z uwzględnieniem kosztu obliczeniowego ich uzyskania. Otrzymane rezultaty konstytuują proces zmierzający do integracji metod uczenia maszynowego i analizy krajobrazu przestrzeni rozwiązań, w kontekście tworzenia zaawansowanego systemu dedykowanego efektywnemu rozwiązywaniu wybranych problemów szeregowania zadań i marszrutyzacji.

Abstract

The subject of this dissertation is Machine Learning and Fitness Landscape Analysis, applied to solving selected Job Scheduling and Routing Problems. The dissertation includes an analysis of various problems in the field of Operations Research, Computer Science and Automation – from the classic Travelling Salesman Problem, through the Cyclic Job Shop Problem, to the Cyclic Assignment Problem in Non-permutational Flow Shop Scheduling Problem. The work also contains a theoretical overview of the methods used, covering: Machine Learning, with an emphasis on Artificial Neural Networks, Fitness Landscape Analysis, Local Optima Networks, Deep Learning, and Reinforcement Learning. The realization of the author's own research required numerous computational experiments. Modifications were made to metaheuristics – classical Discrete Optimization methods – through the use of neural mechanisms, as well as the application of an approach based on Reinforcement Learning. Advanced Fitness Landscape Analysis tools were utilized to extract information about the features of the instances of the considered Discrete Optimization Problems, which then allowed solving the Algorithm Selection Problem. To assess the usefulness of Fitness Landscape Analysis, research was conducted on the relationship between selected metrics and the sampling rate of the solution space, with regard to the consideration of computational cost of obtaining them. The obtained results constitute a process aiming at the integration of Machine Learning methods and Fitness Landscape Analysis, in the context of creating an advanced system dedicated to effectively solving selected Job Scheduling and Routing Problems.

Rozdział 1

Wprowadzenie

Celem niniejszego rozdziału jest wstępne przedstawienie zakresu badań poruszanych w ramach rozprawy doktorskiej, teorii na której te badania zostały oparte oraz motywacji, która przyświecała powstawaniu dysertacji. W rozdziale omówiono również tezy pracy oraz ogólną strukturę rozprawy.

1.1 Wstęp

Świat dysponuje ograniczonymi zasobami, zarówno w kontekście dóbr konsumpcyjnych, jak i nawet materiałów oraz czasu (choćby w postaci ograniczonych dopuszczalnych kosztów) niezbędnego do ich wytworzenia. Dlatego naturalnie nasuwa się potrzeba rozsądnego, a w idealnym przypadku optymalnego, zarządzania tymi zasobami oraz powiązanymi z nimi procesami. Skalę makro tej optymalizacji można przypisać do domeny nauk ekonomicznych (szczególnie w ramach pojęć: rzadkości, niedoboru oraz nieograniczonych potrzeb), natomiast optymalizacja w skali mikro leży w zakresie informatyki, matematyki, automatyki i pokrewnych dziedzin nauki. Udoskonalenie nawet pojedynczego modelu matematycznego, opisującego na przykład pracę linii produkcyjnej czy proces marszrutyzacji dostaw, może mieć realny, pozytywny wpływ na otaczającą nas rzeczywistość.

Optymalizacja nie tylko umożliwia osiąganie wyższych zysków, co za pośrednictwem reinwestycji prowadzi do zwiększania skali produkcji, ale także zwiększa dostępność dóbr konsumpcyjnych, na przykład poprzez efektywniejszy transport do konsumentów. Ważna może być też jej rola w redukcji odpadów produkcyjnych, co nabiera szczególnego znaczenia w dobie rosnącej świadomości ekologicznej. Dzięki temu możliwe jest ograniczenie zanieczyszczeń przy jednoczesnym zachowaniu (lub jedynie minimalnej utracie) jakości i poziomu produkcji. Optymalizacja, niezaprzeczalnie potrzebna

i kluczowa, może być również postrzegana jako moralnie słuszna, choć może to wydawać się nieoczywiste. Jest ona jednak niezaprzeczalnie istotnym elementem produktywności, będącym wyrazem właściwego wykorzystania własnego rozumu, co stanowi cnotę w ramach filozofii obiektywizmu (zob. Ayn Rand [216, 215]).

Chociaż istnieje wiele sposobów na poprawę systemów produkcyjnych w ramach procesu optymalizacji, niniejsza rozprawa koncentruje się, zgodnie ze swoim tytułem, na wykorzystaniu uczenia maszynowego – ze szczególnym uwzględnieniem sztucznych sieci neuronowych – oraz metod analizy krajobrazu przestrzeni rozwiązań do rozwiązywania problemów optymalizacji dyskretnej, w tym problemów szeregowania zadań i marszrutyzacji. Ze względu na połączenie trzech tak obszernych dziedzin, praca koncentruje się jedynie na wybranych problemach – od klasycznego problemu komiwojażera, poprzez cykliczny problem gniazdowy, aż cykliczny problem przydziału w niepermutacyjnym problemie przepływowym szeregowania zadań.

Przedmiotem badań była efektywność rzadziej eksplorowanego podejścia hybrydowego, łączącego klasyczne metody rozwiązywania problemów dyskretnych z sieciami neuronowymi, a także z wykorzystaniem analizy krajobrazu przestrzeni rozwiązań. Wybrano te dwie dziedziny ze względu na ich potencjalną synergię. Analiza krajobrazu przestrzeni umożliwia zbieranie danych o instancjach problemów optymalizacji dyskretnej, reprezentowanych przez numeryczne metryki. To właśnie metryki mogą następnie wzbogacać dane wykorzystywane przy trenowaniu sieci neuronowych, dla których jakość wyników wprost zależy od ilości, jakości i formy reprezentacji danych. Ponadto, zgodnie z literaturą, sieci neuronowe mają potencjał do pozyskania nowych, nieoczywistych zależności między danymi. Dodatkowo sieci neuronowe mogą służyć jako modyfikacja lub uzupełnienie klasycznych metod rozwiązywania problemów optymalizacji dyskretnej.

W ramach dysertacji eksplorowano to zagadnienie poprzez modyfikację wybranych klasycznych metaheurystyk, takich jak przeszukiwanie z zabronieniami i symulowane wyżarzanie, przy użyciu metod opartych o sztuczne sieci neuronowe, jak i za pomocą metod neuronowych. W kontekście analizy przestrzeni rozwiązań, badania skupiły się na ocenie przydatności metody sieci lokalnych optimów do rozwiązywania problemu wyboru algorytmu. Jest to problem praktyczny, często występujący gdy dla zadanego problemu optymalizacji, dostępne jest zróżnicowane portfolio algorytmów służących do jego rozwiązywania. Ponieważ każdy z algorytmów jest inny, cechuje się różną efektywnością rozwiązywania problemu (lub poszczególnych jego instancji). Niestety wskazanie „najlepszego" algorytmu bez bezpośredniego porównania rzeczywistych wyników wszystkich dostępnych metod pozo-

staje zazwyczaj zadaniem trudnym. Wykorzystanie metod FLA może być obiecującym podejściem, ponieważ pozwala ono charakteryzować specyfikę instancji problemów bez konieczności podejmowania próby ich rozwiązywania. Rozwiązywanie problemu wyboru algorytmu sprowadza się wtedy do zadania klasyfikacji, przy wykorzystaniu historii działania algorytmów należących do portfolio na zróżnicowanych instancjach o znanych cechach.

Dodatkowo rozważano możliwość połączenia narzędzi bazujących na sztucznych sieciach neuronowych z metodami analizy przestrzeni rozwiązań (FLA, z ang. *Fitness Landscape Analysis*), w celu stworzenia bardziej zaawansowanego systemu optymalizacji. Taki system miałby na celu polepszenie jakości otrzymanych wyników, szczególnie dla instancji problemów o znaczących rozmiarach, gdzie przegląd pełnej reprezentacja przestrzeni, wymagająca enumeracji wszystkich możliwych rozwiązań, jest w praktyce niemożliwy. Niezbędne jest więc przeprowadzenie ograniczonego próbkowania przestrzeni rozwiązań – co stanowiło podstawę do badań własnych, gdzie analizowano wpływ stopnia próbkowania przestrzeni rozwiązań na dokładność oszacowania wartości metryk sieci lokalnych optimów, będących jedną z technik FLA.

1.2 Tezy pracy

Za właściwy cel pracy przyjęto wykazanie następujących tez:

- korzystając ze sztucznych sieci neuronowych lub analizy krajobrazu przestrzeni rozwiązań możliwe jest poprawienie efektywności działania konwencjonalnych algorytmów metaheurystycznych, rozumiane jako uzyskiwanie w tym samym koszcie (np. czasowym) rozwiązań o wyższej jakości;
- sposób przeprowadzenia próbkowania przestrzeni rozwiązań może mieć istotny wpływ na wartości estymacji miar tej przestrzeni;
- korzystając ze sztucznych sieci neuronowych oraz analizy krajobrazu przestrzeni rozwiązań możliwe jest rozwiązywanie problemu wyboru algorytmu;
- możliwe jest zastępowanie elementów klasycznych algorytmów metaheurystycznych przez zaawansowane metody oparte o sieci neuronowe w ramach uczenia ze wzmocnieniem.

Wyrażone wyżej tezy będą wykazywane w pracy w trakcie procesu badawczego, obejmującego między innymi:

- modyfikację istniejących algorytmów metaheurystycznych z wykorzystaniem metod opartych o sieci neuronowe, w tym uczenia ze wzmoc-

nieniem oraz mechanizmów neuronowych do rozwiązywania problemów optymalizacji dyskretnej;
- opracowanie metody rozwiązywania problemu wyboru algorytmu dla klasycznego problemu komiwojażera, wykorzystującą analizę przestrzeni rozwiązań i uczenie maszynowe;
- analizę wpływu stopnia próbkowania przestrzeni rozwiązań na wartości numeryczne metryk analizy przestrzeni rozwiązań;
- przeprowadzenie eksperymentów obliczeniowych na instancjach losowych oraz na popularnych, uznanych zbiorach danych.

Przeprowadzone badania mają charakter interdyscyplinarny. Obejmują w swoim zakresie: badania operacyjnie, metody sztucznej inteligencji, analizę przestrzeni rozwiązań, uczenie ze wzmocnieniem, projektowanie i analizę algorytmów przybliżonych, metody uczenia maszynowego, teorię złożoności obliczeniowej, teorię szeregowania zadań, praktykę optymalizacji procesów przemysłowych i transportowych.

1.3 Zakres pracy

Z wyłączeniem rozdziału wprowadzającego, dysertacja została podzielona na dwie części: I) stan wiedzy oraz II) opis przeprowadzonych badań. Część pierwsza rozpoczyna się od rozdziału drugiego, w którym zaprezentowano wybrane problemy optymalizacji dyskretnej, takie jak: problem komiwojażera, problem przepływowy, problem gniazdowy czy cykliczny problem szeregowania i przydziału operacji w wielostanowiskowym gnieździe produkcyjnym. Wybrano je ze względu na ich zastosowanie w dalszych częściach dysertacji. Rozdział trzeci zawiera usystematyzowany opis klasycznych metod optymalizacji dyskretnej, z podziałem na metody dokładne oraz metody przybliżone – w tym algorytmy konstrukcyjne, algorytmy przeszukiwania lokalnego oraz algorytmy populacyjne. Ponownie nie starano się o wyczerpujący, „encyklopedyczny przegląd", a zamiast tego poświęcono uwagę metodom wykorzystywanym w kolejnych rozdziałach. W rozdziale czwartym przedstawiono tematykę sieci neuronowych, jak również powiązane pojęcia, w tym rodzaje uczenia maszynowego. Szczegółowo omówiono różnice między klasycznym podejściem do sieci neuronowych, a uczeniem głębokim oraz uczeniem ze wzmocnieniem. Pogłębiono przy tym zagadnienia związane z samym procesem trenowania sieci neuronowych – np. podziału danych na zbiór treningowy, walidacyjny oraz testowy. Rozdział piąty skupia się na analizie krajobrazu przestrzeni rozwiązań, w tym często wykorzystywanych cechach przestrzeni rozwiązań – m.in.: szorstkość, epistazę czy

neutralność. Omówiono również podejście oparte o metodę sieci lokalnych optimów, zastosowaną w badaniach własnych.

Część druga rozprawy poświęcona jest badaniom własnym i rozpoczyna się od rozdziału szóstego. Opisano w nim eksperymenty dotyczące rozwiązywania problemu wyboru algorytmu w kontekście problemu komiwojażera. Informacje koniecznie do podjęcia właściwej decyzji wyboru algorytmu stanowiły rezultat wykorzystania metody analizy przestrzeni rozwiązań – sieci lokalnych optimów. Rozdział siódmy zawiera badania związane z zaobserwowaną zmiennością numeryczną miar sieci lokalnych optimów, zależną od stopnia próbkowania przestrzeni. Uzyskanie „stabilnych" (nawet jeśli kosztownych obliczeniowo) miar LON pozwoliłoby podjąć świadomą decyzję o wykorzystaniu tej metody w ramach rozbudowanego systemu optymalizacji. W rozdziale ósmym opisano modyfikację wybranego mechanizmu składowego klasycznej metaheurystyki przeszukiwania z zabronieniami za pomocą neuronowego mechanizmu pamięci. Modyfikacja ta miała na celu potwierdzenie możliwości usprawnienia klasycznych metod optymalizacji, co następnie daje przesłanki za zasadnością wykorzystania bardziej zaawansowanych metod uczenia maszynowego, jako dalszych modyfikacji. Dziewiąty rozdział prezentuje badania własne, związane z wykorzystaniem sieci neuronowych w ramach podejścia opartego o uczenie ze wzmocnieniem. Przedstawiono w nim również wizję stworzenia bardziej zaawansowanego systemu optymalizacji dyskretnej, wraz ze wstępnymi wynikami, dla modyfikacji schematu chłodzenia algorytmu symulowanego wyżarzania. Rozprawę zamyka podsumowanie, po którym następują bibliografia oraz spisy tabel, rysunków i algorytmów.

Część I

Stan wiedzy

Rozdział 2

Opis wybranych problemów optymalizacji dyskretnej

W rozdziale zostaną przedstawione wybrane problemy optymalizacji dyskretnej. Ze względu na specyfikę pracy świadomie zdecydowano się nie przedstawiać obszernej klasyfikacji licznej rodziny problemów (tego typu materiałowi należałoby poświęcić całą, osobną książkę). Skupiono się więc zamiast tego na opisaniu problemów, których rozwiązywaniu poświęcone są kolejne rozdziały dysertacji, wraz z opisem problemów „bazowych", które umożliwiają łatwiejsze zrozumienie tych pierwszych.

2.1 Problemy optymalizacji dyskretnej

Problem optymalizacji dyskretnej jest problemem maksymalizacji (lub minimalizacji) rzeczywistej funkcji celu zależnej od skończonego całkowitoliczbowego zbioru rozwiązań X. Głównym wyzwaniem jest opracowanie algorytmów pozwalających na znalezienie dostatecznie dobrego rozwiązania (najlepiej optymalnego) dla funkcji celu, bez konieczności dokonania przeglądu zupełnego, czyli sprawdzenia wartości funkcji celu dla każdego możliwego rozwiązania. Podejście to jest konieczne w rzeczywistych zastosowaniach, gdzie moc obliczeniowa jest ograniczona.

Najbardziej ogólną wersję problemu optymalizacji dyskretnej (minimalizacji) można przedstawić jako

$$\min_{x \in X} f(x), \tag{2.1}$$

gdzie f to funkcja celu, x to rozwiązanie, a X to zbiór rozwiązań. W praktyce wiele z rozwiązań należących do X jest niedopuszczalnych (tzw. roz-

wiązania *niedopuszczalne*). Fakt ten może być odzwierciedlony bezpośrednio w definicji funkcji celu, np. przez przypisanie $f(x) \stackrel{\text{def}}{=} \infty$ gdy x jest rozwiązaniem niedopuszczalnym. Innym podejściem może być opisanie warunków dopuszczalności wyrażeniem logicznym. Często spotykaną formą zapisu problemów wykorzystującą takie podejście jest

$$\min_{x \in X} f(x) \tag{2.2}$$

$$\text{s.t. } g(x) \leqslant 0, \tag{2.3}$$

gdzie g jest funkcją definiującą warunki dopuszczalności. Zapis ten należy rozumieć tak, że rozważane są wyłącznie takie x dla których wyrażenie (2.3) jest prawdziwe (warunek jest spełniony).

Ze względu na ogólność i ekspresywność modeli, optymalizacja dyskretna znajduje bardzo szerokie zastosowanie. Cytując za [161], optymalizacja dyskretna jest wykorzystywana w dziedzinach matematyki takich jak: algebra, geometria, logika, analiza numeryczna, topologia, teoria grafów, teoria matroidów, czy kombinatoryka enumeratywna. Z kolei w dziedzinach pokrewnych, jest ona stosowana między innymi w: informatyce, statystyce, fizyce i chemii. Tym samym optymalizacja dyskretna jest zagadnieniem rozważanym zarówno z perspektywy teorii matematycznej, jak i możliwości bezpośredniego zastosowania. Do tej drugiej kategorii można zaliczyć aktywności takie jak: przemysłowe procesy wytwarzania, logistykę, telekomunikację (szczególnie proces projektowania sieci i trasowania) oraz szeroko rozumiane szeregowanie zadań.

2.2 Problem komiwojażera

W problemie komiwojażera (ang. *Traveling Salesman Problem*, TSP) dany jest zbiór miast $\mathcal{M} = \{1, 2, \ldots, m\}$, wraz z kosztami podróży pomiędzy każdą z par tych miast. Zadaniem jest znalezienie kolejności odwiedzenia wszystkich miast dokładnie raz i powrotu do miasta początkowego, o jak najmniejszym koszcie. Rozwiązanie definiowane jest przez kolejność miast i nazywane jest również trasą (ang. *tour*) lub objazdem pomiędzy miastami (ang. *circuit through the cities*). Kolejność ta może zostać zapisana jako krotka $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ dla $\pi \in \Pi$, przy czym zbiór wszystkich Π jest formalnie opisany przez

$$\Pi = \{\pi \in \mathbb{Z}^n \mid \forall i \in \mathcal{M}\ \exists! j \in \mathcal{M}\ \pi(j) = i\}. \tag{2.4}$$

Taka reprezentacja rozwiązania jest nadmiarowa, t.j. dowolna kolejność odwiedzanych miast jest reprezentowana przez m różnych π (tożsame kolejności mają taką samą wartość funkcji celu). Bez utraty ogólności, jedno

z miast można wyłączyć z π – co odpowiada obraniu określonego miasta jako punktu startowego, oraz rozpatrywania $\pi = (\pi(1), \pi(2), \ldots, \pi(n-1))$. Odległości pomiędzy miastami można przedstawić w postaci funkcji dwuargumentowej d (często też zastępowanej macierzą). Wtedy funkcja celu f przyjmuje postać

$$f(\pi) = d_{1,\pi(1)} + d_{\pi(n-1),1} + \sum_{i=1}^{n-2} d_{\pi(i),\pi(i+1)}, \tag{2.5}$$

gdzie $d_{i,j}$ oznacza odległość pomiędzy miastami i i j.

Problem TSP należy do najintensywniej badanych problemów matematyki obliczeniowej (ang. *computational mathematics*). Jest również często wykorzystywany jako doskonały problem początkowy przy omawianiu matematyki dyskretnej. Problem komiwojażera znajduje również dużo zastosowań praktycznych, miedzy innymi w: logistyce, genetyce, produkcji, telekomunikacji czy neurobiologii.

Zgodnie z źródłem [6], TSP jest przykładem problemu o prostej definicji, zrozumiałej również dla osób spoza dziedziny, a jednocześnie na tyle trudnego, by nie dało się wskazać jednoznacznie najlepszego algorytmu czy metody jego rozwiązywania. Sprawia to, że problem nadaje się do rozwijania i testowania nowych pomysłów, potencjalnie możliwych do zastosowania również dla innych problemów obliczeniowych – przez co jest problemem ciągle rozważany w literaturze.

TSPLIB [222] jest szczególnie popularnym zbiorem przykładowych instancji dla problemu komiwojażera wraz z jego odmianami, takimi jak:

- Symetryczny problem komiwojażera (ang. *Symmetric Traveling Salesman Problem*) – dystans pomiędzy parą miast jest taki sam niezależnie od kierunku, tj. dystans z miasta (węzła) i do miasta j jest taki sam jak dystans z miasta j do miasta i.
- Asymetryczny problem komiwojażera (ang. *Asymmetric Traveling Salesman Problem*) – instancje problemu zawierają różne dystanse dla pary miast w zależności od kierunku, na przykład spowodowane obecnością dróg jednokierunkowych lub znaczących różnic wysokości.
- Problem porządkowania sekwencyjnego (ang. *Sequential Ordering Problem*) – odmiana asymetrycznego problemu komiwojażera z dodatkowymi ograniczeniami wymuszającymi pierwszeństwo odwiedzenia określonych miast. Ograniczenie jest przedstawione jako wymóg odwiedzenia pewnego miasta j, zanim miasto i będzie możliwe do odwiedzenia. Rozważane są tylko kolejności odwiedzania miast rozpoczynające z miasta pierwszego.

- Określenie czy dany graf jest grafem hamiltonowskim (ang. *Hamiltonian Cycle Problem*) – czyli czy istnieje w grafie taki cykl, który składa się z każdego wierzchołka i każdy wierzchołek występuje w nim dokładnie jednokrotnie (oprócz pierwszego wierzchołka cyklu).
- Problem marszrutyzacji z ograniczeniem pojemności (ang. *Capacitated Vehicle Routing Problem*, CVRP) – oprócz miast dany jest punkt dostawczy z którego, za pomocą identycznych pojazdów dostawczych, konieczne jest zaspokojenie danego zapotrzebowania w miastach. Rozwiązaniem dopuszczalnym problemu jest zbiór tras dla pojazdów dostawczych przy jednoczesnym nieprzekraczaniu limitu ich pojemności.

Dodatkowo ważny jest sposób określania odległości pomiędzy miastami czy też węzłami. W samym TSPLIB można spotkać co najmniej kilka sposobów jej przedstawienia, jak na przykład:

- sposób jawny, odległość dana jako liczba całkowita (metoda może utrudniać wizualizację nawet jeżeli odległości odpowiadają odległościom miast w wybranej przestrzeni),
- miasta znajdują się w przestrzeni 2D lub 3D z normą euklidesową,
- miasta znajdują się w przestrzeni 2D lub 3D z normą Manhattan (modelowanie kratownicy czy też specyficznych ograniczeń urbanistycznych, nie dopuszczających „ruchu po skosie"),
- miasta są opisane współrzędnymi geograficznymi, dystans pomiędzy nimi obliczany jest po wyidealizowanym modelu Ziemi, sferze o promieniu 6378,388 km.

Sposób wyznaczania odległości jest kluczowy aby zachować możliwość porównywania otrzymanych wyników pomiędzy badaniami. Dokumentacja do TSPLIB jasno wskazuje, że znalezienie nowej, lepszej kolejności niż podana w dokumentacji, najczęściej związana jest z zastosowaniem niewłaściwego sposobu obliczania odległości. To de facto oznacza rozwiązywanie innej instancji problemu (błąd może wynikać np. ze sposób zaokrąglania liczb rzeczywistych). W prowadzonych badaniach należy również uwzględnić istnienie dedykowanego oprogramowania do rozwiązywania TSP – Concorde[1]. Według jego twórców jest to najszybsze narzędzie w swojej klasie, pozwalające uzyskać rozwiązania optymalne dla nawet bardzo dużych instancji – w tym wszystkie instancje z TSPLIB, czyli nawet 89 000 miast.

[1]Zalecane materiały powiązane to: a) strona solvera: https://www.math.uwaterloo.ca/tsp/concorde/index.html dostępna na dzień 13.11.2021 b) książka [6] o problemie komiwojażera, napisana przez twórców solvera z opisem jego wykorzystania w ramach problemu TSP.

Wybór lub definicja wariantu problemu jest efektem „zanurzenia" problemu teoretycznego w wymaganiach „rzeczywistości". Na przykład asymetryczność odległości może wynikać z konieczności uwzględniania dróg jednokierunkowych. Miasta lub punkty odbioru mogą być dostępne tylko w określonych oknach czasowych, ze względu na organizację pracy lub preferencje odbiorców. Dodatkowe ograniczenia mogą wynikać też z prawa, np. ograniczony czas pracy kierowców w CVRP.

2.3 Problemy szeregowania zadań

Ze względu na liczbę publikacji związanych z zagadnieniem szeregowania, problemy należące do tej kategorii zostaną przedstawione w sposób bardziej szczegółowy wraz z omówieniem ich terminologii, korzystając z książki Smutnickiego [258]. Problemy szeregowania skupiają się na jak najlepszym wykorzystaniu zasobów w celu wykonania określonych zadań (zleceń). Przykładowym procesem może być proces montażu samochodu czy też realizacja budowy bloku w budownictwie. Zasobami w tym kontekście są zarówno personel, kapitał, surowce, narzędzia, jak i maszyny, które posiadają określone cechy, np. zakresy dostępności, ilości, kosztu, podzielności, odnawialności, etc. Zadania również posiadają pewne cechy, np.: termin gotowości, żądany termin zakończenia, możliwość przerywania wykonywania, możliwość podzielności operacji.

W klasycznych problemach szeregowania zadań wykorzystuje się uproszczony model rzeczywistości – podstawowymi pojęciami są: zadanie, operacja oraz maszyna. Zadanie modeluje pewien proces składający się z pojedynczych operacji, które muszą być wykonane na maszynach. Zwykle zbiór n zadań jest oznaczany przez $\mathcal{J} = \{1, 2, \ldots, n\}$, z kolei zbiór o operacji oznaczamy jako $\mathcal{O} = \{1, 2, \ldots, o\}$. Operacje muszą być wykonane na m maszynach ze zbioru $\mathcal{M} = \{1, 2, \ldots, m\}$. Dla każdej operacji określane są parametry takie jak:

p_i^a – czas wykonania operacji $i \in \mathcal{O}$ na maszynie $a \in \mathcal{M}$; górny indeks można pominąć jeżeli operacja może być wykonywana wyłącznie na jednej, dedykowanej maszynie,
S_i – termin rozpoczęcia wykonywania operacji $i \in \mathcal{O}$,
C_i – termin zakończenia wykonywania operacji $i \in \mathcal{O}$.

Terminy rozpoczęć i zakończeń wszystkich operacji zebrane są najczęściej w wektorach S i C, definiując *harmonogram*.

Teoria szeregowania obejmuje wiele zróżnicowanych problemów. W celu uporządkowanego ich opisu przyjęło się stosować notację Grahama z 1979

roku [96], charakteryzującą się trójpolowym opisem problemu w postaci:

$$\alpha|\beta|\gamma,$$

gdzie pole α opisuje rodzaj problemu, pole β dodatkowe ograniczenia, a pole γ funkcję celu. Notacja ta była wielokrotnie rozszerzana, miedzy innymi w pracach [4, 3, 51, 224].

Pole α zostanie omówione zgodnie z rozszerzeniem zaproponowanym przez Smutnickiego [257, 258], jako trzy symbole α_3, α_2 i α_1. Symbol α_1 to liczba maszyn w systemie (w wypadku systemów jednomaszynowych α_1 może być pominięte). Symbol α_2 opisuje typ problemu. wraz ze sposobem realizacji zadań. Dla przykładu, symbol FP w tym polu oznacza permutacyjny problem przepływowy szeregowania zadań (ang. *Permutation Flow Shop Scheduling Problem, PFSSP*), F – niepermutacyjny problem przepływowy szeregowania zadań (ang. *Flow Shop Scheduling Problem, FSSP*), J – problem gniazdowy szeregowania zadań (ang. *Job Shop Scheduling Problem, JSSP*), I – problem szeregowania zadań z równoległymi maszynami (ang. *Parallel Machine Scheduling Problem, PMSP*). Symbol α_3 określa typ maszyny, wraz ze sposobem wykonywania operacji na maszynach, o ile operacja może być wykonana na więcej niż jednej maszynie (potrzebna jest decyzja). Symbol P w tym polu oznacza jednakowe maszyny równoległe (ang. *identical parallel machines*), symbol Q – jednorodne maszyny równoległe (ang. *uniform parallel machines*), symbol R – niejednorodne maszyny równoległe (ang. *unrelated parallel machines*). Symbol α_3 jest pomijany jeśli każda z operacji ma dedykowaną maszynę na której ma być wykonana.

Pole β odpowiada za dodatkowe ograniczenia narzucone na problem. Za prawodpodobnie najpopularniejszy przykład mogą posłużyć przezbrojenia, czyli dodatkowy czas, który musi upłynąć pomiędzy wykonywaniem dwóch kolejnych operacji. Przezbrojenia modelują czas potrzebny na dostosowanie maszyny do wykonywania operacji, np. wymianę końcówki roboczej. Przykładowe rodzaje przezbrojeń które można opisać za pomocą pola β: ST_{si} – z czasami przezbrojeń niezależnymi od kolejności, ST_{sd} – z czasami przezbrojeń zależnymi od kolejności, SC_{sd} – z kosztami przezbrojeń uzależnionymi od kolejności. Innym ograniczeniem w polu β może być wymaganie ciągłej pracy maszyn bez przestojów pomiędzy operacjami (ang. *no idle*). Ograniczenie to jest często spotykane w branżach gdzie przestój maszyny generuje znaczne koszty. Obok ograniczenia na przestój maszyn, istnieje ograniczenie wymuszające brak przerw pomiędzy operacjami w ramach zadania (ang. *no-wait*). Ograniczenie to spotykane jest np. w metalurgii, gdy obrabiany materiał nie może „czekać” na kolejny etap obróbki. Wymienione

ograniczenia nie zawsze muszą mieć charakter bezwzględny, t.j. mogą być dopuszczalne od nich pewne odstępstwa. Na przykład, zamiast wymagania braku przerw, wymaganie może przyjąć postać limitu co do czasu oczekiwania (ang. *limited wait*).

Pole γ odpowiada za opis funkcji celu problemu. Jedną z najczęściej rozważanych funkcji celu jest długość uszeregowania (ang. *makespan*), czyli kryterium czasu zakończenia wszystkich operacji $C_{\max}$

$$C_{\max} \stackrel{\text{def}}{=} \max_{i \in \mathcal{O}} C_i - \min_{i \in \mathcal{O}} S_i, \tag{2.6}$$

które przybiera postać:

$$C_{\max} \stackrel{\text{def}}{=} \max_{i \in \mathcal{O}} C_i, \tag{2.7}$$

gdy operacje wykonywane są od chwili 0 (czyli $\min_{i \in \mathcal{O}} S_i = 0$). Innymi przykładami funkcji celu są: T – czas cyklu, $\sum w_i T_i$ – ważona suma spóźnień (ang. *total weighted tardiness*), TSC – całkowity koszt wykonywania przezbrojeń (ang. *total setup cost*), czy TST – całkowity czas wykonywania przezbrojeń (ang. *total setup time*). Pole γ nie może pozostać puste w ramach przyjętej notacji.

Przedstawioną próbę bardzo ogólnego zarysowania szerokiej dziedziny problemów szeregowania zadań należy uznać za dość „karkołomną", ale jej celem jest łatwiejszy odbiór problemów opisanych w kolejnych rozdziałach i podrozdziałach pracy. Dodatkowo próba ta stanowi jedynie wstęp do zagadnienia opisywania rzeczywistości za pomocą modeli matematycznych, które następnie poddawane są optymalizacji. W modelowaniu rzeczywistych systemów (tematyka która wykracza poza ramy tej pracy), trzeba uwzględnić znacznie więcej informacji o systemie – choćby jakość danych. Za [258], parametry problemu mogą mieć różną formę:

- deterministyczną – parametry przyjmują ustalone i niezmienne wartości (np. ruch ramienia robota transportowego jest przewidywalny),
- probabilistyczną – parametry są nieznane ale mogą być opisane zmiennymi losowymi lub rozkładami empirycznymi (np. popyt na wytwarzane produkty jest zmienny),
- rozmytą – parametry opisane są wielkościami rozmytymi, ze zdefiniowanymi funkcjami przynależności (podejście może np. ułatwić skodyfikowanie praktycznej wiedzy ekspertów).

2.4 Problem przepływowy

Problem przepływowy stanowi jeden z najstarszych problemów szeregowania zadań. Jego inspiracją mogły być już pierwsze linie montażowe, takie jak

linia produkcyjna Modelu T Henry'ego Forda, uruchomiona w 1913 roku [8]. Dwa klasyczne warianty problemu to permutacyjny problem przepływowy oraz niepermutacyjny problem przepływowy. Oba wymienione problemy przepływowe (z kryterium $C_{\max}$) są NP-trudne.

W obu wspomnianych wariantach problemu występuje o nieprzerywalnych operacji, „pogrupowanych" w ramach n zadań. Opierając się na formalnym zapisie przedstawionym w książce C. Smutnickiego [258], każde zadanie $j \in \mathcal{J}$ składa się z m operacji, numerowanych kolejno od $1+(j-1)m$ do $m+(j-1)m$, i wykonywanych w kolejności wynikającej z numeracji,

$$1+(j-1)m \;\rightarrow\; 2+(j-1)m \;\rightarrow \cdots \rightarrow\; m+(j-1)m, \tag{2.8}$$

na maszynach $1, 2, \ldots, m$. Zależność ta jest odpowiednikiem wymagań w produkcji np. mebli, gdzie wycięcie kształtu musi nastąpić koniecznie przed lakierowaniem – kolejne etapy produkcji tym samym modelowane są w postaci kolejnych maszyn. W tym uproszczonym przykładzie różnice pomiędzy zadaniami mogłyby odpowiadać różnicom w czasie wykonywania różnych rodzajów produktów meblarskich.

Rozwiązanie w problemach przepływowych może być reprezentowane przez zdefiniowany wcześniej harmonogram, zadany przez parę $S, C \in \mathbb{R}^o$. Brak możliwości przerywania operacji (t.j. dla każdego $i \in \mathcal{O}$ spełniona jest zależność $C_i = S_i + p_i$) oznacza, że jeden z wektorów jest wystarczający aby opisać rozwiązanie problemu, ponieważ łatwo jest jednoznacznie odtworzyć drugi. W praktyce częściej wykorzystywany jest wektor C, ze względu na popularność kryterium optymalizacji jakim jest $C_{\max}$ (definiowany równaniem (2.7)), czyli kryterium minimalizacji czasu zakończenia wszystkich operacji.

2.4.1 Permutacyjny problem przepływowy

Permutacyjny problem przepływowy (ang. *Permutation Flow Shop Scheduling Problem*, PFSSP) z kryterium minimalizacji czasu zakończenia wszystkich operacji, zgodnie z wcześniej przedstawioną notacją może być zapisany jako $FPm||C_{\max}$. W problemie każda maszyna przetwarza zadania w tej samej kolejności. Z uwagi na specyfikę ograniczeń problemu, rozwiązanie nie musi być reprezentowane przez harmonogram, ale użyteczna jest również postać kolejnościowa

$$\pi = (\pi(1), \pi(2), \ldots, \pi(j), \ldots, \pi(n)), \tag{2.9}$$

gdzie $\pi(j)$ oznacza numer zadania wykonywanego jako j-te. Kolejność ta jest jedną z możliwych permutacji elementów zbioru zadań $\mathcal{J}$.

Formalnie, ograniczenia problemu można zapisać jako

$$\forall i \in \mathcal{O} \qquad C_i = S_i + p_i, \tag{2.10}$$

$$\forall a \in \mathcal{M}\ \forall i \in J \setminus \{1\} \qquad S_{a+(\pi(i)-1)m} \geqslant C_{a+(\pi(i-1)-1)m}, \tag{2.11}$$

$$S_{1+(\pi(1)-1)m} = 0, \tag{2.12}$$

$$\forall a \in \mathcal{M} \setminus \{1\}\ \forall i \in J \qquad S_{a+(i-1)m} \geqslant C_{a+(i-1)m-1}, \tag{2.13}$$

przy czym: (2.10) odpowiada za brak możliwości przerywania operacji już rozpoczętej na maszynie – wymagana jest ciągłość wykonywania operacji. Ograniczenie (2.11) oznacza, że w ramach każdej z maszyn dla kolejności π, operacja z zadania j zakończy się, zanim rozpocznie się operacja następnika zadania j na tej maszynie. Wyjątkiem jest moment rozpoczęcia pierwszego zadania z kolejności π, ponieważ nie ma ono poprzednika. Dla uproszczenia, przyjmuje się zero za moment rozpoczęcia procesu, „zakotwiczając" harmonogram (ograniczenie (2.12)). Ograniczenie (2.13) odnosi się do kolejności przechodzenia zadania przez kolejne maszyny $1 \to 2 \to 3 \to \cdots \to m$.

2.4.2 Niepermutacyjny problem przepływowy

Niepermutacyjny problem szeregowania zadań (ang. *non-permutation Flow Shop Scheduling Problem* – FSSP) z kryterium minimalizacji czasu zakończenia wszystkich operacji może być opisany zgodnie z wcześniej przedstawioną notacją jako $Fm||C_{\max}$. Problem FSSP odróżnia się od wariantu przepływowego tym, że kolejność wykonywania zadań na maszynach może być różna dla każdej maszyny. To sprawia, że rozwiązanie problemu przyjmuje postać „kolejności złożonej z kolejności"; czyli m kolejności, każda określająca kolejność wykonywania zadań na innej maszynie. Formalnie tak zdefiniowane rozwiązanie może być zapisane jako

$$\pi = (\pi_1, \pi_2, \ldots, \pi_a, \ldots, \pi_m), \tag{2.14}$$

$$\pi_a = (\pi_a(1), \pi_a(2), \ldots, \pi_a(n)), \tag{2.15}$$

gdzie $\pi_a(j)$ jest numerem operacji która jest wykonywana jako j-ta ($j \in \mathcal{J}$) na maszynie $a \in \mathcal{M}$. Łatwo obliczyć, że operacja ta należy do zadania $\pi_a(j) - (a-1)m$.

Ograniczenia wariantu niepermutacyjnego FSSP są analogiczne do ograniczeń dla wariantu permutacyjnego, w szczególności ograniczenia gwarantujące nieprzerywalność operacji (2.10) oraz kolejność technologiczną (2.13). Z kolei ograniczenia wymuszające kolejność maszynową (2.11) i za-

kotwiczające harmonogram (2.12), przyjmują następującą postać:

$$\forall a \in \mathcal{M}\ \forall i \in J \setminus \{1\} \quad S_{a+(\pi_a(i)-1)m} \geqslant C_{a+(\pi_a(i-1)-1)m}, \tag{2.16}$$

$$\forall a \in \mathcal{M}\ \forall i \in J \setminus \{1\} \quad S_{\pi_a(i)} \geqslant C_{\pi_a(i-1)}, \tag{2.17}$$

$$S_{\pi_1(1)} = 0. \tag{2.18}$$

Warto przy tym zwrócić uwagę, że każde rozwiązanie spełniające ograniczenie (2.11) spełnia również ograniczenie (2.17) – tym samym permutacyjny problem przepływowy jest szczególnym przypadkiem bardziej ogólnego, niepermutacyjnego problemu przepływowego.

Cytując za [258], niepermutacyjny problem przepływowy, pomimo że pozwala uzyskać mniejsze wartości funkcji celu, to jest rzadziej analizowany w literaturze, ze względu na:

- różnice w wartościach funkcji celu optymalnych rozwiązań tych samych instancji rozumianych jako PFSSP i FSSP są w praktyce zwykle niewielkie (pomimo iż w najgorszym przypadku różnica może być bardzo znacząca);
- lepsze wartości $C_{\max}$ okupione są znaczną różnicą w rozmiarze przestrzeni rozwiązań – dla FSSP jest to $(n!)^m$ rozwiązań, w porównaniu do „jedynie" $n!$ dla PFSSP;
- obsługa kolejnych maszyn jest prostsza w wypadku systemu PFSSP, zgodna z regułą FIFO, wymaga jedynie jednej kolejności wprowadzania zadań do systemu;
- część procesów technologicznych, można modelować wyłącznie w postaci rozwiązań permutacyjnych;
- dla niektórych instancji różnych problemów przepływowych istnieją rozwiązania optymalne leżące w klasie rozwiązań permutacyjnych.

Rozwiązania PFSSP mogą być używane jako rozwiązania początkowe dla algorytmów rozwiązujących FSSP jako obiecujące rozwiązanie startowe. Podobnie, instancje testowe PFSSP mogą również posłużyć dla ewaluacji algorytmów dla FSSP, ponieważ problemy różnią się wyłącznie ograniczeniami oraz postacią rozwiązania.

2.5 Problem gniazdowy

Problem gniazdowy (ang. *Job Shop Scheduling Problem*, JSSP) jest zgodnie z przyjętą notacją oznaczony przez $Jm||C_{\max}$ (dla kryterium $C_{\max}$ i m maszyn). Podobnie jak w wypadku problemu przepływowego, w problemie JSSP występuje zbiór operacji $\mathcal{O}$, pogrupowanych w ramach n zadań, które

należy wykonać na m maszynach. Jednak w ramach problemu JSSP zadanie nie musi być wykonywane na każdej z dostępnych maszyn. Zadania mogą składać się z różnej liczby operacji, wykonywanych na różnych maszynach; dodatkowo istnieje możliwość ponownego odwiedzenia maszyny w ramach zadania (kolejność odwiedzenia maszyn w ramach zadań jest nadal zawsze z góry ustalona). Formalnie, za zapisem wykorzystanym w książce C. Smutnickiego [258], zadanie $j \in \mathcal{J}$ polega na sekwencyjnym wykonaniu n_j operacji, numerowanych od $1 + l_{j-1}$ do $n_j + l_{j-1}$, i wykonywanych w kolejności wynikającej z numeracji,

$$1 + l_{j-1} \;\rightarrow\; 2 + l_{j-1} \;\rightarrow \cdots \rightarrow\; n_j + l_{j-1}, \tag{2.19}$$

gdzie l_j oznacza sumaryczną liczbę operacji należących do zadań od 1 do j, co można zapisać formalnie jako

$$l_j \stackrel{\text{def}}{=} \begin{cases} \sum_{i=1}^{j} n_i & \text{gdy } j = 1, 2, \ldots, n, \\ 0 & \text{gdy } j = 0. \end{cases} \tag{2.20}$$

Liczba operacji wykonywanych na maszynie $a \in \mathcal{M}$ oznaczono przez m_a.

Korzystając z wprowadzonej notacji, ograniczenia JSSP można zapisać jako układ równań

$$\forall i \in \mathcal{O} \qquad C_i = S_i + p_i, \tag{2.21}$$

$$\forall j \in \mathcal{J}\ \forall i \in \{1, 2, \ldots, n_j - 1\} \qquad C_{i+l_{j-1}} \leqslant S_{i+l_{j-1}+1}, \tag{2.22}$$

$$\forall a \in \mathcal{M}\ \forall i \in \{1, 2, \ldots, m_a - 1\} \qquad C_{\pi_a(i)} \leqslant S_{\pi_a(i+1)}, \tag{2.23}$$

$$\min_{a \in \mathcal{M}} S_{\pi_a(1)} = 0, \tag{2.24}$$

gdzie (2.21) odpowiada za ciągłość wykonywania operacji, ograniczenie (2.22) odpowiada za kolejność technologiczną, a ograniczenie (2.23) za kolejność maszynową. Opcjonalne ograniczenie (2.24) zakotwicza początek harmonogramu w momencie 0.

Problem gniazdowy jest problemem bardziej ogólnym niż wcześniej opisany problem przepływowy. Różnice pomiędzy tymi problemami można przedstawić w sposób uproszczony na następującym przykładzie. FSSP może służyć do modelowania wyspecjalizowanej linii produkcyjnej, stworzonej z myślą o efektywnym produkowaniu z góry określonych produktów (wykorzystanie wszystkich maszyn dla każdego z zadań). Z kolei JSSP może być wykorzystany do modelowania produkcji bardziej elastycznej np. małych zakładów meblowych – zadania (kontynuując przykład: pewne rodzaje mebli) mogą wykorzystywać tylko pewną część wszystkich maszyn (nie zawsze potrzebne może być np. lakierowanie).

Podobnie jak dla innych problemów szeregowania zadań, także dla JSSP rozważane są różne rodzaje funkcji celu. Do najpopularniejszych należą: minimalizacja czasu zakończenia wykonywania wszystkich zadań ($C_{\max}$) oraz minimalizacja sumy terminów zakończenia zadań ($\sum C_i$). Kryterium $C_{\max}$ (definiowane równaniem (2.7)) można przedstawić w prosty sposób jako czas wykonania ostatniej operacji, co odpowiada najpóźniejszemu czasowi zakończenia spośród ostatnich operacji na każdej z maszyn lub czasowi zakończenia ostatniej operacji zadania które kończy się najpóźniej:

$$C_{\max}(\pi) = \max_{i \in \bigcup_{a \in \mathcal{M}} \{\pi_a(m_a)\}} C_i = \max_{i \in \bigcup_{j \in \mathcal{J}} \{l_j\}} C_i. \tag{2.25}$$

Kryterium $\sum C_i$, nazywane również minimalizacją całkowitego czasu przepływu (ang. *Total Flow Time*), może zostać z kolei wyrażone wzorem

$$\sum C_i \stackrel{\text{def}}{=} \sum_{j=1}^{n} C_i. \tag{2.26}$$

W ramach badań opisanych w dysertacji, wykorzystanym kryterium optymalizacji będzie jedynie pierwsze z wymienionych, czyli $C_{\max}$. Podobnie jak dla wcześniej opisanych problemów szeregowania zadań, istnieją liczne wariacje problemu bazowego spotykane w literaturze. Na przykład, w zbiorze dostępnych maszyny mogą występować duplikaty (lub maszyny podobne), co doprowadza do konieczności wyboru pomiędzy maszynami dla zadanych operacji (elastyczny problem przepływowy, ang. *Flexible Job Shop* [50]). Istnieje też pewien trend związany z przemysłem 4.0, aby w ramach modelowania produkcji uwzględniać jej rozproszenie [73, 311]. Oczywiście istnieją też warianty problemu obejmujące dodanie standardowych ograniczeń, takich jak brak możliwości wystąpienia przerw pomiędzy operacjami w ramach zadania [242].

Duża część algorytmów przeznaczonych do rozwiązywania problemu gniazdowego testowa jest w pierwszej kolejności na instancjach wygenerowanych przez Taillard'a [270]. Tym samym w dużej mierze badana jest efektywność rozwiązywania na szczególnie trudnych instancjach o rozmiarze z przedziału od 15 zadań na 15 maszynach (225 operacji) do 100 zadań na 20 maszynach (2 000 operacji). Jednak cytując za [258], przy wykorzystaniu dostępnej mocy obliczeniowej oraz obliczeń równoległych, możliwe jest uzyskanie „dobrych" (ale nie optymalnych) rezultatów dla instancji liczących do 100 000 operacji.

2.6 Cykliczny problem gniazdowy

Cykliczny problem gniazdowy (ang. *Cyclic Jobs Shop Scheduling Problem*, CJSSP) jest wariantem omawianego wcześniej JSSP, służącym do modelowania powtarzających się procesów produkcyjnych. Procesy takie mogą być elementem na przykład produkcji masowej, niezwykle istotnej w konteście współczesnej globalnej gospodarki.

Podstawowym założeniem CJSSP jest powtarzalność harmonogramu, t.j. fragment harmonogramu powtarza się cyklicznie. Czas cyklu T to okres powtórzeń harmonogramu. Zadania i operacje składające się na pojedyncze powtórzenie harmonogramu konstytuują MPS (od ang. *Minimal Part Set*). Powtórzenie x harmonogramu jest zadane przez $S^x = (S_1^x, S_2^x, \ldots, S_o^x)$, gdzie S_i^x to moment rozpoczęcia instancji operacji $i \in \mathcal{O}$ w MPS-ie x. Analogicznie definiowany jest wektor C^x. Dodatkowym ograniczeniem, mającym na celu uproszczenie zarządzania procesem lub modelowanie pewnych procesów technologicznych, jest zabronienie przeplotu MPS-ów na maszynach. Innymi słowy, gdy na dowolnej maszynie rozpocznie się wykonywanie pewnego MPS-a, wszystkie operacje wykonywane na tej maszynie, a należące do tego MPS-a, muszą zostać zakończone, zanim kolejny MPS może być rozpoczęty.

Dla każdego MPS-a, indeksowanego przez $x = 1, 2, \ldots$, ograniczenia przyjmują postać

$$\forall i \in \mathcal{O} \qquad C_i^x = S_i^x + p_i, \tag{2.27}$$

$$\forall j \in \mathcal{J}\ \forall i \in \{1, 2, \ldots, n_j - 1\} \qquad C_{i+l_{j-1}}^x \leqslant S_{i+l_{j-1}+1}^x, \tag{2.28}$$

$$\forall a \in \mathcal{M}\ \forall i \in \{1, 2, \ldots, m_a - 1\} \qquad C_{\pi_a(i)}^x \leqslant S_{\pi_a(i+1)}^x, \tag{2.29}$$

$$\forall i \in \mathcal{O} \qquad S_i^x \geqslant 0, \tag{2.30}$$

$$\forall a \in \mathcal{M} \qquad C_{\pi_a(m_a)}^x \leqslant S_{\pi_a(1)}^{x+1}, \tag{2.31}$$

$$\forall i \in \mathcal{O} \qquad S_i^x + T = S_i^{x+1}, \tag{2.32}$$

gdzie ograniczenia (2.27)–(2.29) są analogiczne do tych z JSSP, ograniczenie (2.30) uniemożliwia wykonywania zadań przed czasem 0, ograniczenie (2.31) gwarantuje rozłączność MPS-ów, a ograniczenie (2.32) periodyczność harmonogramu. Rozwiązanie problemu może być reprezentowane przez kolejność wykonywania operacji π, ponieważ w czasie $O(nm^2)$ można wyznaczyć minimalny czas cyklu $T(\pi)$ dla tej kolejności oraz S^x harmonogramu dosuniętego w lewo dla dowolnego x (szczegółowe uzasadnienie znajduje się w [37]).

Tak sformułowane ograniczenia problemu odpowiadają cyklicznemu problemowi gniazdowemu z powtarzającymi się sekwencjami na maszynach

(ang. *Cyclic Job Shop with Machine Chains Repetition*), opisanego przez Kampmeyera w swojej dysertacji doktorskiej [134, str. 69]. Tym niemniej, przedstawiony tutaj model został zaczerpnięty z późniejszej pracy innych autorów [37]. Model ten ma przewagę (z uwagi na relatywnie niewielki rozmiar przestrzeni rozwiązań) w zastosowaniu do projektowania algorytmów przeszukiwania lokalnego dla technik analizy przestrzeni rozwiązań, które stanowią przedmiot zainteresowania dalszych badań własnych.

2.7 Cykliczny problem szeregowania i przydziału operacji w wielostanowiskowym gnieździe produkcyjnym

Przedstawione wcześniej modele problemów szeregowania zadań zakładały istnienie ustalonej i niezmiennej marszruty technologicznej. To znaczy, że każda z operacji była wykonywana na ściśle określonej maszynie, a poddawana optymalizacji była jedynie ich kolejność. Niemniej, jak zauważono w licznych pracach przeglądowych [50, 306, 85], elastyczne modele wytwarzania są również zagadnieniem niezwykle istotnym w praktyce. W niniejszej sekcji opisano cykliczny problem szeregowania i przydziału operacji w wielostanowiskowym gnieździe produkcyjnym, badany wcześniej w różnych wariantach i kontekstach m.in. w [93, 27, 26].

W problemie rozważane jest pojedyncze gniazdo produkcyjne z jednym operatorem. Składa się ono ze zbioru stanowisk $\mathcal{Q} = \{1, 2, \ldots, q\}$. Gniazdo produkcyjne wykonuje cyklicznie operacje ze zbioru $\mathcal{O} = \{1, 2, \ldots, n\}$. Podobnie jak w omówionym wcześniej CJSSP, operacje z $\mathcal{O}$ konstytuują MPS, który musi być wykonywany cyklicznie co czas cyklu T. W ramach pojedynczego MPS-a, operacje są wykonywane w kolejności zadanej przez π, przy czym $\pi(i)$ to numer operacji wykonywanej w i-tej kolejności. Każda operacja może być wykonywana na każdym ze stanowisk, a wykonywanie operacji $i \in \mathcal{O}$ na stanowisku $r \in \mathcal{Q}$ trwa p_i^r czasu. Pomiędzy każdymi dwiema operacjami, $i, j \in \mathcal{Q}$, wykonywanymi na tym samym stanowisku $r \in \mathcal{Q}$, musi zostać wykonane przezbrojenie trwające $s_{i\to j}^r$ czasu. Dla uproszczenia definicji problemu, przezbrojenie musi być wykonane bezpośrednio przed operacją która go wymaga. Przydział operacji do stanowisk wyrażony jest przez σ, gdzie $\sigma(i)$ określa na którym stanowisku wykonywana jest i-ta operacja. Dodatkowo, ze względu na obecność jednego operatora, w gnieździe jednocześnie może być wykonywana co najwyżej jedna operacja *albo* co najwyżej jedno przezbrojenie. Celem jest znalezienie dopuszczalnego harmonogramu o minimalnym czasie cyklu T. Korzystając z metody opisanej między innymi w [93], problem może być transformowany do wyznaczenia

takiej pary π, σ, która pozwala na zbudowanie harmonogramu o minimalnym czasie cyklu.

Aby sformułować problem formalnie, wprowadzone zostanie pojęcie poprzednika stanowiskowego i-tej w kolejności operacji:

$$\operatorname{prev}(\sigma,\pi,i) = \begin{cases} \max \mathcal{A}(\sigma,i) & \text{dla: } |\mathcal{A}(\sigma,i)| \geqslant 1, \\ \max\{j \in \mathcal{O} :\ \sigma(j) = \sigma(i)\} & \text{w przeciwnym razie,} \end{cases} \tag{2.33}$$

$$\mathcal{A}(\sigma,i) = \{0 < j < i :\ \sigma(i) = \sigma(j)\}. \tag{2.34}$$

Dla przydziału operacji do stanowisk σ, kolejności wykonywania operacji π oraz operacji i, poprzednik stanowiskowy $\operatorname{prev}(\sigma,\pi,i)$ to poprzednia operacja wykonywana bezpośrednio przed operacją i na maszynie $\sigma(i)$. Poprzednik stanowiskowy istnieje zawsze, ponieważ nawet jeżeli na stanowisku $\sigma(i)$ wykonywana jest tylko operacja i, to jej poprzednikiem jest operacja i z poprzedniego MPS-a.

Niech harmonogram dla MPS-a x będzie zadany przez S^x, C^x, analogicznie do CJSSP. Wtedy harmonogram jest dopuszczalny wtedy i tylko wtedy gdy spełnia ograniczenia:

$$\forall i \in \mathcal{O} \setminus \{o\} \quad S^x_{\pi(i+1)} \geqslant C^x_{\pi(i)} + s^{\sigma(i+1)}_{\operatorname{prev}(\sigma,\pi,i+1)\to\sigma(i+1)}, \tag{2.35}$$

$$S^{x+1}_{\pi(1)} \geqslant C^x_{\pi(o)} + s^{\sigma(1)}_{\operatorname{prev}(\sigma,\pi,1)}, \tag{2.36}$$

$$\forall i \in \mathcal{J} \qquad C^x_i = S^x_i + p^{\sigma(\pi^{-1}(i))}_i, \tag{2.37}$$

$$\forall i \in \mathcal{O} \qquad S^{x+1}_i = S^x_i + T, \tag{2.38}$$

gdzie $\pi^{-1}(i)$ to pozycja operacji i w π, tj. $\pi^{-1}(i) = j \iff \pi(j) = i$. Ograniczenie (2.35) reprezentuje kolejność wykonywania operacji w gnieździe oraz wymusza długość przezbrojeń. Ograniczenie (2.36) gwarantuje rozłączność MPS-ów. Równanie (2.37) odpowiada za nieprzerywalność wykonywania operacji, a równanie (2.38) cykliczność harmonogramu.

2.8 Wnioski i uwagi

W rozdziale przedstawiono przyjęty w dysertacji ogólny model problemów optymalizacji dyskretnej. Następnie, na podstawie tego modelu, opisano bardziej szczegółowo wybrane problemy. Zdecydowano się na opis jedynie problemów wykorzystywanych w dalszych badaniach stanowiących przedmiot dysertacji, wraz z ich wariantami bazowymi. Tym samym przegląd nie jest wyczerpujący, ale konstytuuje przyjęte w modelach oznaczenia i stanowi odniesienie do źródeł, w przypadku konieczności poszerzenia wiedzy

o omówionych zagadnieniach. Ze względu na zakres przeprowadzonych badań własnych, znaczącą część wybranych problemów stanowiły problemy szeregowania zadań.

Rozdział 3

Klasyczne metody optymalizacji dyskretnej

Większość problemów optymalizacji opisanych we wcześniejszym rozdziale należy do klasy problemów NP-trudnych. Oznacza to, że nie są znane dla nich algorytmy dokładne (czyli pozwalające wyznaczyć rozwiązanie optymalne) działające w czasie wielomianowym. W odpowiedzi na tę kwestię, powstało wiele algorytmów podejmujących się rozwiązania tych problemów, w tym algorytmów heurystycznych, które nie zapewniają wyznaczenia rozwiązania optymalnego, a niekiedy nawet dopuszczalnego.

W przeglądzie zdecydowano się na zarysowanie tylko wybranych algorytmów, przeznaczonych do rozwiązywania omawianych wcześniej problemów; ze szczególnym naciskiem na metody wykorzystywane w dalszych badaniach, opisanych w kolejnych rozdziałach. Algorytmy pokazano z uwzględnieniem ich podziału na metody dokładne oraz przybliżone. Drugą z grup rozbito z kolei na: algorytmy konstrukcyjne, algorytmy lokalnej poprawy oraz algorytmy populacyjne.

3.1 Metody dokładne

Oczywistą zaletą metod dokładnych jest gwarancja wyznaczenia najlepszego istniejącego rozwiązania – rozwiązania optymalnego. Niestety wiąże się to często z koniecznością eksploracji dużej części przestrzeni rozwiązań (czy też po prostu wszystkich rozwiązań, jak w przeglądzie zupełnym, czyli naiwnym podejściu siłowym – ang. *Brute Force*). Przekłada się to na gwałtowny wzrost czasu potrzebnego do rozwiązania problemu, wraz ze wzrostem jego rozmiaru. Jedną z metod zredukowania czasu rozwiązywania jest wykorzystywanie właściwości badanego problemu przy projektowaniu

metod dokładnych. Własności mogą pozwolić na rozwiązywanie coraz większych instancji, choć niestety ostatecznie przydatność metod dokładnych ogranicza się zwykle do przykładów o relatywnie niewielkich rozmiarach.

3.1.1 Programowanie liniowe oraz programowanie liniowe całkowitoliczbowe

Opis metody programowania liniowego zostanie przedstawiony na podstawie książki Cormen i in. [61][1]. Programowanie liniowe jest metodą umożliwiającą rozwiązywanie szerokiej klasy problemów, które można zapisać w postaci maksymalizacji lub minimalizacji liniowej funkcji celu, przy zachowaniu ograniczeń zmiennych tej funkcji opisanych w postaci układu równań lub nierówności liniowych. Problemy które mogą zastać przedstawione w tej postaci nazywamy problemami programowania liniowego (ang. *Linear Programming*, LP).

Dla zbioru n zmiennych decyzyjnych $x_1, x_2, \ldots, x_n \in \mathbb{R}$ oraz zbioru parametrów $a_1, a_2, \ldots, a_n \in \mathbb{R}$, możliwe jest zdefiniowanie liniowej funkcji

$$f(x_1, x_2, \ldots, x_n) = a_1x_1 + a_2x_2 + \cdots + a_nx_n = \sum_{j=1}^{n} a_jx_j, \tag{3.1}$$

dla której konieczne jest spełnienie ograniczeń wyrażonych w postaci równań liniowych (b jest liczbą rzeczywistą)

$$f(x_1, x_2, \ldots, x_n) = b, \tag{3.2}$$

lub nierówności liniowych w postaci

$$f(x_1, x_2, \ldots, x_n) \leqslant b, \tag{3.3}$$

lub

$$f(x_1, x_2, \ldots, x_n) \geqslant b, \tag{3.4}$$

określanych mianem ograniczeń liniowych. Problemy optymalizacji liniowej są zwykle przedstawiane w formie nazywanej postacią standardową

$$\min / \max \sum_{j=1}^{n} c_jx_j \tag{3.5}$$

z ograniczeniami:

$$\sum_{j=1}^{n} a_{ij}x_j \leqslant b_i \text{ dla } i = 1, 2, \ldots, m, \tag{3.6}$$

$$x_j \geqslant 0 \qquad \text{dla } j = 1, 2, \ldots, n. \tag{3.7}$$

[1]Pozycja polecana również dla innych metod opisanych w tym rozdziale dysertacji.

Programowanie liniowe mieszane całkowitoliczbowe (ang. *Mixed Integer Linear Programming*, MILP) jest metodą rozwiązywania klasy problemów, które można wyrazić jako problemy programowania linowego, w którym na niektóre lub wszystkie zmienne decyzyjne nałożono warunek, iż muszą przyjmować wartości całkowite. Problemy LP i MILP mogą być rozwiązywane za pomocą dedykowanego oprogramowania, w tym popularnych pakietów komercyjnych takich jak IBM CPLEX oraz Gurobi Optimizer[2] czy też bezpłatnego, otwartego oprogramowania takiego jak Google OR-Tools. Użyteczność obu opisanych metod jest ograniczona przez wielkość rozwiązywanych instancji. Algorytmy rozwiązujące LP i MILP często stanowią dobry punkt odniesienia dla innych metod dokładnych i metod heurystycznych, jako baza do porównania.

3.1.2 Schemat podziału i ograniczeń

Schemat podziału i ograniczeń (ang. *Branch and Bound*, B&B) [160] to nazwa popularnej strategii przeszukiwania zbioru dopuszczalnych rozwiązań, pozwalającej na konstrukcję efektywnych dokładnych algorytmów dedykowanych. Główną ideą jest wykorzystanie strategii pozwalającej w niektórych przypadkach na nierozpatrywanie wszystkich rozwiązań (jak ma to miejsce w przypadku podejścia siłowego). Wykorzystuje się w tym celu metodę przeszukiwania drzewa rozwiązań, z uwzględnieniem reguł odrzucania. Pozwalają one „odcinać" niektóre jego gałęzie gdy pewne jest, że nie zawierają rozwiązania optymalnego.

Aby tego dokonać, schemat wykorzystuje informacje o górnym (ang. *Upper Bound*, UB) oraz dolnym ograniczeniu (ang. *Lower Bound*, LB). Górne ograniczenie stanowi oszacowanie wartości funkcji celu rozwiązania optymalnego. Dla zadania minimalizacji, jest to największa wartość jaką może przyjąć funkcja celu rozwiązania optymalnego. Zwykle, górnym ograniczeniem jest wartość funkcji celu najlepszego dotychczas rozważanego rozwiązania. Natomiast dolne ograniczenie wyznaczane jest dla zadanego zbioru rozwiązań problemu. Dla zadania minimalizacji, LB dla rozwiązań zbioru S mówi, że dla każdego rozwiązania $s \in S$, $\mathrm{LB} < f(s)$. Jeżeli $\mathrm{LB} > \mathrm{UB}$, to rozwiązania z S nie muszą być rozważane, ponieważ na pewno nie znajduje się pośród nich rozwiązanie optymalne. W schemacie podziału i ograniczeń, przestrzeń rozwiązań jest sukcesywnie dzielona na podzbiory, na bieżąco aktualizując UB i wyliczając LB dla kolejnych podzbiorów. Możliwość *szybkiego* i jak najściślejszego obliczenia LB jest kluczowa do stworzenia efektywnego algorytmu podziału i ograniczeń. Pozwala ona „wycinać" duże

[2]Gurobi Optimizer jest uważany przez twórców za najszybsze i najpotężniejsze oprogramowanie do rozwiązywania problemów programowania matematycznego.

obszary przestrzeni rozwiązań, nie zawierające rozwiązania optymalnego. W pesymistycznym przypadku – pomimo zastosowanych reguł odrzucania – dla algorytmów B&B konieczne może być rozważenie wszystkich rozwiązań, co sprawia że złożoność obliczeniowa często jest tożsama ze złożonością obliczeniową podejścia siłowego.

Przykładem zastosowania schematu podziału i ograniczeń dla problemu komiwojażera jest algorytm w którym w każdym węźle rozważane jest kolejne miasto w konstruowanej trasie stanowiącej rozwiązanie. Nie ma sensu „rozwijać" gałęzi dla których dolne ograniczenie (definiowane dalej) jest większe od długości najlepszej dotychczas znalezionej trasy (górne ograniczenie). Jednym z najprostszych dolnych ograniczeń jest długość trasy od korzenia drzewa do wierzchołka i suma długości najkrótszych dopuszczalnych połączeń dla jeszcze nie odwiedzonych miast.

Algorytmy dokładne wykorzystujące schemat podziału i ograniczeń są z powodzeniem stosowane dla wielu problemów optymalizacji, między innymi: problemu komiwojażera [171], problemu marszrutyzacji pojazdów [283], problemu plecakowego [159] czy problemów szeregowania zadań [205].

3.1.3 Programowanie dynamiczne

Programowanie dynamiczne (ang. *Dynamic Programming*, DP) to technika rozwiązywania problemów optymalizacji poprzez „rozbicie" problemu na łatwiejsze podproblemy w sposób rekurencyjny, a następnie wykorzystaniu optymalnych rozwiązań podproblemów w celu stopniowego „uzyskania" rozwiązania optymalnego problemu pierwotnego. Nie wszystkie problemy można rozbić w ten sposób. O problemach dla których jest to możliwe mówimy, że posiadają własność optymalnej podstruktury. Innymi słowy, ich rozwiązanie optymalne jest funkcją optymalnych rozwiązań podproblemów.

Aby stworzyć algorytm bazujący na schemacie programowania dynamicznego można skorzystać z kroków:

1. Opisanie struktury optymalnego rozwiązania z wykorzystaniem definicji podproblemów i zależności pomiędzy nimi.
2. Wyznaczenie rekurencyjnej formuły pozwalającej na obliczanie wartości funkcji celu podproblemu, przy wykorzystaniu wartości funkcji celu innych podproblemów.
3. Uzyskanie wartości funkcji celu rozwiązania optymalnego korzystając z równań z punktu 2. – poprzez rozwiązanie podproblemów (zaczynając od najprostszych).
4. Konstrukcja rozwiązania optymalnego dzięki obliczeniom z punktu 3.

Podstawowy schemat DP zostanie przedstawiony na podstawie książki C. Smutnickiego [258]. Programowanie dynamiczne można rozumieć jako

transformację zadania rozwiązywania problemu optymalizacji dyskretnej do problemu sekwencyjnego podejmowania decyzji. Stan procesów na poszczególnych etapach oznaczony będzie poprzez $S_i,\ i = 1, 2, \ldots, N$, a poszczególne decyzje przez $d_i,\ i = 1, 2, \ldots, N$. To pozwala zapisać wieloetapowy proces decyzyjny w postaci transformacji $S_k = T(S_{k-1}, d_{k-1})$. Aby oceniać ciąg decyzji $d_1, d_2, \ldots, d_N$ wykorzystuje się związaną z procesem podejmowania decyzji funkcje celu $F(S_1, S_2, \ldots, S_N; d_1, d_2, \ldots, d_N)$, którą należy minimalizować. Dla znanego stanu S_1, przebieg wieloetapowego procesu decyzyjnego wyznaczonego poprzez ciąg decyzji dopuszczalnych $d_1, d_2, \ldots, d_N$, nazywamy strategią. Strategia minimalizująca funkcję F jest nazywana strategią optymalną.

Wyznaczenie strategii optymalnej dla przypadku ogólnego ma postać trudnego zadania optymalizacji nieliniowej

$$\min_{d1,d2,d3,\ldots d_N} F(S_1, S_2, \ldots, S_n; d_1, d_2, \ldots, d_N) \tag{3.8}$$

$$S_k = T(S_{k-1}, d_{k-1}), \quad k = 2, \ldots, N, \tag{3.9}$$

gdzie stan S_1 jest dany. Dla rozwiązywalności problemu w praktyce kluczowa jest tzw. własność Markowa problemu – mówiąca, że przebieg procesu w stanie k-tym nie zależy od historii procesu, lecz jedynie od stanu S_k. Tym samym zysk z zastosowania schematu DP jest możliwy dla podklasy problemów decyzyjnych bez pamięci. Ograniczenie to nie wyklucza zastosowania DP do rozwiązywania stosunkowo dużej klasy problemów o znaczeniu technicznym i ekonomicznym.

Strategia optymalna dla wieloetapowych procesów decyzyjnych z własnością Markowa charakteryzuje się zasadą optymalności Bellmana. Strategia taka ma własność, że niezależnie od stanu początkowego i decyzji początkowej – pozostałe decyzje muszą stworzyć strategię optymalną z punktu widzenia stanu wynikłego z pierwszej decyzji. Kontynuując wykorzystanie [258], przedstawiony zostanie przykład zastosowania DP dla zadania minimalizacji funkcji addytywnej

$$F(S_1, S_2, \ldots, S_N,\ d_1, d_2, \ldots, d_N) = \sum_{i=1}^{N} f_i(S_i, d_i). \tag{3.10}$$

Zgodnie z zasadą optymalności mamy

$$\begin{aligned} &\min_{d1,d2,\ldots,d_N} (f_1(S_1, d_1) + \cdots + f_N(S_N, d_N)) = \\ &= \min_{d_1}(f_1(S_1, d_1) + \min_{d2,\ldots,d_N} (f_2(S_2, d_2) + \cdots + f_N(S_N, d_N))) \end{aligned} \tag{3.11}$$

Oznaczając dla $k = 1, 2, .., N$

$$q_k(S_k) = \min_{d_k,\ldots,d_N} (f_k(S_k, d_k) + \cdots + f_N(S_N, d_N)), \tag{3.12}$$

otrzymamy

$$q_N(S_N) = \min_{d_N} f_N(S_N, d_N), \tag{3.13}$$

$$q_k(S_k) = \min_{d_k}(f_k(S_k, d_k) + q_{k+1}(S_{k+1})). \tag{3.14}$$

Ostatecznie, możemy zapisać rezultat w postaci równania Bellmana

$$q_k(S_k) = \min_{d_k}(f_k(S_k, d_k) + q_{k+1}(T(S_k, d_k))), \quad k = N, \ldots, 1, \tag{3.15}$$

gdzie

$$q_{N+1}(S_{N+1}) = 0. \tag{3.16}$$

Decyzję optymalną wyznacza się parametrycznie.

Programowanie dynamiczne jest wykorzystywane do budowy algorytmów dokładnych, między innymi do wyznaczanie najkrótszych ścieżek pomiędzy wszystkimi parami wierzchołków (algorytm Floyda-Warshalla [81]), czy do parsowania tekstu w przetwarzaniu języka naturalnego (algorytm Earley'a [79]). Metoda może być również wykorzystywana jako element składowy bardziej złożonych algorytmów, np. do przeglądania otoczenia w algorytmach poszukiwań lokalnych (np. algorytmie *Dynasearch* [60]). Złożoność obliczeniowa konkretnego algorytmu zależy od aplikacji i problemu – np. wielomianowa dla wyznaczania najdłuższej drogi w grafie, pseudowielomianowa dla problemu załadunku, wykładnicza dla TSP.

3.2 Metody przybliżone

Metody przybliżone (heurystyki) są wykorzystywanie ze względu na potrzebę kompromisu pomiędzy jakością uzyskanego rozwiązania, a czasem potrzebnym na jego wyznaczenie. Oczywistym przykładem obszaru zastosowań heurystyk są duże instancje problemów dla których złożoność czasowa lub pamięciowa metod dokładnych rośnie ponadwielomianowo. Algorytmy te mogą na przykład wymagać czasu liczonego w latach, co jest zwykle nie do przyjęcia. Tym samym za pierwsze kryterium oceny algorytmu przybliżonego należy uznać zasoby potrzebne do jego wykonania – co naturalnie prowadzi do pojęcia złożoności obliczeniowej algorytmu.

Drugim kryterium dla metod przybliżonych jest jakość znalezionego rozwiązania. W tym celu stosuje się porównanie do rozwiązania referencyjnego

lub innej wartości odniesienia. Rozwiązaniem referencyjnym może być rozwiązanie optymalne (o ile jest znane, co ma często miejsce w wypadku popularnych *benchmarków*, np. `TSPLIB` dla problemu komiwojażera) lub najlepsze znane w literaturze rozwiązanie. Z kolei wartością referencyjną jest często ograniczenie dolne wartości funkcji celu (kontynuując przykład problemu komiwojażera – np. suma długości najkrótszych dopuszczalnych połączeń). W celu budowy systematyki metod przybliżonych wykorzystana została praca [271], dzieląca je na: konstrukcyjne, przeszukiwania lokalnego oraz populacyjne.

3.2.1 Algorytmy konstrukcyjne

Klasa algorytmów konstrukcyjnych zawiera algorytmy, których rozwiązanie jest „konstruowane" krok po kroku w trakcie działania algorytmu dla dedykowanego problemu, najczęściej wiąże się to z podejmowaniem zachłannych decyzji w kolejnych krokach algorytmu – jednak lokalnie dobre decyzje nie gwarantują uzyskania finalnie rozwiązania globalnie optymalnego. Przykładem może być wybieranie najbliższego miasta jako kolejnego na ścieżce w trakcie „konstruowania" rozwiązywania dla problemu komiwojażera – co stanowi algorytm Najbliższego Sąsiada [145] (ang. *Nearest Neighbor*). Warto zwrócić uwagę, że algorytmów konstrukcyjnych nie można przerwać aby uzyskać szybciej rozwiązanie o gorszej jakości (dla przykładu wyżej dotyczącego komiwojażera – przerwanie działania algorytmu zwróci tylko niepełną część ścieżki – nie stanowiącej rozwiązania problemu) – tym samym uzyskujemy rozwiązanie „dobre" albo żadne.

W dziedzinie szeregowania zadań jednym z najbardziej znanych algorytmów konstrukcyjnych jest NEH [193]. Służy on do rozwiązywania problemu przepływnego z kryterium minimalizacji czasu zakończenia wszystkich zadań, choć jego warianty i modyfikacje mogą również być wykorzystywane dla innych problemów – na przykład INSA [198] dla problemu gniazdowego. Złożoność czasowa algorytmu NEH to $O(n^2m)$. Kolejne kroki algorytmu to:

1. Wyznacz priorytety dla każdego z zadań $\omega(j)$, czyli sumy wszystkich operacji danego zadania na wszystkich maszynach

$$\omega(j) = \sum_{i=1}^{m} p_{i,j}.$$

2. Posortuj zadania nierosnąco wg. $\omega(j)$, tak aby zadania o największej sumie czasów wykonywania były na początku;
3. Wybierz zadanie o największym $\omega(k)$ które nie zostało jeszcze uwzględnione w finalnej kolejności π^* (π^* jest na początku puste).

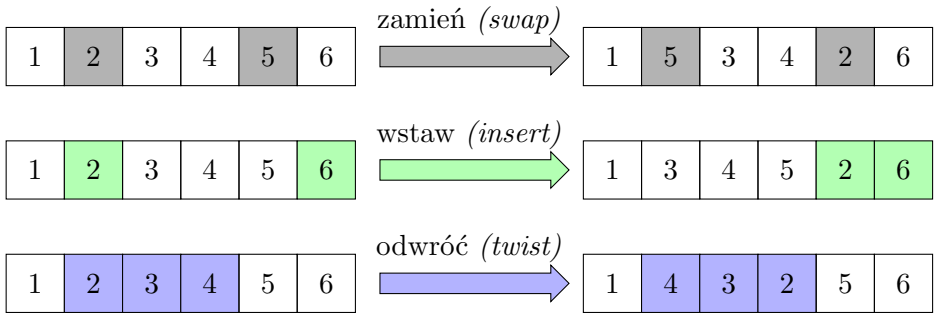
4. Wstaw wybrane zadanie k z wcześniejszego kroku metodą wstawień na wszystkie potencjalne pozycje – konstruując w ten sposób częściowe, niepełne rozwiązania π o długości o 1 element większej niż dotychczasowego π^*. Spośród nich wybierz to które ma najmniejszą wartość $C_{max}(\pi)$ jako π^*. Następnie wróć do kroku 3 o ile wszystkie zadania nie zostały już wykorzystane.

3.2.2 Algorytmy przeszukiwania lokalnego

Algorytmy przeszukiwania lokalnego (inaczej — lokalnej poprawy) należą do klasy metaheurystyk, czyli ogólnych przepisów na tworzenie algorytmów rozwiązywania problemów obliczeniowych. Pozwala to zastosować je dla szerokiej gammy odmiennych problemów. Główną strategią dla algorytmów lokalnej poprawny jest próba ulepszenia pojedynczego rozwiązania i jego sekwencyjne przekształcanie. Kolejne iteracje algorytmu tworzą swojego rodzaju „trajektorię" – złożoną z kolejno odwiedzonych przez algorytm rozwiązań – przez przestrzeń wszystkich dopuszczalnych rozwiązań. Algorytmy przeszukiwania lokalnego różnią się między sobą między innymi sposobem na opuszczanie lokalnych ekstremów, czyli rozwiązań w „pobliżu" których nie ma rozwiązań „lepszych". Sprecyzowanie poprzedniego zdania wymaga wytłumaczenia pojęć takich jak: rozwiązanie początkowe, sąsiedztwo, transformacja rozwiązania poprzez ruch – co nastąpi przy okazji opisu najprostszego algorytmu przeszukiwania lokalnego: algorytmu zstępowania.

Algorytm zstępowania

Algorytm zstępowania (dla zadania maksymalizacji – algorytm wspinaczki, ang. *hill climbing*) jest jedną z najbardziej elementarnych odmian przeszukiwania lokalnego. Algorytm rozpoczyna próbę poprawy bieżącego rozwiązania, zaczynając od rozwiązania początkowego (które stanie się rozwiązaniem obecnym dla pierwszej iteracji algorytmu). Rozwiązanie początkowe może być wybrane w różny sposób – na przykład: wygenerowane losowo, ustalone arbitralnie lub będące efektem działania innego algorytmu. Następnie w każdej iteracji budowane jest otoczenie bieżącego rozwiązania, poprzez zastosowanie na nim transformacji, zwanej ruchem (przykłady ruchów pokazano na rys. 3.1). Za każdym razem z otoczenia wybierane jest rozwiązanie o korzystniejszej wartości funkcji celu niż rozwiązanie bieżące (może to być pierwsze poprawiające rozwiązanie lub najlepsze rozwiązanie z otoczenia). To rozwiązanie staje się nowym rozwiązaniem bieżącym. Algorytm kończy swoje działanie, kiedy w otoczeniu nie znajduje się rozwiązanie o lepszej funkcji celu.



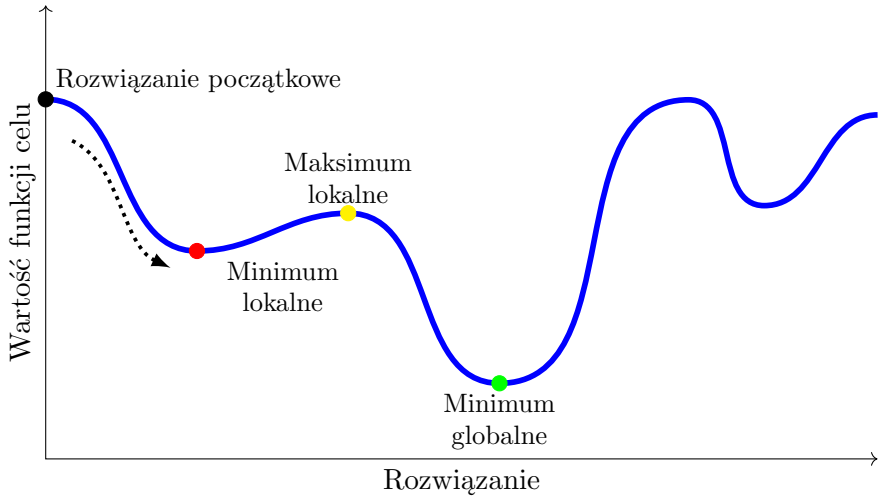
Rysunek 3.1: Często wykorzystywane transformacje rozwiązań przy generowaniu sąsiedztwa – inaczej ruchy (Rysunek inspirowany pracą [100]).

Największą wadą opisanego algorytmu przeszukiwania lokalnego jest brak możliwości wydostania się z lokalnego minimum – co dla przykładowego problemu minimalizacji prostej funkcji zostało przedstawione na rysunku 3.2. Opracowano wiele strategii rozszerzających algorytm zstępowania, mających na celu eliminację opisanej wady. Spośród tych strategii, w niniejszym rozdziale opisano szerzej symulowane wyżarzanie i przeszukiwanie z zabronieniami.

2-Opt

Algorytm zstępowania, pomimo swojej prostoty i ewidentnych ułomności, może być w niektórych zastosowaniach bardzo skuteczny. Jego odmiana – algorytm 2-Opt – jest bardzo efektywnym narzędziem rozwiązywania problemu komiwojażera z metryką euklidesową [63]. Algorytm ten opiera się na wykorzystaniu sąsiedztwa wygenerowanego za pomocą ruchu typu *twist*, co w problemie przekłada się na odwrócenie kolejności odwiedzania miast dla pewnego fragmentu trasy.

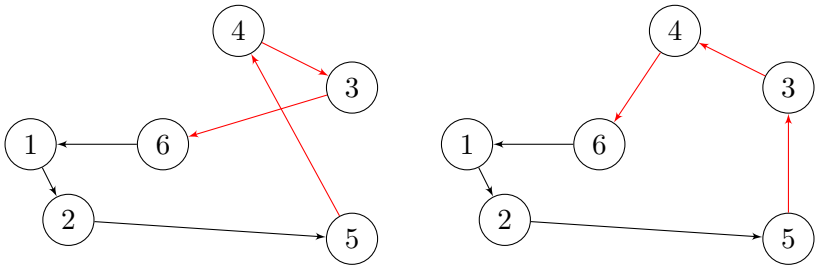
Dzięki wykorzystaniu wiedzy o problemie (dobór odpowiedniego sąsiedztwa), gwarantowany jest brak przecięć w wynikowej ścieżce – co pozwala na odrzucenie dużego zbioru nieoptymalnych tras, pomimo wykorzystania stosunkowo prostego i szybkiego algorytmu. Złożoność obliczeniowa pojedynczego kroku algorytmu 2-Opt (polegającego na wyborze z otoczenia kolejnego rozwiązania bieżącego) to $O(n^3)$, w wypadku wyznaczania wartości funkcji celu dla każdego rozwiązania z sąsiedztwa oddzielnie. Możliwe jest jednak zastosowanie akceleracji, czyli wykorzystania części wyników potrzebnych do obliczenia jednej trasy przy ocenie innej trasy z sąsiedztwa. Wtedy złożoność obliczeniowa kroku wynosi $O(n^2)$.



Rysunek 3.2: Ilustracja wyzwań związanych z przeszukiwaniem lokalnym. Po osiągnięciu minimum lokalnego, wszystkie rozwiązania w otoczeniu są gorsze od obecnego rozwiązania, co uniemożliwia jego opuszczenie. Rozważane problemy są zwykle wielowymiarowe, co jeszcze bardziej utrudnia proces przeszukiwania. (Źródło rysunku: [47])

Symulowane wyżarzanie

Algorytm symulowanego wyżarzania (ang. *Simulated Annealing*, SA) [144, 49] to wariant przeszukiwania lokalnego wyposażony w mechanizm opuszczania ekstremów lokalnych inspirowany procesem wyżarzania metali. Podczas wyżarzania, metal początkowo jest rozgrzewany do wysokich temperatur, co ułatwia jego obróbkę. Następnie jest on powoli ostudzany w celu osiągnięcia pożądanych właściwości (jak np. odpowiednia elastyczność). W SA przeszukiwanie odbywa się w podobny sposób jak w przeszukiwaniu lokalnym, jednak oprócz akceptowania rozwiązań jakościowo lepszych, czasami akceptowane są rozwiązania gorsze. Prawdopodobieństwo akceptacji takiego rozwiązania zależy od obecnej „temperatury" procesu oraz różnicy w jakości rozwiązania gorszego od obecnego rozwiązania – co jest określane jako stopień degradacji. Początkowo temperatura jest wysoka, by wraz z kolejnymi iteracjami algorytmu być stopniowo schładzana, co redykuje prawdopodobieństwo akceptacji przejścia do rozwiązania „gorszego". W literaturze [275] wykorzystywane są różne schematy obniżania temperatury w trakcie pracy algorytmu, m.in.: a) geometryczny, b) logarytmiczny, lub c) liniowy. Możliwe jest też wykorzystanie innych schematów czy modyfikacji algorytmu np. chwilowych powrotów do wyższych temperatur. Oprócz



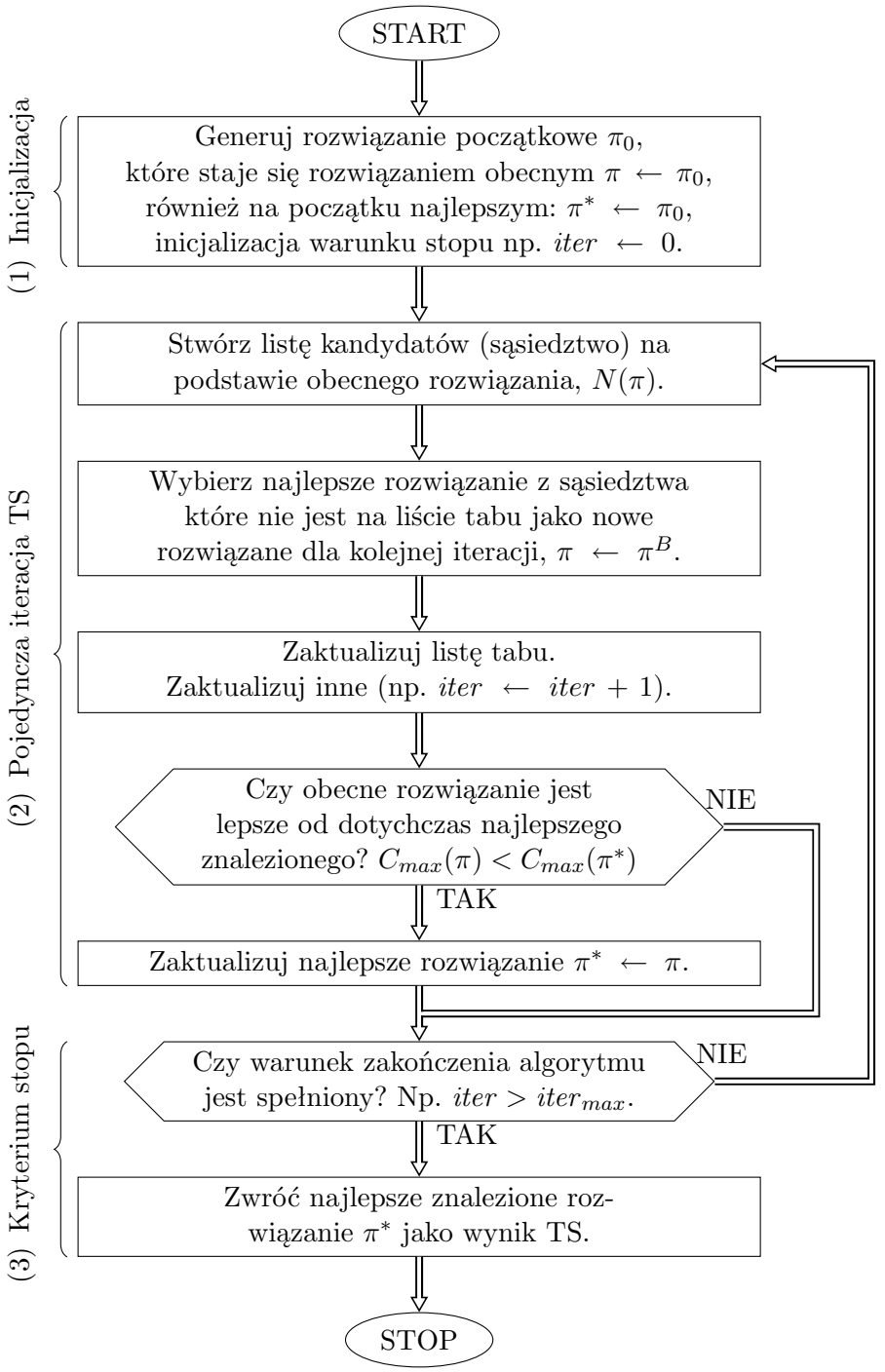
Rysunek 3.3: Przykład zastosowania ruchu typu *twist* na przykładowej trasie w problemie komiwojażera. Transformacja ta jest kluczowa w algorytmie 2-Opt. Po lewej przed transformacją – trasa jest „zaplątana", bo część drogi komiwojażera przecina się. Po prawej trasa po zastosowaniu ruchu – trasa jest „odplątana", brak przecięcia.

schematu schładzania, na efektywność algorytmu mają również wpływ inne czynniki, takie jak: wybór rozwiązania początkowego, definicja ruchu generującego nowe rozwiązania (otoczenie), warunek zakończenia przeszukiwania. Stanowi to o dużej elastyczności heurystyki, ale jednocześnie często niesie za sobą konieczność przeprowadzenia wielu eksperymentów w celu doboru *nastaw* odpowiednich dla problemu (typowa cecha metaheurystyk).

Przeszukiwanie z zabronieniami

Algorytm przeszukiwania z zabronieniami (ang. *Tabu Search*, TS) jest klasyczną metaheurystyką, algorytmem przeszukiwania lokalnego. TS jest z powodzeniem wykorzystywany do rozwiązywania wielu problemów optymalizacji dyskretnej [90, 91], w tym szeregowania zadań [29], oraz innych, takich jak problem pakowania [158]. Jako metaheurystyka, był wielokrotnie modyfikowany na przestrzeni lat (np. poprzez badania nad efektywnym generowaniem sąsiedztwa [151]) i dostosowywany do problemów jakie rozwiązywał [198]. Główną różnicą w stosunku do przeszukiwania lokalnego jest istnienie listy FIFO (z ang. *First In First Out*) zabronień (tabu). Pozwala ona pominąć część rozwiązań podczas przeszukiwania przestrzeni rozwiązań – lista ta stanowi pamięć krótkotrwałą odwiedzonych już rozwiązań.

Przykładowy schemat blokowy przeszukiwania z zabronieniami został przedstawiony na rysunku 3.4. Działanie algorytmu rozpoczyna się na schemacie od fazy inicjalizacji – wybrania startowego rozwiązania które staje się rozwiązaniem obecnym. Popularne wybory co do rozwiązania początkowego to: a) rozwiązanie losowe problemu, b) wybór w sposób arbitralny np. kolejność naturalna $(1, 2, 3, 4, \dots)$, c) rezultat działania innego algorytmu, takiego jak np. algorytm NEH (dla problemu przepływowego). Za popu-



Rysunek 3.4: Schemat blokowy dla metody przeszukiwania z zabronieniami.

larne modyfikacje tego kroku można uznać: a) multistart – czyli równoległe wykonywanie algorytmu z różnych rozwiązań początkowych, czy też b) restart – wylosowanie nowego rozwiązania w wypadku nie uzyskania poprawy w zadanej liczbie iteracji.

Następnym krokiem w algorytmie jest generacja sąsiedztwa rozwiązania obecnego – czyli przekształcenia rozwiązania obecnego poprzez zastosowanie na nim transformacji nazywanej ruchem. Z sąsiedztwa wybierane jest rozwiązanie najlepsze, które nie znajduje się na liście zabronień (liście tabu). Staje się ono nowym rozwiązaniem obecnym, a cecha ruchu który je wygenerował dodawana jest do listy zabronień. Ogranicza to możliwość powrotu do rozwiązań już odwiedzonych i jednocześnie pozwala na wydostanie się z minimum lokalnego poprzez „przejście" przez rozwiązania o gorszej wartości funkcji celu. Za typowe modyfikacje generowania otoczenia można uznać między innymi:

- modyfikacje dotyczące listy tabu, takie jak:
  - adaptacyjne wydłużanie lub skracanie listy;
  - wykorzystanie kilku list, np. jednej przechowującej atrybuty ruchów (zwykle krótszej) oraz drugiej przechowującej bezpośrednio rozwiązania (zwykle dłuższej) – za analogią do pamięci krótko i długotrwałej;
- kryterium aspiracji – przejście do rozwiązania które normalnie byłoby zakazane, stosowane szczególnie gdy rozwiązanie to byłby najlepszym dotychczas znalezionym rozwiązaniem.

Po wybraniu nowego rozwiązania obecnego, aktualizacji listy zabronień oraz potencjalnego zastąpienia dotychczas najlepszego znalezionego rozwiązania, sprawdzany jest warunek końcowy algorytmu. Popularne warunki końcowe to: a) osiągniecie zadanej liczby iteracji, b) osiągniecie zadanej liczby iteracji bez poprawy, c) przekroczenie z góry ustalonego czasu. Algorytm kończy swoje działanie poprzez zwrócenie najlepszego rozpatrywanego w dowolnej iteracji rozwiązania.

Pomimo istnienia listy tabu, w czasie wykonywania algorytmu może dojść do powstawania cykli. W tym kontekście, cykle oznaczają odwiedzania tych samych rozwiązań w sposób powtarzalny na przestrzeni kilku iteracji. Zjawisko często jest spowodowane zbyt krótką listą zabronień. W wypadku zbyt długiej listy zabronień może dojść do sytuacji kiedy wszystkie rozwiązania są zabronione. Oprócz dostosowania parametru jakim jest długość listy tabu, pomocne może być zastosowanie modyfikacji polegającą na nawrotach, czyli skokach powrotnych do wcześniej odwiedzonych rozwiązań ale ze skierowaniem przeszukiwania w inną stronę.

Jak każdą metaheurystykę, algorytm przeszukiwania z zabronieniami można dostosować do rozwiązywanego problemu, w celu uzyskania lepszych wyników. Na przykład, dla TSP szczególnie dobrym rodzajem ruchu przy generowaniu sąsiedztwa jest ruch typu *twist* (co pozwala na eliminację przecięć wyznaczanej ścieżki). Tymczasem dla problemów szeregowania częściej wykorzystywany jest ruch typu *insert*. Z kolei istotną modyfikacją generacji sąsiedztwa w kontekście szeregowania zadań było wykorzystanie własności blokowych do ograniczenia wielkości otoczenia – co przekłada się na znacznie przyspieszenie działania całego algorytmu [198, 199].

3.2.3 Algorytmy populacyjne

Algorytmy populacyjne opierają swoje działanie na stopniowej zmianie rozwiązań problemu reprezentowanych w ramach populacji. Proces ten inspirowany jest zarówno ewolucją biologiczną (w tym mechanizmami doboru naturalnego), jak i czerpie inspirację z zachowań stadnych gatunków zwierząt. Mnogość tego drugiego podejścia została poruszona w sekcji „Trendy w algorytmach populacyjnych".

Algorytmy ewolucyjne

Algorytm ewolucyjne bazują na populacji, czyli zbiorze osobników (zakodowanych rozwiązań problemu). Osobniki na przestrzeni iteracji poddawane są zróżnicowanym procesom inspirowanym biologicznie, między innymi: mutacji, reprodukcji, rekombinacji czy selekcji. Celem tych procesów jest stopniowa poprawa przystosowania populacji, czyli aby składała się ona z coraz „lepszych" osobników.

Z uwagi na bardzo szeroką i relatywnie nieformalną definicję, trudno jasno określić ramy stosowania pojęcia „algorytmy ewolucyjne". W szczególności, często dany algorytm może być jednocześnie przypisany do kilku kategorii, w zależności od tego na jaki jego element kładziony jest nacisk. Niemniej, za [253, 185], można przyjąć podział algorytmów ewolucyjnych na między innymi:

- Algorytmy genetyczne (ang. *Genetic Algorithms*) – bardzo popularne i typowo kojarzone z reprezentacją rozwiązania w postaci ciągu binarnego oraz wykorzystaniem takich operatorów genetycznych jak krzyżowanie oraz mutacja (podejście pochodzi z badań Holland'a [113]).
- Strategie ewolucyjne (ang. *Evolution Strategies*), w których wykorzystuje się reprezentację rozwiązań dostosowaną do rozwiązywanego problemu ze specjalnymi procedurami przeszukiwania dostosowanymi do optymalizacji numerycznej (nie ograniczonymi przez analogię do

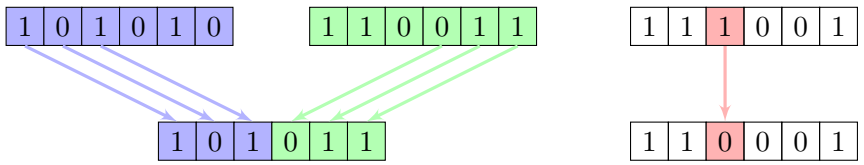
biologicznej ewolucji). Podejście pochodzi od badań Schwefel'a [243, 244], gdzie wykorzystywano reprezentację rozwiązania w postaci pary zmiennopozycyjnych wektorów oraz mutację jako jedyny operator służący rekombinacji populacji, czy raczej pojedynczego osobnika w wypadku pierwotnych badań.

- Programowanie genetyczne (ang. *Genetic Programming*), zamiast reprezentować rozwiązanie problemu w postaci osobnika, podejście to opiera się na populacji programów komputerowych które są ocenianie w ramach dopasowania do rozwiązywanego zadania. Programy w populacji są budowane jako wyrażenia matematyczne z wykorzystaniem struktur takich jak drzewa binarne czy grafy skierowane. Podejście to zapoczątkował Koza w badaniach [154, 155].

Podział ten nie jest jednak ostry, cytując [70] za [185]: *Zauważyłem kiwanie głowami nad systemem innych badaczy zajmujących się algorytmami genetycznymi [...] i szczere zdumienie, że system, który opracowaliśmy, jest algorytmem genetycznym (gdyż nie używaliśmy reprezentacji binarnej, binarnego krzyżowania i binarnej mutacji).* Źródło [185] sugeruje sposób radzenia sobie z niejasną klasyfikacją: *Możemy dodatkowo zapytać na przykład, czy strategia ewolucyjna jest algorytmem genetycznym? A czy odwrotne pytanie jest prawdziwe? Aby uniknąć wszystkich spraw związanych z klasyfikacją systemów ewolucyjnych, nazywamy je po porostu „programami ewolucyjnymi"*. Przy czym źródło to przypisywało terminowi „programu ewolucyjnego" szerokie znacznie, opisując w ten sposób wszystkie systemy korzystających z zasad ewolucji.

Podobnie jak w przypadku innych algorytmów metaheurystycznych, również dla algorytmów ewolucyjnych niezbędne jest dostosowanie techniki do rozwiązywanego problemu. Na przykład, w celu stworzenia algorytmu genetycznego – cytując za [185] – trzeba zdecydować m.in. o:

- Sposobie reprezentacji rozwiązania w postaci osobnika, czyli kodowaniu rozwiązania problemu w postaci ciągu binarnego.
- Metodzie wyboru populacji początkowej.
- Wyborze funkcji oceniającej przystosowanie osobnika do środowiska, np. można bezpośrednio zastosować funkcję celu problemu.
- Wyborze operatorów genetycznych które zostaną wykorzystane w modelowaniu procesu ewolucji.
- Ustaleniu hiperparametrów algorytmu, np. liczby osobników w populacji, sposobie wyboru osobników do kolejnych iteracji (np. zgodnie z zasadą ruletki, rodzice wybrani do krzyżowania są losowani, z uwzględnieniem jakości rozwiązania które reprezentują – większa



(a) Operator krzyżowania (b) Operator mutacji

Rysunek 3.5: Przykład zastosowania wybranych operatorów genetycznych. Po lewej operator krzyżowanie dla pojedynczego potomka z punktem przecięcia genów w połowie ciągu binarnego, potencjalny kolejny potomek mógłby składać się z pozostałych części nie wykorzystanych na rysunku. Po prawej operator mutacji dla pojedynczego genu.

szansa wylosowania dla lepiej przystosowanych), wyborze warunku stopu algorytmu czy też ustaleniu prawdopodobieństwa mutacji.

Za [253], podstawowe, klasyczne operatory genetyczne to krzyżowanie i mutacja. Mutacja polega na zamianie wartości znaku bitowego genu na przeciwny. Ta wyjątkowo prosta zmiana służy dostarczeniu nowej informacji do populacji – bez operatora mutacji, populacja stawałaby się coraz bardziej homogeniczna na przestrzeni iteracji. Tym samym, mutacja pomaga w eksploracji przestrzeni rozwiązań i potencjalnie pozwala na ucieczkę z lokalnego optimum. Współczynnik szansy wystąpienia mutacji jest hiperparametrem metody który należy dostosować do rozwiązywanego problemu. Jednakże, za [253], w większości implementacji mutacja jest rzadka – często przyjmuje się prawdopodobieństwa rzędu 1–2%. Przykład operatora mutacji został pokazany na rysunku 3.5b.

Krzyżowanie jest operatorem który umożliwia wykorzystanie części informacji zawartej w każdym z rodziców i przekazanie jej do potomków. Często wykorzystywana implementacja polega na wylosowaniu punku przecięcia w ramach ciągów binarnego rodziców. Przecięcie takie dzieli ciąg binarny na dwa podciągi, potomek tworzony jest poprzez wykorzystanie po jednej z części od każdego z rodziców. Przykład dla punktu przecięcia znajdującego się w połowie ciągu binarnego został przedstawiony na rysunku 3.5a. Krzyżowanie osobników reprezentujących „dobre” rozwiązania służy stworzeniu potomstwa które będzie znajdowało się coraz bliżej optimum lokalnego. Właściwy balans pomiędzy krzyżowaniem oraz mutacją stanowi swojego rodzaju kompromis pomiędzy szybkim zbieganiem do lokalnego optimum, a poszukiwaniem nowych i obiecujących obszarów przestrzeni rozwiązań. Jedną z zalet rozmnażania płciowego, którego uproszczoną wersją jest operator krzyżowania, jest pozyskanie nowych kombinacji genów.

Ma ona na celu zwiększenia różnorodności osobników wewnątrz populacji. To bezpośrednio prowadzi do większych szans przetrwania grupy, poprzez zwiększenia jej odporności na niekorzystne lub wręcz śmiertelnych zmiany warunków środowiskowych[3]. W ramach optymalizacji pozytywny wpływ zróżnicowania populacji dla algorytmów ewolucyjnych został opisany między innymi w pracach [208, 296]. Dodatkowo, zróżnicowanie populacji ma pozytywny wpływ na metrykę „zdolności do ewolucji" opisaną w sekcji 5.2.5. Stan wiedzy o wykorzystaniu algorytmów ewolucyjnych do rozwiązywania problemów optymalizacji dyskretnej można znaleźć w pracy [77].

Algorytm optymalizacji za pomocą roju cząstek

Głównym założeniem algorytmów optymalizacji za pomocą roju cząstek (ang. *Particle Swarm Optimization*, PSO) [140], jest wymiana „wiedzy" pomiędzy cząsteczkami (rozwiązaniami problemu), tworzącymi wspólnie większy byt – rój. Każde rozwiązanie jest reprezentowane poprzez wektor położenia i prędkości w przestrzeni wielowymiarowej. Prędkość cząsteczki zmieniana jest na przestrzeni iteracji algorytmu poprzez ważoną sumę:

- wcześniejszej prędkości tej cząsteczki (ważoną z uwzględnieniem współczynnika bezwładności);
- wektora informacji o dotychczas najlepszym znalezionym globalnie rozwiązaniu, przez którąkolwiek z cząsteczek (ważony o współczynnik dążenia do globalnego rozwiązania);
- wektora informacji o dotychczas najlepszym rozwiązaniu znalezionym przez tą cząsteczkę (ważonym o współczynnik dążenia do „lokalnego" rozwiązania).

Waga każdego z wymienionych współczynników wpływa znacząco na działanie algorytmu i umożliwia położenie większego nacisku na wykorzystanie zdobytej wiedzy lub na eksplorację środowiska w celu jej pozyskania (ang. *exploration and exploitation dilemma*). Algorytm jest ciekawy ze względu na łatwość implementacji dedykowanej na platformy wieloprocesorowe i rozproszone oraz brak zależności od gradientu funkcji optymalizowanej. Co za tym idzie, PSO może sprawdzić się tam gdzie metody oparte o gradient nie mogłyby zostać zastosowane.

[3] Jako przykład katastrofalnych skutków niedostatecznej różnorodności genetycznej można przytoczyć los bananów z gatunku *Gros Michel*, które do granic wymarcia doprowadziła grzybiczna choroba *Panama disease*.

Algorytm kolonii pszczół

Algorytm kolonii pszczół (ang. *Artificial Bee Colony*) [136] jest algorytmem wzorowanym na sposobie zbierania nektaru przez pszczoły. W celu maksymalizacji zebranych zasobów, pszczoły są wyspecjalizowane w określonych zadaniach: poszukiwaniu nowych źródeł nektaru, wykonywaniu „tańca" w celu promocji obiecujących obszarów, czy też po prostu zbieraniu samego nektaru. Algorytm opiera się o populację składającą się z określonej liczby rozwiązań (początkowo losowych). W każdej iteracji pierwszym krokiem jest generacja dla każdego rozwiązania sąsiada. Jeśli jest on lepszy, zastępuje rozwiązanie pierwotne z populacji. Następnym krokiem jest podobny proces, jednak tym razem nie wykonywany dla każdego rozwiązania osobno. Zamiast tego generowane jest losowo wiele sąsiadów rozwiązań, przy czym rozwiązania o wyższej jakości mają większą szansę na wygenerowanie sąsiada. Oznacza to, że dla obiecujących rozwiązań rozważane jest potencjalnie więcej rozwiązań sąsiednich. Ostatnim krokiem jest zastępowanie rozwiązaniem losowym, tych osobników z populacji, dla których nie dokonano poprawy. Wykorzystuje się w tym celu prosty licznik rozważanych rozwiązań sąsiednich, zwiększany niezależnie od kroku algorytmu, a resetowany przy zastępowaniu rozwiązania w kolonii przez inne.

Trendy w algorytmach populacyjnych

Za pewną tendencje można uznać próby stworzenia nowych algorytmów populacyjnych na podstawie procesów występujących w naturze. Algorytmy takie są inspirowane między innymi stadnymi zachowaniami zwierząt (owadów, ssaków, ryb) ale również zjawiskami przyrodniczymi, zachowaniami politycznymi czy też zjawiskami socjologicznymi. Brak jednoznacznej i uniwersalnej metody ewaluacji nowych, arbitralnie konstruowanych a „inspirowanych naturą" algorytmów rodzi ryzyko tworzenia algorytmów dla samego procesu ich tworzenia, w pogoni za pseudo-innowacyjnością. Krytykę tego zjawiska, wraz z próbą jego kategoryzacji, można znaleźć w pozycjach [173, 260][4]. Duża część tego typu algorytmów może zostać uznana za odmianę algorytmów ewolucyjnych z „nowym", czy też po prostu „(bio)-inspirowanym", sposobem transformacji rozwiązań, z opcjonalną metodą poprawy w każdej iteracji algorytmu.

Pomimo przytoczonej polemiki, należy podkreślić, że część „inspirowanych" algorytmów wydaje się obiecująca dla rozwiązywania problemów

[4]Przykładem aktywnie utrzymywanego spisu algorytmów, czy jak określają go autorzy – ze względu na liczbę pozycji – „bestiariusza", jest projekt na platformie *GitHub* o nazwie *Evolutionary Computation Bestiary*: https://github.com/fcampelo/EC-Bestiary; dostępny na dzień 13.04.2022

optymalizacji. Za przykład takiego algorytmu można uznać omówiony wcześniej algorytm inspirowany życiem kolonii pszczół. Ciągle jednak wyzwaniem pozostaje metoda ewaluacji algorytmu. Krytyczna analiza wymagałaby między innymi zbadania na ile stopień skomplikowania pojedynczej iteracji przedkłada się na efektywność algorytmu, szczególnie w porównaniu do prostszych algorytmów populacyjnych. Taka analiza mogłaby odbywać się na poziomie teoretycznym lub numerycznym, w odniesieniu do zróżnicowanego portfolio problemów.

3.3 Wnioski i uwagi

Rozdział zawiera przegląd oraz kategoryzację często wykorzystywanych metod optymalizacji dyskretnej. Opracowanie nie jest wyczerpujące i zostało ukierunkowane na algorytmy wykorzystywane w badaniach przedstawionych w kolejnych rozdziałach pracy. Przykładem może być przeszukiwanie z zabronieniami, którego modyfikacja została wykorzystana w badaniach przedstawionych w rozdziale 8, czy też metoda 2-Opt z powodzeniem stosowana jako sposób generowania rozwiązania początkowego przed zastosowaniem bardziej złożonych algorytmów dla TSP.

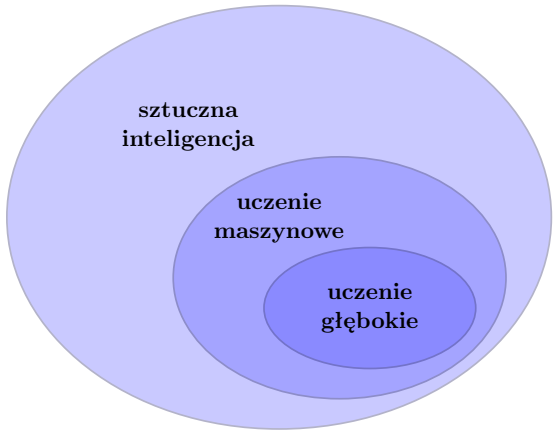
Rozdział 4

Sieci neuronowe

Rozdział stanowi syntetyczne omówienie wybranych zagadnień związanych ze sztucznymi sieciami neuronowymi. Został podzielony na trzy części. W pierwszej z nich omówione zostały niezbędne, powiązane pojęcia, takie jak sztuczna inteligencja czy uczenie maszynowe. Następnie omówiono zarówno klasyczne sieci neuronowe, wraz z uwagami co do ich uczenia, jak i podejście współczesne oparte o głębokie uczenie. Na końcu omówione zostały wybrane pojęcia związane z podejściem bazującym na uczeniu ze wzmocnieniem. Poruszona tematyka jest bardzo szeroka – przez co rozdział ten stanowi raczej przedstawienie najważniejszych terminów, które dopiero można pogłębić poprzez materiały źródłowe: w tym podręczniki akademickie [23, 94, 233, 266, 314], książki specjalistyczne [56, 78, 189] czy materiały popularyzujące tematykę sieci neuronowych i sztucznej inteligencji [121, 162, 268].

4.1 Podstawowe pojęcia

Przed przejściem do opisu klasycznych sieci neuronowych, należy przybliżyć istotne pojęcia pokrewne. W tym celu opisano dziedzinę do jakiej należą sieci neuronowe, czyli sztuczną inteligencję, wraz z uczeniem maszynowym, czyli grupą metod do której sieci neuronowe są przyporządkowane. Zakres oraz wzajemną zależność pomiędzy pojęciami sztuczna inteligencja, uczenie maszynowe oraz uczenie głębokie została przedstawiona na ideowym schemacie rysunku 4.1.



Rysunek 4.1: Schematyczny zakres pojęć: sztuczna inteligencja, uczenie maszynowe oraz uczenie głębokie – rysunek na podstawie źródła: [56].

4.1.1 Sztuczna inteligencja

Pojęcie sztucznej inteligencji (AI[1], od ang. *Artificial Intelligence*) powstało w latach 50. XX wieku. Jedna z definicji określa ją jako: „dziedzinę informatyki zajmującą się tworzeniem algorytmów i metod, które naśladują zachowania tradycyjnie kojarzone z ludzką inteligencją, pozwalając rozwiązywać zadania i problemy z nią kojarzone”[2]. Choć istnieją pewne zastrzeżenia wobec słowa „sztuczna”, to jednak dość dobrze oddaje ono „nie-biologiczną” genezę tych metod, w tym programów komputerowych oraz implementacji elektronicznych.

Bardziej interesujący wydaje się komponent „inteligencja”, szczególnie w wypadku odwołań do ludzkiej inteligencji. Za [263] można prześledzić

[1]Polski skrót sztuczna inteligencja: „SI” będzie z premedytacją ignorowany w ramach tej dysertacji, po części ze względu na zbieżność do skrótu od *Międzynarodowego Układu Jednostek Miar*.

[2]Inne definicje sztucznej inteligencji – za [233]: *attempts not just to understand but also to build intelligent entities* [...] *inteligence is concerned mainly with rational action. Ideally, an intelligent agent takes the best possible action in a situation.*, za [56]: *the effort to automate intellectual tasks normally performed by humans*, za [189]: *branch of computer science involved in the creation of computer programs capable of demonstrating intelligence.* Źródło [94] unika generalnej definicji podając podział na podejście klasyczne: *in the early days of AI* [...] *tackled and solved problems that are intellectually difficult for human beings but relatively straightforward for computers – problems that can be described by a list of formal, mathematical rules* oraz obecne wyzwania: *The true challenge* [...] *to be solving the tasks that are easy for people to perform but hard for people to describe formally – problems that we solve intuitively, that feel automatic, like recognizing spoken words or faces in images.*

psychologiczne podejście do pojęcia inteligencji. Za początki współczesnego rozumienia inteligencji można przedstawić badania Bineta i Simona z 1904 roku, które miały konkretny cel: znalezienie sposobu na ustalenie, które z dzieci podczas nauki szkolnej potrzebują dodatkowej uwagi, by następnie zapewnić im pomoc. Ten szczytny, bardzo konkretny cel na przestrzeni lat został niestety tragicznie wypaczony przez ruchy eugeniczne, doprowadzając do ludzkiego cierpienia [165]. Podejście Bineta i Simona reprezentuje nurt psychometryczny – w ramach którego inteligencję traktuje się jako cechę różnicującą ludzi (podobnie jak np. wzrost), wyzwaniem staje się wiec skuteczny jej pomiar oraz określenie jej wartości w kontekście jednostki; przy jednoczesnym odwzorowaniu zróżnicowania w populacji. Jest to rozumienie inteligencji zupełnie inne niż wykorzystywane w ramach tworzenia sztucznej inteligencji – w tym wypadku bardziej pomocne może być podejście poznawcze skoncentrowane na poznaniu procesów umysłowych, wraz z ustaleniem ogólnych prawidłowości w trakcie wykonywania zadań umysłowych (rozwiązywania problemów).

Za [263] możemy podać współczesne pojęcie inteligencji: *inteligencja to konstrukt teoretyczny odnoszący się do względnie stałych warunków wewnętrznych człowieka, determinujących efektywność wykonywania zadań lub rozwiązywanych problemów wymagających typowo ludzkich procesów poznawczych, takich jak wnioskowania, rozumowanie, planowanie itp. Warunki te kształtują się w ontogenezie w wyniku specyficznego dla jednostki oddziaływania między genotypem a środowiskiem.* Definicja ta została przywołana ze względu na zobrazowanie licznych braków współczesnego rozumienia inteligencji, samo wyjaśnienie z [263] wprost mówi o stosunkowo ubogim stanie wiedzy, utrzymującym się mimo licznych badań, niepozwalającym jednoznacznie wskazać co determinuje zachowanie które uznajemy za inteligentne. To pokazuje niejasność samej istoty inteligencji, która w połączeniu z brakiem zgody co do jednej jej definicji na przestrzeni lat[3] obrazuje na jak „grząskim" gruncie rozwijane są badania nad sztuczną inteligencją. W badaniach tych często ignoruje się brak jasności generalnego pojęcia inteligencji, decydując się na replikację przejawów ludzkiej inteligencji jedynie w wybranych, ściśle sprecyzowanych, niepołączonych ze sobą obszarach.

[3]Za [263], w 1921 roku podczas sympozjum w celu ustalenia czym jest inteligencja, wśród jej badaczy uzyskano bardzo różne definicje, jak np.: a) *zdolność do udzielenia dobrych odpowiedzi z punktu widzenia prawdy lub istniejących faktów*; b) *zdolność przejawiająca się w myśleniu abstrakcyjnym*; c) *możliwości, których tylko część mierzona jest testami inteligencji*; d)*zdolność wyrażająca stopień nauczenia się lub możliwości nauczenia się adaptacji do własnego środowiska*; e) *zdolność jednostki do adekwatnej adaptacji do względnie nowych sytuacji życiowych*; f) *efektywność wykonywania zadań szczególnie typowych dla sytuacji szkolnych*; g) *obecny stan wiedzy nie pozwala na sformułowanie definicji inteligencji.*

Podejście replikujące inteligencje w taki właśnie ograniczony sposób, w ramach pojedynczych problemów, nazywa się „słabą sztuczną inteligencją" (ang. *weak AI*, ale również ang. *narrow AI*[4]). Przeciwieństwo w postaci sztucznej inteligencji zdolnej w pełni replikować ludzką inteligencję określa się jako „silną sztuczną inteligencję" (ang. *strong AI*, ale również ang. *Artificial General Intelligence*, AGI). Pomimo swoistego statusu Grala badań nad AI [189], ciągle pozostaje nieosiągniętym celem – co nie oznacza, że nie są prowadzone badania, aby stworzyć tego typu technologię (czy właściwie sztuczny byt?). Konsekwencje stworzenia takiej technologii na współczesny świat autor dysertacji pozostawia futurologom (w tym autorom książek takich jak: [162, 164]), choć z drobną uwagą o niezmiernej ciekawości dotyczącej tego, jak wiele lat badań jest jeszcze potrzebnych do osiągnięcia punktu przełomowego. Za taki punkt autor rozumie następny spodziewany krok po stworzenia AGI – osobliwość, jaką jest samo-poprawiająca się silna sztuczna inteligencja.

Kolejnym przykładem dążenia do rozwijania „szerokiego" podejścia do sztucznej inteligencji jest test, zaproponowany przez matematyka Alana Turinga w 1950 w celu zoperacjonalizowania pojęcia inteligencji – w tym zastąpienie trudnego, wręcz filozoficznego pytania: *czy maszyna może myśleć?* Komputer zda ten test, jeśli podczas pisemnej rozmowy z ludzkim sędzią ten nie będzie w stanie, na podstawie odpowiedzi na zadane pytania, odróżnić, czy rozmawia z maszyną, czy z innym człowiekiem. Jak podkreśla źródło [233], by przejść taki test, program musi wykazać się „szerokimi" umiejętnościami, w tym: a) zdolnością do przetwarzania języka naturalnego, aby komunikować się z rozmówcą; b) umiejętnością reprezentacji zgromadzonej wiedzy, aby przechowywać uzyskane w rozmowie informacje; c) automatycznym rozumowaniem w celu odpowiedzi na pytania, ale również wyciągania wniosków; d) uczeniem maszynowym w celu adaptacji do nowych okoliczności, w tym ekstrapolację wzorców zaobserwowanych w trakcie rozmowy. W tym kontekście można podać przykład modelu LaMDA ([280], od ang. *Language Model for Dialogue Applications*) firmy Google z roku 2022 (jak i kontrowersji wokół niego), który obrazuje jak bardzo zbliżyć można się do wrażenia „świadomego rozmówcy", pod warunkiem zastosowania odpowiednich środków – w tym korpusu językowego złożonego z ponad 1,56 bilionów słów! Fakt sukcesów oraz coraz powszechniejszego wykorzystaniem ogromnych modeli językowych (ang. *Large Language Models*, LLM), takich jak przytoczony model LaMDA lub popularny ChatGPT, prowadzi nawet do pytań, czy modele tego typu nie są już bardzo wczesnymi i niekomplet-

[4]Można spotkać tłumaczenie „wąska sztuczna inteligencja", choć lepiej oddawałby sendo: „ograniczona (do pojedynczego problemu) sztuczna inteligencja".

nymi „protoplastami" AGI [46].

Choć historia powstawania sztucznej inteligencji jako osobnej dziedziny nauki jest fascynująca (w tym początkowy okres z lat 1943–1969, przed pierwszą „zimą AI"[5]) – nie zostanie ona prześledzona w ramach dysertacji. Zamiast tego czytelnicy zostaną odesłani do znacznie lepszych, już wspomnianych pozycji – takich jak [94, 233, 268]. Pewnym odstępstwem od wspomnianej decyzji ograniczenia zakresu rysu historycznego będą przykłady służące podkreśleniu, że sztuczna inteligencja obejmowała podejście mocno interdyscyplinarne, które nie wpasowywało się w ograniczenia innych dziedzin nauki. Do takich przykładów należą: a) badania nad metodami obliczeń koniecznych do modelowania biologicznych sieci neuronowych Marvin'a Minsky'ego, jednego z założycieli *Massachusetts Institute of Technology* – które przyjęto z dozą sceptycyzmu, czy można uznać je za należące do dziedziny matematyki; b) hipoteza dotycząca uczenia poprzez wzmacnianie połączeń pomiędzy pobudzonymi równocześnie neuronami (oparta o neuroplastyczność), która została sformułowana przez psychologa Donalda Hebba; c) badania Franka Rosenblata (również psychologa), który stworzył fizyczny model sieci neuronowej do analizy obrazu – *perceptron* – w postaci układu elektronicznego.

4.1.2 Uczenie maszynowe

Uczenie maszynowe to jeden z obszarów sztucznej inteligencji. Cytując definicję zaproponowaną w [314]: *Uczenie maszynowe to zbiór technik pozwalających na poprawę wydajności systemu poprzez trening bazujący na wcześniejszych doświadczeniach z wykorzystaniem metod obliczeniowych. W ramach systemów komputerowych przeszłe doświadczenia są reprezentowane w postaci danych, przez co głównym celem uczenia maszynowego staje się opracowanie algorytmów uczenia pozwalających na stworzenie modeli z wykorzystaniem danych. Poprzez wykorzystanie przykładów ze zbioru treningowego możliwe jest otrzymanie modeli, które następnie można wykorzystać w celu otrzymania wyników na nowych, wcześniej niedostarczonych w ramach treningu danych. Jeśli określimy informatykę za podgrupę algorytmiki, to podobny podział można zastosować do uczenia maszynowego jako podgrupy algorytmów opartych o proces treningu.*

[5] Za początek jednej z „zim AI" przyjmuje się okres publikacji w 1969 roku książki *Perceptrons: an introduction to computational geometry*, której autorami są M. Minsky i S. Papert. Choć zawierała jedynie matematyczny dowód ograniczenia modeli opartych o funkcje liniowe, wyznacza okres zmniejszonego zainteresowania środowiska tematyką sieci neuronowych, co było połączone również z ograniczeniem funduszów badawczych (jak np. zmiana podejścia *Defense Advanced Research Projects Agency* w USA, stawiającej na bezpośrednią użyteczność projektów, której w tamtym czasie brakowało AI.)

Rysunek 4.2: Uproszczona różnica między podejściem reprezentującym klasyczne programowanie, a uczeniem maszynowym (należy jednak zaznaczyć, że istnieją inne ważne cechy np. reguły nie muszą być jawne w wypadku uczenia maszynowego – rysunek na podstawie książki [56]).

Przewodnik techniczny F. Cholleta [56] przedstawia uczenie maszynowe jako nowy paradygmat techniki programowania, podkreślając różnicę pomiędzy uczeniem maszynowym, a klasycznym programowaniem. W programowaniu klasycznym programista wprowadza reguły algorytmu w postaci programu komputerowego wraz z danymi, na których taki program ma działać (dane wejściowe), aby uzyskać oczekiwane rezultaty (dane wyjściowe). W ramach uczenia maszynowego wprowadza się zarówno dane wejściowe, jak i odpowiadające im oczekiwane rezultaty (dane wyjściowe) – natomiast reguły, które je łączą, stanowią wynik działania algorytmu uczenia maszynowego. Tak otrzymane reguły (w postaci modelu) można następnie wykorzystać w celu przetworzenia nowych, wcześniej niewykorzystanych danych. Różnice pomiędzy tymi paradygmatami zostały przedstawione na rysunku 4.2. Sam system uczenia maszynowego za [56] jest trenowany poprzez dostarczenie wielu przykładów, co pozwala wykorzystać ich statystyczną strukturę, aby ustalić reguły umożliwiające automatyzację procesu. Brak jawnego programowania systemu wiąże się z koniecznością odpowiedniego przygotowania danych – poprzez przygotowanie przykładów złożonych z par: dane wejściowe, wraz z odpowiadającymi im oczekiwanymi danymi wyjściowymi z systemu. Przykładowo do przygotowania nowego zbioru treningowego w zadaniu klasyfikacji obrazów zawierających określone gatunki zwierząt, konieczne jest manualne i dość żmudne etykietowanie występujących gatunków zwierząt na każdym ze zdjęć w ramach zbioru treningowego.

Choć w kontekście dysertacji sieci neuronowe są kluczową metodą uczenia maszynowego której poświęcono najwięcej uwagi, to należy podkreślić, że w ramach uczenia maszynowego można wyróżnić wiele podejść i metod służących między innymi do: klasyfikacji, predykcji, regresji, grupowania,

redukcji wymiarów czy wykrywania anomalii. Aby wymienić choć małą część z popularnych metod, można przywołać za [314]: drzewa decyzyjne (ang. *Decision Trees*), metodę regresji liniowej (ang. *Linear Regression*), regresji logistycznej (ang. *Logistic Regression*), naiwny klasyfikator bayesowski (ang. *Naive Bayes*), metodę maszyn wektorów nośnych (ang. *Support Vector Machine*), lasy losowe (ang. *Random Forest*), metodę klasteryzacji algorytm k-najbliższych sąsiadów (ang. *k-nearest Neighbors Algorithm*), metodę analizy głównych składowych (ang. *PCA* od *Principal Component Analysis*), czy metodę analizy danych wielowymiarwych t-SNE [287] (ang. *t-distributed Stochastic Neighbor Embedding*).

Na podstawie wspomnianej juz książki [56], warto przybliżyć możliwości algorytmu wzmacniania gradientowego[6] [52] (ang. *Extreme Gradient Boosting*). Metoda ta opiera się na łączeniu wielu słabych klasyfikatorów – drzew decyzyjnych – ale dodatkowo o: a) iteracyjne podejście do treningu nowych modeli, tak aby poprawić wyniki na przykładach problematycznych dla modeli z wcześniejszych iteracji; oraz o b) zastosowanie metod regularyzacji, co pozwala zapobiec zbytniemu przetrenowaniu, a jednocześnie nakłada karę za zbyt dużą złożoność modelu (w tym: za dużą liczbę liści w drzewie decyzyjnym). Wykorzystanie algorytmu wzmacniania gradientowego pozwala zwykle uzyskać lepsze rezultaty zarówno od metody lasów losowych jako i sieci neuronowych na danych niepercepcyjnych[7]. Algorytm ten pozostaje popularny w konkursach organizowanych na platformie `Kaggle` (w latach 2016–2022). Jednocześnie stanowi świetny przykład konieczności doboru odpowiedniej metody uczenia maszynowego do rozwiązywanego problemu – nie zawsze modele skomplikowane, takie jak współczesne sieci neuronowe, są odpowiednim narzędziem.

Rodzaje uczenia maszynowego

Istnieje co najmniej kilka rodzajów metod pozyskania informacji z danych, wykorzystując sieci neuronowe oraz inne metody uczenia maszynowego. Trzy główne, często wymieniane (szczególnie w kontekście podziału metod opartych o sztuczne sieci neuronowe), to:

- Uczenie nadzorowane (ang. *Supervised Learning*) – w podejściu tym znane są dane wejściowe, jak i odpowiadające im dane oczekiwane

[6] Algorytm dostępny w ramach biblioteki `https://xgboost.readthedocs.io/`, dostęp dnia 02.01.2023

[7] Chodzi o skuteczność metody na danych tabelarycznych, szczególnie o ograniczonej liczbie cech – które można przedstawić właśnie w postaci tabeli. Tego typu dane różnią się od danych charakterystycznych dla sukcesów sieci neuronowych: zapisów języka naturalnego lub danych wizyjnych, w tym zarówno obrazu, jak i wideo.

na wyjściu modelu. Do zebrania tego typu danych w większości przypadków potrzebny jest udział człowieka – bardzo często w postaci tworzenia adnotacji (przykładowo: ręczne oznaczanie budynków na zdjęciach satelitarnych, czyli dostarczenie oczekiwanego wyjścia dla modelu). Podejściu temu sprzyja powszechność coraz większych zbiorów danych. W ramach procesu uczenia sieć neuronowa działa jako przekształcenie danych wejściowych w dane wyjściowe, na podstawie dostarczonych par danych (zbioru treningowego). Podejście to kładzie nacisk na jakość funkcji odwzorowującej – konieczne jest nadzorowanie jakości generalizacji, tak aby nauczona sieć działała również dobrze na nowych danych spoza zbioru treningowego. Podejście to jest wykorzystywane powszechnie w klasyfikacji czy regresji – szczególnym sukcesem okazały się sieci konwolucyjne, pozwalające na wręcz „nadludzkie" sukcesy w wizualnym rozpoznawaniu obiektów (np. rozpoznawaniu ręcznego pisma czy segmentacji semantycznej obrazów – dla niektórych zbiorów danych, uzyskane wyniki są lepsze, niż uzyskane przez ludzi).

- Uczenie ze wzmocnieniem (ang. *Reinforcement Learning*) – dane wejściowe są znane, jednak zamiast danych wyjściowych, dostępna jest jedynie informacja czy generowane przez sieć wyjście jest pożądane, czy nie (w postaci nagrody lub kary za podejmowane przez model decyzje). Zwykle w podejściu tym sieć stanowi część modelu agenta, który eksploruje nieznane środowisko z zadaniem maksymalizacji zebranej nagrody na przestrzeni zadanej liczby iteracji (model reprezentuje więc „strategię" działania w środowisku). W podejściu tym ważny jest kompromis pomiędzy zbieraniem nowych danych, a wykorzystywaniem już pozyskanej wiedzy (ang. *exploration vs. exploitation tradeoff*). Sieć neuronowa jako komponent agenta musi jednocześnie działać w środowisku oraz uczyć się na podstawie popełnianych błędów. Sukcesy w tym obszarze to pokonywanie coraz bardziej skomplikowanych środowisk, w tym kolejnych gier: szachów, Go [252] czy zaawansowanych współczesnych gier, takich jak Starcraft [293] (niepełna informacja o stanie „planszy"), czy drużynowa DOTA [21] (wpływ innych agentów na stan gry).
- Uczenie nienadzorowane (ang. *Unsupervised Learning*) – znane są jedynie dane wejściowe, brak informacji o oczekiwanym wyjściu. Podejście to jest wykorzystywane do odkrywania wcześniej nieznanych wzorów, wstępnej analizy danych, czy też redukcji wymiarowości danych. Do podejścia tego można zaliczyć chociażby klasyczny algorytm klasteryzacji k-średnich. Choć z zasady podejście to prowadzi do odkrycia nieoczywistych wzorców czy zależności, to końcowa interpreta-

cja tak przetworzonych danych wymaga najczęściej interwencji człowieka. Do nieoczywistych, „nowych" sukcesów w tym obszarze można zaliczyć odwzorowywanie odległości pomiędzy słowami w przestrzeni wielowymiarowej, na podstawie korpusów tekstu w przetwarzaniu języka naturalnego – metoda „osadzania słów" [168] (ang. *Word Embeddings*).

Przedstawiona sekcja nie wyczerpuje tematyki uczenia maszynowego, stanowiąc jedynie wprowadzenie dla dalszej części nieniejszego rozdziału. Dokładne przedstawienie historii tego obszaru sztucznej inteligencji oraz opis popularnych metod konkurencyjnych do sieci neuronowych można znaleźć w przywołanych już podręcznikach akademickich: [233, 314].

4.2 Sieci neuronowe

Na podstawie podręcznika akademickiego autorstwa S. Russella i P. Norviga [233], można opisać sieć neuronową jako model złożony z pojedynczych jednostek (czy też węzłów) – nazywanych neuronami – połączonych ze sobą w specyficzny sposób. Schemat bazowego, pojedynczego, „pełnoprawnego"[8] neuronu został przedstawiony na rysunku 4.3[9]. Połączenie pomiędzy neuronem i oraz j służy propagacji sygnału z neuronu i do neuronu j oraz posiada wartość numeryczną w postaci wagi $w_{i,j}$, która określa moc tego połączenia. Dodatkowo każdy neuron ma dodatkowy sygnał wejściowy stanowiący punkt odniesienia $a_0 = 1$ z przyporządkowaną mu wagą $w_{0,j}$. Każdy z neuronów j przetwarza ważoną sumę połączeń wejściowych w postaci sygnału wejściowego in_j,

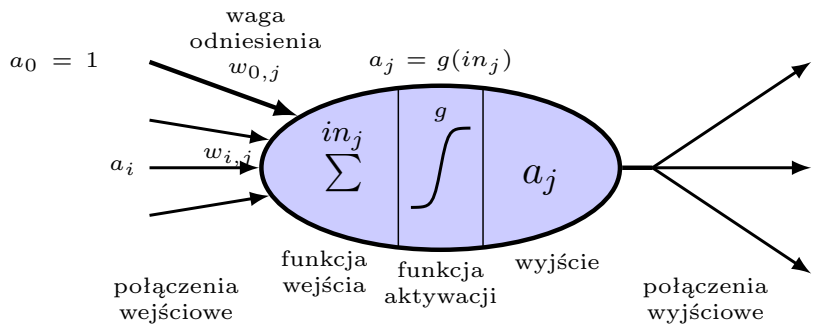
$$in_j = \sum_{i=0}^{n} w_{i,j} a_i. \tag{4.1}$$

Następnie uzyskuje sygnał wyjściowy z danego neuronu a_i za pomocą zastosowania funkcji aktywacji g na sygnale wejściowym,

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right). \tag{4.2}$$

[8] Sygnał wejściowy przekazywany na wejście sieci neuronowej jest modelowany w postaci węzłów które można uznawać za niepełnoprawne neurony, tj. nie posiadają funkcji aktywacji, a jedynie przekazują sygnał wejściowy dalej, do kolejnych warstw.

[9] Jest to zmodyfikowany model McCullocha-Pittsa, choć prosty w porównaniu do rzeczywistych biologicznych neuronów, stał się on podstawą dla wielu późniejszych modeli sieci neuronowych. Modyfikacją w stosunku do oryginału z 1943 jest np. brak ograniczenia funkcji aktywacji do funkcji progowej.



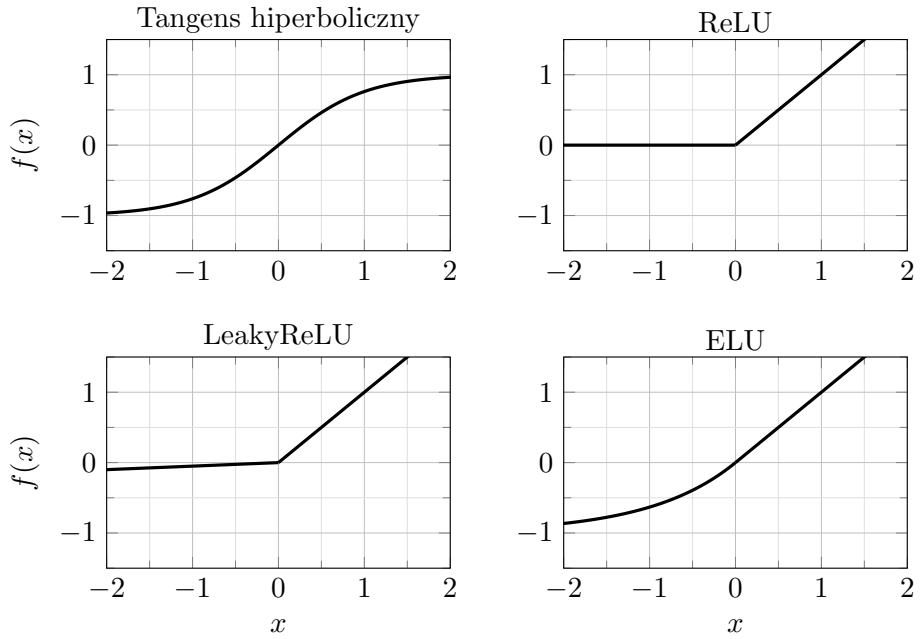
Rysunek 4.3: Schemat pojedynczego neuronu, podstawowej jednostki z której składają się sieci neuronowe.

Wybór funkcji aktywacji ma wpływ na działanie całej sieci neuronowej. Jedną z klasycznych funkcji aktywacji jest funkcja progowa,

$$g(in_j) = \begin{cases} 1 & \text{gdy } in_j \geqslant 0, \\ 0 & \text{gdy } in_j < 0, \end{cases} \tag{4.3}$$

dla której neuron działa jako element binarny. Choć funkcja tego typu stanowi swojego rodzaju uproszczony model biologicznego neuronu – w którym impuls elektryczny powstaje dopiero po otrzymaniu dostatecznie silnego bodźca – to poza zaletą nieliniowości, posiada ograniczenia związane z brakiem ciągłości. Funkcja progowa jest nieróżniczkowalna, co stanie się wyraźną wadą w późniejszym opisie metody trenowania sieci neuronowej. Z kolei prostsze podejście opierające się na zastosowaniu wyłącznie funkcji liniowej redukuje model sieci neuronowej do liniowego klasyfikatora, niezdolnego do przekraczania granic liniowej separowalności, niezbędnej do modelowania skomplikowanych, nieliniowych zależności występujących w rzeczywistych danych.

Historycznymi następcami funkcji progowej są funkcje sigmoidalna i tangens hiperboliczny. Obie są ciągłe i nieliniowe, lecz mają tendencję do zanikającego gradientu w obszarach daleko oddalonych od punktu zerowego, co utrudnia aktualizację wag w trakcie uczenia. Z tego względu w nowoczesnych sieciach często stosuje się funkcje takie jak ReLU (ang. *Rectified Linear Units*), która łączy prostotę operacji matematycznych dzięki zachowaniu liniowości dla wartości dodatnich, umożliwia reprezentację rzadkich aktywacji i nie ma problemu zanikającego gradientu. Jednakże, ReLU może prowadzić do problemu „martwych neuronów" w przypadku niewłaściwej inicjalizacji wag lub zbyt gwałtownych aktualizacji. Aby temu zaradzić, stosuje się warianty takie jak LeakyReLU czy ELU [57], które zachowują



Rysunek 4.4: Wybrane funkcje aktywacji wykorzystywane w treningu sieci neuronowych.

niewielką pochodną dla wartości ujemnych. Rozważania na temat wyboru właściwej funkcji aktywacji znaleźć można w źródłach [88, 105, 308]; na potrzeby dysertacji nie pogłębiono jednak tej tematyki, choć przedstawiono wybrane funkcje na rysunku 4.4.

W tym miejscu można dodać, że choć same sieci neuronowe są uproszczonym modelem, wzorowanym pierwotnie na elementach pracy mózgu, to nie jest to ich wada – szczególnie że biologiczna wierność oryginałowi nie jest tak kluczową i pożądaną cechą. Za przykład można tu podać impulsowe sieci neuronowe [126] – w których model neuronu (choć ciągle stosunkowo prosty) jest bliższy biologicznemu odpowiednikowi – poprzez symulację czasowego aspektu działania biologicznych neuronów. Pomimo tej „zalety" wykorzystanie sieci tego typu nie upowszechniło się. Koszt implementacji bardziej skomplikowanego modelu, oddającego na przestrzeni kilku iteracji narastanie potencjału elektrycznego, aż do uwolnienia impulsu nerwowego – nie doprowadził do wystarczającej poprawy jakości uzyskanych rezultatów aby uzasadnić zwiększony nakład obliczeniowy.

W ramach przykładu opisana zostanie, często wykorzystywana (choć prosta) architektura jednokierunkowej sieci neuronowej (ang. *feed-forward*

network, FNN) – czyli sieci, w której sygnał przepływa od warstwy wejściowej do wyjściowej, poprzez ewentualne warstwy ukryte (o ile występują), bez tworzenia cyklicznych połączeń zwrotnych. W rezultacie neurony takiej sieci tworzą skierowany graf acykliczny. Sieć neuronowa składająca się wyłącznie z warstwy wejściowej i wyjściowej, bez warstw ukrytych, nazywana jest perceptronem jednowarstwowym. Klasyczne metody trenowania takich prostych sieci (na przykład poprzez algorytm uczenia perceptronu, ang. *perceptron learning rule*) nie zostaną tu szerzej opisane. Materiały na ten temat można znaleźć w literaturze, w tym w podręczniku akademickim wykorzystanym przy tworzeniu tego podrozdziału [233].

Sieci neuronowe w których występują cykle są nazywane rekurencyjnymi sieciami neuronowymi (ang. *Recurrent Neural Network*, RNN). Ich stan zależy zarówno od dostarczonego sygnału wejściowego, jak i wcześniejszego stanu sieci. Architektura RNN pozwalana na zachowanie swojego rodzaju pamięci krótkotrwałej, co jest porządną cechą np. w przetwarzaniu języka naturalnego, gdzie kontekst ma znacznie. Pewną ceną za to jest jednak skomplikowanie sieci RNN, ich zachowanie można opisać jako system dynamiczny – który może zarówno osiągnąć stan stabilny, jak i przejawiać niepożądane zachowania, w tym: oscylacje, gwałtowne narastanie lub wygaszanie sygnałów czy inne zachowania chaotyczne. Sieci RNN nie będą szerzej omawiane w dysertacji, choć należy podkreślić, że pomimo trudności w ich wykorzystaniu, są one z powodzeniem stosowane w zadaniach wymagających pamięci sekwencyjnej, a ich wady mogą być ograniczane, dzięki dedykowanym modelom neuronów, jak LSTM[112] czy GRU[54].

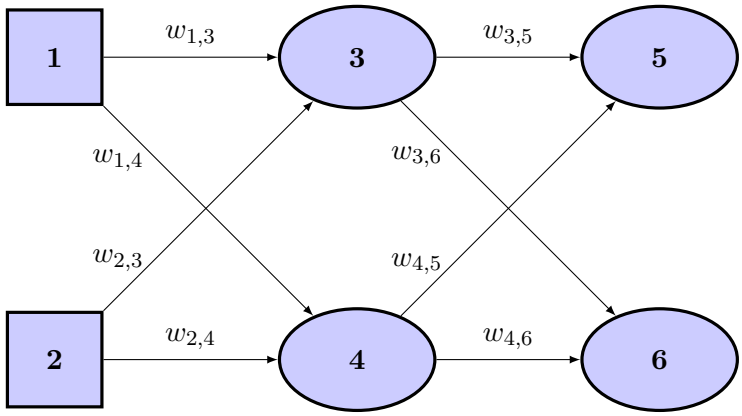
Wracając do jednokierunkowej sieci neuronowej, dla przykładu z rysunku 4.5 (ze źródła [233]) i dla wektora danych wejściowych $x = (x_1, x_2)$, wartości wyjściowe węzłów wejściowych a_1 i a_2 są ustalone na x_1 i x_2. Sygnał jest propagowany przez kolejne warstwy w ten sposób, że dla wyjścia węzła 5, mamy

$$a_5 = g(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) \tag{4.4}$$

$$= g\big(w_{0,5} + w_{3,5} \cdot g(w_{0,3} + w_{1,3}a_1 + w_{2,3}a_2) + w_{4,5} \cdot g(w_{0,4} + w_{1,4}a_1 + w_{2,4}a_2)\big) \tag{4.5}$$

$$= g\big(w_{0,5} + w_{3,5} \cdot g(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5} \cdot g(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2)\big). \tag{4.6}$$

W ten sposób jedno z wyjść sieci neuronowej jest wyrażone jako funkcja wejść i wag sieci. Analogicznie można przedstawić wyjście dla neuronu numer 6. Kluczowe jest, że jeśli jesteśmy w stanie obliczyć pochodne takich funkcji względem wag, możemy zastosować metodę spadku gradientu minimalizującą stratę do trenowania sieci. Co więcej, ponieważ funkcja reprezen-



Rysunek 4.5: Przykładowa jednokierunkowa wielowarstwowa sieć neuronowa – warstwa wejściowa złożona z dwóch wejść oznaczonych jako 1 i 2, warstwy ukrytej z neuronami 3 i 4 oraz warstwy wyjściowej złożonej z neuronów 5 i 6.

towana przez sieć może być nieliniowa, sieci neuronowe możemy efektywnie wykorzystywać do przeprowadzania nieliniowej regresji czy klasyfikacji.

Ze względu na swoją rolę w historii rozwoju sieci neuronowych przedstawiony zostanie algorytm propagacji wstecznej (ang. *Backpropagation Algorithm*, BP) [230], w oparciu o materiały z podręcznika akademickiego [233] – w tym przetłumaczony pseudokod algorytmu 1. Kluczowe znaczenie algorytmu BP wynika z jego zdolności do efektywnego trenowania wielowarstwowych sieci neuronowych, co stanowiło istotny krok w kierunku rozwoju architektur głębokich. Algorytm BP wykorzystuje regułę łańcuchową do dostosowywania wag w sieci, co rozwiązuje niektóre problemy z którymi borykały się wcześniejsze metody. Uniwersalność i skalowalność algorytmu BP pozwalają na jego zastosowanie w różnych architekturach sieci neuronowych. Przejrzysty mechanizm aktualizacji wag algorytmu BP przyczynił się do rozwoju metodyki treningu wielowarstwowych sieci neuronowych oraz ich zastosowania w praktyce, tym samym wpłynął na postęp w dziedzinie uczenia maszynowego.

Uproszczony schemat treningu sieci neuronowej z wykorzystaniem algorytmu BP, z pseudokodu 1, podzielić można na etapy takie jak:

- Inicjalizacja wag (linie 1–3 pseudokodu) – Trening rozpoczyna się od inicjalizacji wag sieci małymi wartościami losowymi, co stanowi krok przygotowawczy. Losowość wartości wag ma na celu zapobieganie symetrycznym aktualizacjom podczas treningu, co mogłoby mieć negatywny wpływ na proces.

Algorytm 1: Propagacja wsteczna dla wielowarstwowej sieci neuronowej, tłumaczenie własne na podstawie źródła [233]

Wejście: zbiór treningowy – każdy przykład złożony z wektora wejściowego x oraz wyjściowego y;
sieć neuronowa – sieć neuronowa złożona z L warstw z wagami $w_{i,j}$ oraz funkcją aktywacji g;

Wyjście: sieć neuronowa – ze zmienionymi wagami $w_{i,j}$

zainicjuj wektor błędów Δ, indeksowany jak węzły sieci;
for *wagi* $w_{i,j}$ *w sieci neuronowej* **do**
    $w_{i,j} \leftarrow$ niewielka wartość losowa;
do
    for *przykłady* (x, y) *w zbiorze treningowym* **do**
        `/* Propaguj sygnał wejściowy przez sieć w celu uzyskania sygnału wyjściowego. */`
        for *węzły* i *w warstwie wejściowej* **do**
            $a_i \leftarrow x_i$;
        for *warstwy* $l = 2, 3, \ldots, L$ **do**
            for *węzły* j *w warstwie* l **do**
                $in_j \leftarrow \sum_i w_{i,j} a_i$;
                $a_j \leftarrow g(in_j)$;
        `/* Propaguj błąd wstecz od warstwy wyjściowej do warstwy wejściowej. */`
        for *węzły* j *w warstwie wyjściowej* **do**
            $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$;
        for *warstwy* $l = L - 1, L - 2, \ldots, 1$ **do**
            for *węzły* i *w warstwie* l **do**
                $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$;
        `/* Wykonaj aktualizację wszystkich wag w sieci, wykorzystując propagowany błąd */`
        for *wagi* $w_{i,j}$ *w sieci neuronowej* **do**
            $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$;
while *osiągnięcia warunku stopu*;
return *wytrenowana sieć neuronowa.*

- Propagacja sygnału w przód (linie 6–11) – Dane wejściowe są przetwarzane przez całą sieć neuronową, po kolei dla każdej z warstw w przód, aż do osiągnięcia warstwy wyjściowej. W ramach neuronów tworzących poszczególne warstwy obliczana jest suma ważona wejść do neuronu, po czym aplikowana jest funkcja aktywacji g w celu uzyskania wartości wyjściowej z neuronu a.
- Obliczenie błędu (linie 12–13) – Obliczany jest błąd w postaci różnicy między wartością oczekiwaną na wyjściu sieci neuronowej y_j, a wartością otrzymaną a_j z wykorzystaniem obecnych stanu sieci – różnica ta stanowi podstawę do korekty wag w kolejnych krokach.
- Propagacja błędu wstecz (linie 14–16) – Błąd obliczony dla warstwy wyjściowej jest następnie propagowany wstecz przez sieć. Pozwala to określić, w jakim stopniu każda waga przyczynia się do błędu, co jest niezbędne dla korekty ich wartości.
- Obliczenie gradientu (linia 16) – Na tym etapie obliczany jest gradient błędu w odniesieniu do wag, z wykorzystaniem pochodnej funkcji aktywacji oraz propagowanej wartości wektora błędów Δ. Etap ten jest kluczowym elementem treningu sieci neuronowej, określającym, w którym kierunku zmiana wag doprowadzi do redukcji błędu w kolejnych iteracjach.
- Aktualizacja wag (linie 17–18) – Wagi w sieci są aktualizowane (zachodzi zmiana stanu sieci, tym samym następuje zmiana związana z „uczeniem") w oparciu o obliczony gradient i współczynnik uczenia α; tym samym zmiany wag $w_{i,j}$ prowadzą do poprawy otrzymanych wyników sieci, poprzez minimalizację błędu w następnych iteracjach.

Na koniec opisu należy zwrócić uwagę na iteracyjny charakter algorytmu BP, gdzie kroki przedstawione w liniach 4–19 pseudokodu 1 są powtarzane do momentu spełnienia kryterium zakończenia, które można dostosowywać do potrzeb – przykładowo: osiągnięcia zadanej wartości błędu czy też wykonania maksymalnej określonej liczby powtórzeń (epok).

Ważnym kolejnym usprawnieniem treningu jest wykorzystanie dodatkowo dedykowanych algorytmów optymalizacji, które pozwalają na określenie, jak duże powinny być aktualizacje tych parametrów – a tym samym, jak efektywnie aktualizować wagi sieci neuronowej. Jednym z popularnych algorytmów w których uwzględniono wspomniane modyfikacje jest Adam (skrót od ang. *Adaptive Moment Estimation*[143]). Jest to również algorytm gradientowy, ale w sposób adaptacyjny dostosowujący wielkość kroku dla każdego z parametrów, na podstawie szacunków pierwszego i drugiego momentu gradientów. Może to prowadzić do szybszej i bardziej stabilnej konwergencji (co jest możliwe poprzez pewną odporność na utykanie w lo-

kalnych optimach i płaskich obszarach). Adam jest swojego rodzajem połączeniem pomysłów z innych metod optymalizacji parametrów sieci.

Wybór metody optymalizacji jest dodatkową decyzją w trakcie treningu sieci neuronowych. Istnieją prace przeglądowe [25] czy badania [55], w których można znaleźć porównania i sugestie dotyczące właściwego wyboru algorytmu optymalizacji, w zależności od rozwiązywanego problemu, jak i wybranego modelu sieci neuronowej. W praktyce wybór ten często staje się kolejnym hiperparametrem który należy uwzględnić w trakcie treningu. Szczegółowy opis i problem wyboru algorytmów optymalizacji przeznaczonych do treningu jest zagadnieniem zbyt szerokim, aby został pogłębiony w dysertacji – dlatego podjęto decyzję, aby poprzestać jedynie na zarysowaniu tej ważnej tematyki.

4.2.1 Uczenie sieci neuronowych

W poniższym fragmencie tekstu omówiono cel uczenia sieci neuronowych oraz główne rodzaje błędów związanych z tym procesem. Wyzwaniem treningu sieci neuronowych jest otrzymanie modelu, z pomocą którego możliwe będzie uzyskanie wystarczająco dobrych wyników na wcześniej niewykorzystanych danych – taką korzystną cechę modelu w literaturze [94] określa się jako zdolność do generalizacji (ang. *generalization*). Jest ona bardzo ważna ze względu na praktyczne ograniczenia uczenia maszynowego: dane wykorzystane w ramach treningu przeważnie są jedynie ograniczoną, małą próbką w porównaniu do ogromnej populacji, z której pochodzą. Co gorsze próbka taka może być jeszcze obarczona błędami lub naturalnie występującym szumem związanym z procesem pomiarowym. Świetny wynik wyłącznie na zbiorze treningowym nie jest celem uczenia maszynowego – szczególnie w sytuacji, gdzie znajomość poprawnych rozwiązania dla próbki jest konieczna do samego rozpoczęcia procedury treningu w ramach uczenia nadzorowanego.

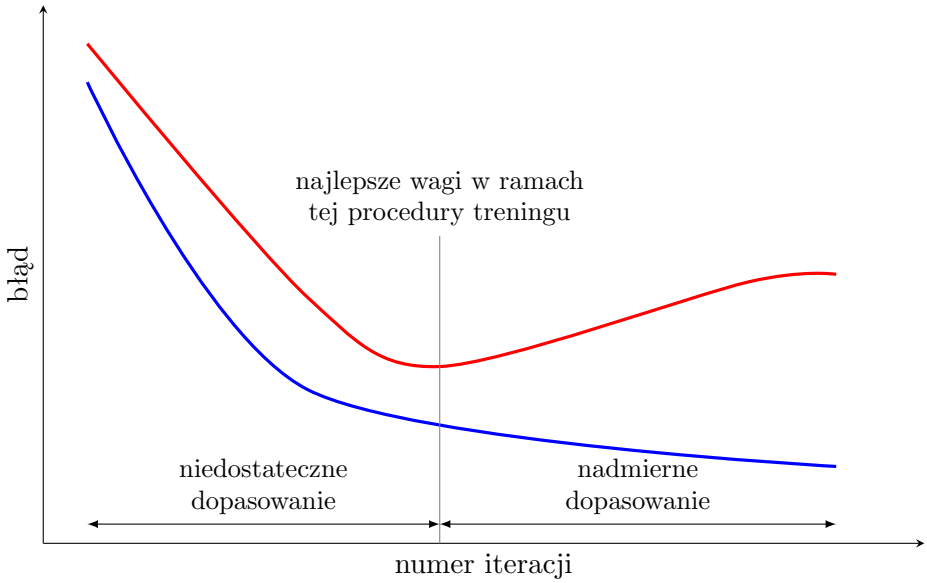
Standardem pozwalającym zwiększyć generalizację trenowanych modeli jest podział wykorzystanych danych w procesie uczenie maszynowego na rozdzielne zbiory:

- zbiór treningowy – wykorzystany do zmian wag sieci neuronowej w procesie treningu, to z tych danym sieć neuronowa ma za zadanie pozyskać wzorce pozwalające dopasować otrzymane dane wejściowe na sieć do danych wyjściowych, czyli oczekiwanych rezultatów. Oczekiwany jest proces indukcji od konkretnych przykładów do generalnych zasad. Zbiór treningowy powinien być zdywersyfikowany – posiadać wystarczająco reprezentatywnych przykładów – aby zdolność do ge-

neralizacji była możliwa do uzyskania poprzez trening modelu. Pojedyncze wykorzystanie wszystkich danych ze zbioru treningowego odmierza jedną „epokę" procesu treningu. Często wykorzystywany rozmiar zbioru treningowego mieści się w zakresie od 60% do 80% wykorzystanej, „całej" próbki danych.

- zbiór walidacyjny – zbiór wykorzystywany do sprawdzenia jakości otrzymanych rezultatów przez model w trakcie treningu, równocześnie ze zbiorem treningowym. Zadaniem zbioru walidacyjnego jest pomoc w dostrojeniu hiperparametrów oraz konfiguracji modelu. Głównym zadaniem zbioru walidacyjnego jest pomoc w ocenie procesu treningu – informacja czy proces ten „zmierza" w dobrym kierunku (pozwala ocenić pojawienie się typowych błędów związanych z treningiem). Zbiór walidacyjny wykorzystywany jest po każdej epoce treningu w celu pozyskania miar jakości modelu na danych niewykorzystanych bezpośrednio do treningu. Często wykorzystywany rozmiar zbioru walidacyjnego mieści się w zakresie od 10% do 20% wykorzystanej, „całej" próbki danych.
- zbiór testowy – zbiór wykorzystywany do finalnej oceny modelu po treningu. Zadaniem tego zbioru jest dostarczenie informacji o końcowej jakości modelu na danych wcześniej nie wykorzystywanych, stanowi to swojego rodzaju symulację co do spodziewanej jakości modelu na nowych danych spoza wykorzystanej próbki – ma za zadanie pomóc w predykcji zachowania modelu w warunkach zbliżonych do „produkcyjnych". Często wykorzystywany rozmiar zbioru treningowego mieści się w zakresie od 10% do 20% wykorzystanej, „całej" próbki danych.

I. Goodfellow, Y. Bengio i A. Courville w podręczniku akademickim [94] sugerują, że uwzględnienie generalizacji stanowi cechę wyróżniającą uczenie maszynowe od klasycznej optymalizacji. Prioretyzowanie wyłącznie minimalizacji błędu na zbiorze treningowym byłoby typowym problemem optymalizacji. To nie błąd na zbiorze treningowym jest priorytetem, a wynik na zbiorze testowym, który – co ważne – również nie jest ostatecznym celem, a służy za podstawę do oceny spodziewanej przydatności modelu na nowych, wcześniej niewykorzystywanych danych należących do rozwiązywanego problemu. Oczywiście, co podkreślono w [94], zdolność do poprawy wyników na zbiorze testowym na podstawie wykorzystania zbioru treningowego wynika z pewnych zakładanych cech problemu i zbioru danych. Przyjmuje się, że choć dane są przyporządkowane do różnych zbiorów (treningowy, walidacyjny, testowy), to są one od siebie niezależne ale pochodzą z identycznego rozkładu prawdopodobieństwa. Założenie o wspólnym



Rysunek 4.6: Wyidealizowany rysunek na którym przedstawiono proces trenowania modelu. Spadek błędu (oś Y) na zbiorze treningowym (niebieska krzywa) wraz z kolejnymi iteracjami (oś X) połączony jest z początkowym spadkiem błędu na zbiorze walidacyjnym (krzywa czerwona).

rozkładzie podstawowym pozwala na powiązanie oczekiwanego błędu na zbiorze treningowym z oczekiwanym błędem na zbiorze testowym.

Częstym powodem niedostatecznie dobrych rezultatów uczenia maszynowego – a więc braku dostatecznej generalizacji – są: błąd niedostatecznego dopasowania (ang. *underfitting*) lub błąd nadmiernego dopasowania (ang. *overfitting*). W ramach kolejnych iteracji treningu nawet dobrze dobranego modelu sieci neuronowej, model taki przechodzi ze stanu cechującego się najpierw niedostatecznym dopasowaniem, by przejść do nadmiernego dopasowania – co w sposób wyidealizowany zostało przedstawione na rysunku 4.6. Na rysunku oznaczono iterację od której błąd na zbiorze walidacyjnym zaczyna rosnąć, to właśnie z tej iteracji należy zwrócić wagi jako rezultat treningu, ze względu na prawdopodobnie najlepszy wynik generalizacji w ramach procedury treningu. Lewa część wykresu od tej iteracji reprezentuje błąd niedostatecznego dopasowania – prawa błąd nadmiernego dopasowania. Konieczną dobrą praktyką staje się więc monitorowanie przebiegu procesu i przeciwdziałanie obu tym błędom.

Niedostateczne dopasowanie modelu – błąd ten występuje, kiedy model osiąga złe rezultaty już na zbiorze treningowym, co w naturalny sposób

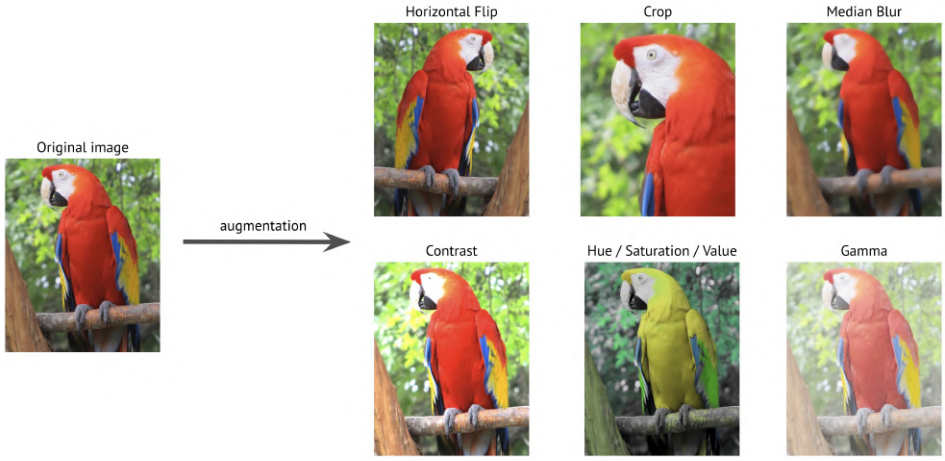
wyklucza możliwość generalizacji. Powodem niedostatecznego dopasowania może być: a) niedostateczny rozmiar danych treningowych b) brak właściwego przygotowania danych lub słaba jakość danych c) niedostateczna złożoność modelu dla rozwiązywanego problemu[10]. W celu ograniczania tego negatywnego efektu można spróbować rozwiązań takich jak: a) dostosowanie wykorzystanych danych poprzez inżynierię cech (w rozumieniu ang. *feature engineering*, [312]), np. poprzez zwiększanie liczby cech danych wejściowych b) próba „czyszczenia" danych – próba eliminacji błędów grubych, ale również redukcja szumów c) przedłużenie procesu treningu – poprzez zwiększanie liczby iteracji d) wykorzystanie bardziej skomplikowanego modelu.

Nadmierne dopasowanie – błąd ten występuje, kiedy model osiąga dobre rezultaty na zbiorze treningowym, jednak odbywa się to kosztem pogorszenia rezultatów na zbiorze testowym – tym samym zmniejsza się zdolność do generalizacji. Powodem nadmiernego dopasowania jest: a) wykorzystanie zbyt złożonego modelu w stosunku do rozwiązywanego problemu b) niedostatecznie duży rozmiar danych zbioru treningowego. W celu ograniczenia negatywnych efektów nadmiernego dopasowania można spróbować: a) zwiększyć rozmiar danych treningowych b) zmniejszyć złożoność wykorzystanego modelu c) zastosować techniki dedykowane do zapobiegania temu błędowi.

Do licznych technik pozwalających walczyć z nadmiernym dopasowaniem można zaliczyć metody takie jak: a) wcześniejsze przerwanie procesu treningu (ang. *early stopping*), w praktyce oznacza wykorzystanie wag modelu z iteracji przed rozpoczęciem pogorszenia wyników na zbiorze walidacyjnym b) wykorzystanie metody walidacji krzyżowej przy podziale danych b) stosowanie metod regularyzacji np. poprzez *dropout* [10] – mała szansa na „wyłączanie" małej części neuronów z całej sieci w procesie treningu c) wykorzystanie augmentacji [248] – szczególnie często spotykane w przetwarzania obrazu – gdzie zmienia się oryginalny obraz w celu wygenerowania na jego podstawie „nowych próbek" – zmieniając np. rozmiar oryginału, jego kolory, kontrast, saturację czy też wprowadza się przekształcenia zaburzające obraz. Przykład augmentacji na rysunku 4.7.

Innym rodzajem problemu wpływającego negatywnie na generalizację w ramach trenowania sieci neuronowych może być wyciek danych (ang. *data leakage*, [139]) – sytuacja, w której wykorzystana próbka danych zawierają informację, która znacznie ułatwia zadanie modelu do wskazania oczeki-

[10]Więcej informacji o pojęciu pojemności (ang. *capacity*) modelu, a więc jego zdolności do dopasowania się do hipotetycznej przestrzeni zbioru funkcji oraz niuansami z tym związanymi jak efektywna pojemność (ang. *effective capacity*) można znaleźć w źródle [94].



Rysunek 4.7: Augmentacja – proces sztucznego rozszerzania zbioru treningowego poprzez przekształcenie oryginalnych danych – przykład przekształceń z dokumentacji popularnej biblioteki *Albumentations* przeznaczonej do augmentacji obrazu – źródło: https://albumentations.ai/docs/introduction/image_augmentation/ dostęp 13.07.2022.

wanego rezultatu, co prowadzi do osiągnięcia przez model bardzo dobrych rezultatów na tej próbce jednak z obserwowalnym spadkiem jakości w wypadku zastosowania modelu na wcześniej niewykorzystanych danych, gdzie ułatwienie nie występuje[11]. Przykłady wycieku danych: dane typowe dla serii czasowej np. do przewidywania kursu giełdowego na podstawie danych z ostatniego miesiąca wstecz – przydzielone w sposób losowy do zbioru treningowego, walidacyjnego oraz testowego (co jest dobrą praktyką dla danych nienależących do serii czasowej) – część z danych, które model ma za zadanie przewidywać w ramach zbioru walidacyjnego i testowego, staje się dostępna w zbiorze treningowym.

Chociaż w ramach przedstawionego tekstu podkreślono rolę minimalizacji błędów na zbiorach treningowym, walidacyjnym i testowym, warto zaznaczyć, że błąd minimalizowany w trakcie treningu nie musi być identyczny z błędem ocenianym przy finalnym wyborze i ocenie modelu sieci neuronowej. W trakcie treningu wykorzystuje się funkcje strat (ang. *loss functions*), które kwantyfikują różnicę między predykcją modelu, a rzeczywistą wartością zawartą w przykładach treningowych. Kluczowe jest aby posiadały one pożądane właściwości umożliwiające efektywne stosowanie algorytmów optymalizacyjnych wagi sieci neuronowej, w tym me-

[11] Cytatem pasującym do tego kontekstu są słowa ekonomisty C. Goodhart'a: „Gdy dowolny wskaźnik zaczyna być traktowany jako cel, przestaje być dobrym wskaźnikiem".

tod gradientowych (czyli np. ciągłość i różniczkowalność). Jako klasyczne przykłady funkcji strat można przedstawić: błąd średniokwadratowy (ang. *Mean Squared Error*, MSE), stosowany w problemach regresji czy entropię krzyżową (ang. *Cross-Entropy Loss*), używaną w problemach klasyfikacji. Wybór konkretnej funkcji strat powinien być dostosowany do natury rozwiązywanego problemu, w tym uwzględniać typ wykorzystywanych danych. Można zauważyć pewną analogię między potrzebą minimalizacji funkcji straty, a maksymalizacją funkcji celu w problemach optymalizacyjnych.

Po procesie treningu ocena modelu może opierać się na metrykach niezależnych od wymagań samego procesu optymalizacji, takich jak wspomniana ciągłość czy różniczkowalność. W tym kontekście warto użyć bardziej złożonych metryk niż wyłącznie dokładność, w tym takich przykładowo: precyzja, czułość, miara F1, czy współczynnik podobieństwa Jaccarda. Każda z tych metryk ma swoje ograniczenia i może nie odzwierciedlać rzeczywistych potrzeb, dlatego niezwykle ważne jest odpowiednie dopasowanie wykorzystanej metryki do specyfiki problemu, który model sieci neuronowej ma rozwiązywać. Chociaż truizmem może wydawać się konieczność dopasowania odpowiedniej metryki do konkretnego rozwiązywanego problemu przy zadanym kontekście – warto wspomnieć o prostym przykładzie wykorzystania macierzy błędów w klasyfikacji binarnej dla danych medycznych, aby podkreślić wagę odpowiedniej oceny modelu i wynikających z tego konsekwencji. Hipotetyczny model oceniający chorego pacjenta jako chorego (wynik prawdziwie pozytywny) oraz zdrowego pacjenta jako zdrowego (wynik prawdziwie negatywny) jest sytuacją niewątpliwie pożądaną. Natomiast ocena zdrowego pacjenta jako chorego (wynik fałszywie pozytywny) jest błędem, który może doprowadzić do niepotrzebnego stresu dla pacjenta i dodatkowych badań diagnostycznych, które są potencjalnie kosztowne, ale ostatecznie prawdopodobnie doprowadzi do wykluczenia choroby. Z kolei ocena chorego pacjenta jako zdrowego (wynik fałszywie negatywny) może skutkować przedwczesnym wykluczeniem pacjenta z dalszych badań i stworzyć realne zagrożenie dla jego zdrowia i życia. Dlatego minimalizacja błędów fałszywie negatywnych powinna być kluczowa w przypadku testów diagnostycznych, nawet jeżeli prowadzi to do zwiększenia liczby błędów fałszywie pozytywnych, chociaż nadrzędnym celem jest zminimalizowanie obu typów błędów, o ile jest to możliwe. Kwestia doboru odpowiednich funkcji strat oraz metryk oceny końcowego modelu po treningu nie zostanie pogłębiona w ramach tej dysertacji; zamiast tego warto odesłać do źródeł, które szerzej omawiają tę tematykę [56, 80, 94, 121, 210, 233].

4.2.2 Głębokie uczenie

Uczenie głębokie (ang. *Deep learning*, DL), w najprostszych i mocno uogólnionych słowach za podręcznikiem akademickim Z. H. Zhou [314], odnosi się typowo do treningu modeli sieci neuronowych z dużą liczbą warstw. Jednak za podręcznikiem S. Russella i P. Norviga [234] można podać bardziej rozbudowaną definicję: *głębokie uczenie to obszerna rodzina technik uczenia maszynowego, w której hipotezy przyjmują postać złożonych obwodów algebraicznych z dostrajalną siłą (wagą) poszczególnych połączeń. Słowo „głęboki", odnosi się do faktu, że obwody są zwykle zorganizowane w wiele warstw (ang. layers), co oznacza, że ścieżki obliczeniowe prowadzące od wejść składają się z wielu kroków.*

Więcej warstw ukrytych w sieci neuronowej, przekłada się ma większą „pojemności" modelu, co oczywiście wynika ze wzrostu liczby parametrów (w postaci wag). Czemu więc uczenie głębokie jest tak ważne i gdzie leży jakościowy przełom? Pytanie to, staję się tym bardziej zasadne, ze względu na szerokie zastosowanie DL – np. za [234]: *głębokie uczenie jako obecnie najczęściej stosowanym podejściem do rozwiązywania problemów z takich dziedzin jak wizualne rozpoznawanie obiektów, tłumaczenie maszynowe, rozpoznawanie i syntezowanie mowy, a także synteza obrazów.* Odpowiedź można podać za [314]: Choć klasyczne sieci neuronowe zapewniały znaczącą zdolność do generalizacji, kluczowe w uczeniu głębokim jest wykorzystanie wielu ukrytych warstw do przetwarzania informacji warstwa po warstwie. Mechanizm ten jest stopniową konwersją reprezentacji danych wejściowych, które początkowo nie muszą być ściśle związana z pożądanym wynikiem końcowym – zadanie, które pierwotnie mogło być trudne, staje się łatwiejsze w realizacji. Innymi słowy, poprzez wielowarstwowe przetwarzanie, reprezentacje niskiego poziomu cech są konwertowane do reprezentacji cech wysokiego poziomu – tym samym uczenie głębokie może być postrzegane jako sposób uczenie cech (ang. *feature learning*) lub uczenie reprezentacji (ang. *representation learning*).

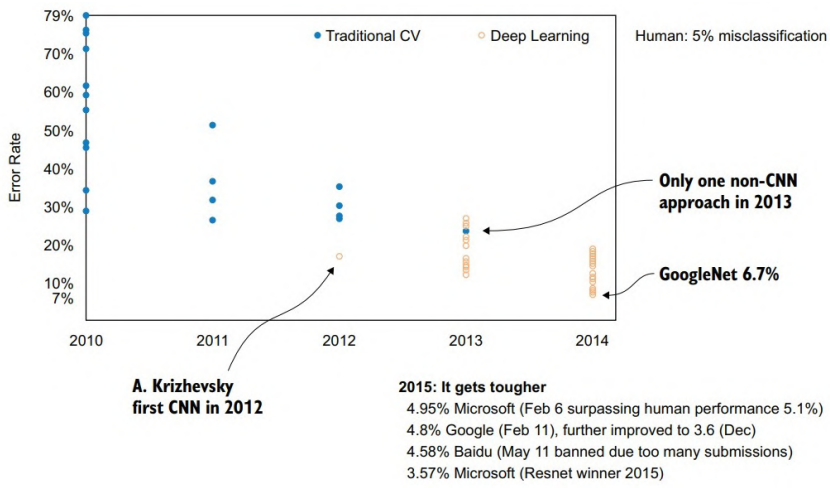
W tym miejscu można podać analogię (czy raczej biologiczną inspirację do powstania sieci konwolucyjnych) w postaci przetwarzanie sygnału wizyjnego w ramach mechanizmów biologicznych – od przekształcenia światła w sygnał elektryczny, aż do procesu rozpoznawania i nadawania znaczenia – co odbywa się stopniowo przez kolejne obszary kory wzrokowej (w tym przykładowo: V1 – obszar odpowiedzialny za analizę prostych kształtów i konturów; V2 i V4 – w których zachodzi rozpoznawanie bardziej złożonych kształtów i cech obrazu; obszar IT – gdzie dochodzi do rozpoznawania pełnych obiektów). Podobnie w konwolucyjnych sieciach neuronowych początkowe warstwy służą do rozpoznawania prostych kształtów i konturów, by

w kolejnych warstwach rozpoznawać części obiektów lub specyficzne wzory. W końcowych warstwach sieć zdolna jest do klasyfikacji całych obiektów, jednak bazując na cechach niższego poziomu.

Zastosowanie głębokiego uczenia oznacza znaczy postęp w stosunku do klasycznego uczenia maszynowego, gdzie aplikacja systemów uczenia maszynowego wymagała ludzkich ekspertów, odpowiedzialnych za inżynierię cech (ang. *feature engineering*), wykorzystujących wiedzę dziedzinową w przygotowaniu (a często też odkrywaniu) właściwych cech dostosowanych do przetwarzanych danych. Zadanie to było kluczowe dla jakości generalizacji modelu, ale było trudne i czasochłonne. Eliminacja potrzeby wykorzystywania wiedzy eksperckiej oraz zdolność do swoistego „samodzielnego odkrywania" odpowiednich cech bezpośrednio z danych przyspiesza proces tworzenia skutecznych modeli dla różnorodnych zbiorów danych i przybliża rozwój sieci neuronowych do osiągnięcia pożądanego stanu „w pełni automatycznej analizy danych". Należy jednak podkreślić, że głębokie uczenie posiada także swoje ograniczenia. Wymaga ono, na przykład, znacznie większych zbiorów danych podczas treningu, aby uniknąć nadmiernego dopasowania modelu. W konsekwencji, dla zbiorów danych o mniejszej skali, klasyczne metody uczenia maszynowego mogą okazać się bardziej adekwatne.

Przełom spowodowany uczeniem głębokim można wyraźnie prześledzić na podstawie historii zbioru danych ImageNet. Zbiór ten (zapoczątkowany w 2006) jest dedykowany wybranym problemom z dziedziny widzenia komputerowego (ang. *computer vision*), w tym: klasyfikacji obrazów (określenie klasy obiektu występującego na danym obrazie, ograniczonym do wcześniej zdefiniowanych kategorii) czy detekcji obiektów (oprócz klasy obiektu, należy też wskazać jego położenie oraz skale w postaci prostokąta ograniczającego obszar jego występowania – ang. *bounding box*). Za jeden z powodów powstania zbioru można podać pewną frustrację związaną z wynikami klasycznych metod widzenia maszynowego. Twórcy (w tym Fei-Fei Li) jako inspirację wskazali, że skoro ludzki system wizyjny trenowany jest przez lata na ogromnej ilości danych, to potrzebny jest podobny poziom ekspozycji przy tworzeniu algorytmów. Tymczasem wcześniejsze zbiory danych były stosunkowo małe i nie odzwierciedlały złożoności „prawdziwego świata".

Wraz ze zbiorem danych powstał (w 2010 roku) powiązany z nim konkurs ILSVRC (*ImageNet Large Scale Visual Recognition Challenge* [232]), konkurs podobnie jak sam zbiór był tworzony z myślą o dużej skali. Przykładowo, za [78], w ramach zadania klasyfikacji w ILSVRC z roku 2010 dostępnych było ponad 1,2 miliona obrazów w samym zbiorze treningowym, przy czym obiekty występujące na obrazach pochodziły z 1000 ka-



Rysunek 4.8: Błąd rozwiązań w konkursie ILSVRC na przestrzeni lat. Szczególnie ciekawy jest przeskok pomiędzy zakończeniem zastosowania metod klasycznych przetwarzania obrazów, na rzecz rozwiązań opartych o konwolucyjne sieci neuronowe (źródło rysunku: [78]).

tegorii. Na wykresie 4.8 ze źródła [78] widać procent błędu dla wybranych rozwiązań konkursu ILSVRC w latach 2010–2014. W edycjach z lat 2010 i 2011 dominowały klasyczne metody rozpoznawania obrazów, w tym techniki takie jak SIFT czy HOG, dodatkowo ważnym elementem rozwiązania było ręczne projektowanie cech – jednak nawet najlepsze klasyczne podejście posiadało błąd rzędu 25,8%. Znaczny przełom nastąpił w roku 2012, kiedy to, pomimo rozszerzenia zbioru ImageNet, model AlexNet [156] – głęboka konwolucyjna sieć neuronowa – zdeklasowała pozostałe klasyczne algorytmy, wygrywając z kolosalną różnicą, aż o 10% współczynnika błędu – model AlexNet osiągnął jedynie 15,3% współczynnika błędu. Tak duży przeskok pokazał potencjał głębokich sieci neuronowych oraz stanowił wyraźną inspirację do dalszej eksploracji i optymalizacji struktur CNN. Warto zwrócić na uwagę na oznaczenie CNN jako pomarańczowe kropki na wykresie 4.8 oraz przeskok, czy wręcz wyeliminowaniu rozwiązań klasycznych w konkursie, oznaczonych jako kropki niebieskie. Tym samym, choć AlexNet była jedyną siecią konwolucyjną podczas ILSVRC 2012 – to już w ILSVRC w roku 2013 proporcje zupełnie się odwróciły, i większość zgłoszeń opierała się o różne warianty konwolucyjncyh sieci neuronowych, jedynie jedno zgłoszenie można było określić mianem podejścia klasycznego.

Rok 2015 konkursu ILSVRC obrazuje kolejny ciekawy trend: liczny udział korporacji w konkursie – co można odebrać, jako potwierdzenie świa-

domości potencjału komercyjnego wykorzystania uczenia głębokiego (ale też budowanie przewagi konkurencyjnej oraz wizerunku firmy jako lidera AI – pomocny obraz w walce o uzdolnionych pracowników). Firma Microsoft, jako zwycięzca konkursu w 2015, dzięki modelowi Resnet [106], była w stanie osiągnąć błąd na poziomie jedynie 3,57%; rozwiązanie korporacji Baidu, osiągnęło finalny wynik 4.58%, choć zostało zdyskwalifikowane, w związku ze zbyt dużą liczbą zgłoszeń do konkursu; rozwiązanie firmy Google z błędem 4,8%. Co interesujące w kontekście tak dobrych wyników, już poziom błędu 5% był określany jako zbliżony do ludzkich możliwości – przy czym wynikał z trudności samego zadania, przy tak ogromnej liczbie kategorii w klasyfikacji, część klas oznacza obiekty, wymagające pewnej specjalistycznej wiedzy, której przeciętny człowiek nie musi koniecznie posiadać – np. posiadania informacji pozwalającej na odróżnienie konkretnej rasy psa z ponad 100 oznaczonych i występujących w zbiorze danych.

Ze względu na ograniczony zakres rozdziału, pominięty zostanie szczegółowy rys historyczny i wyczerpujący przegląd pojęć związanych z głębokim uczeniem[12]. Nie sposób jednak nie odwołać się do choć kilku popularnych architektur, wraz z powiązanymi badaniami, by tym dobitniej podkreślić sukces wykorzystania uczenia głębokiego.

Sieć DBN [111] (ang. *Deep Belief Network*), będąca jednym z pierwszych modeli wielowarstwowych sieci neuronowych, została zaprojektowana w celu ograniczenia problemów związanych z zanikającym gradientem. Wielowarstwowość, zapewniającą zdolność do ekstrakcji abstrakcyjnych cech danych, osiągnięto poprzez hierarchiczne ułożenie Ograniczonych Maszyn Boltzmanna [110] (ang. *Restricted Boltzmann Machines*, RBM) lub autoenkoderów. RBM to dwuwarstwowy model, który zawiera warstwę wejściową (widoczną) oraz jedną warstwę ukrytą, przy czym jest ograniczony brakiem połączeń pomiędzy neuronami tej samej warstwy. Uczenie RBM odbywa się poprzez aktualizację wag, wykorzystując różnicę pomiędzy danymi oczekiwanymi a uzyskanymi z modelu, minimalizując energetyczną funkcję kosztu aż do osiągnięcia stanu równowagi. W sieci DBN, warstwy RBM ułożone są hierarchicznie, a uczenie modelu odbywa się w dwóch fazach: a) fazie wstępnego uczenia bez nadzoru (ang. *pre-training*) – każda z warstw jest trenowana niezależnie, korzystając z danych pochodzących z poprzedniej warstwy, a kolejne warstwy są trenowane po wstępnym uczeniu warstwy poprzedniej, gdzie wyjścia z jednej warstwy są używane jako dane wejściowe dla następnej; b) fazie dostrojenia z nadzorem (ang. *fine-tuning*) – mającej na celu poprawę wydajności całego modelu na danych etykietowanych przez dodatkowy krok treningu w sposób nadzorowany, najczęściej z wy-

[12]Ponownie można tu przywołać źródła takie jak [56, 78, 94, 233, 314].

korzystaniem algorytmu propagacji wstecznej. DBN były z powodzeniem wykorzystywane [119, 190, 237] w zakresie klasyfikacji, redukcji wymiarowości, regresji czy klasteryzacji, łącząc w bardzo praktyczny sposób wstępne uczenie sieci głębokich w sposób nienadzorowany z dostrojeniem całego modelu w sposób nadzorowany. Wieloetapowy proces treningu należy również zaliczyć do szerszego, a powszechnego trendu w ramach uczenia głębokiego.

Model Yolo [219] stanowi ważny krok w dziedzinie wykrywania obiektów, wprowadzając efektywne i wydajne podejście do problemu detekcji. Wyznaczenie prostokąta ograniczającego obszar występowania klasy ograniczono do jednoetapowej analizy obrazu, w przeciwieństwie do klasycznego podejścia dwuetapowego, gdzie najpierw proponowano obszary zainteresowania, aby następnie przeprowadzić klasyfikację. Rodzina Yolo ewoluowała na przestrzeni kolejnych badań, zwiększając numer wersji modelu (poprawiając jakość detekcji, w tym przez lepsze wykrywanie małych obiektów czy zwiększenie wydajności). Doczekała się nawet nieoficjalnej, kontrowersyjnej kontynuacji, niepowiązanej z pierwotnymi twórcami, a stworzonej przez firmę *Ultralytics*[13][129]. Produkt *Ultralytics* zapewnił szeroki dostęp do przetestowanej i łatwej w wykorzystaniu implementacji, co przyczyniło się do jeszcze większej popularności modelu.

Model U-Net [226] został zaprojektowany specjalnie z myślą o uzyskaniu bardzo dobrych wyników segmentacji w warunkach niewielkiej ilości danych treningowych, pierwotnie dla obrazów biomedycznych (ze względu na koszty pozyskania tego rodzaju danych, a także ich niewielką dostępność w tym zastosowaniu). Sama architektura U-Net była adaptowana i modyfikowana w wielu różnych aplikacjach i stała się jednym z podstawowych modeli dla segmentacji obrazów. Nazwa modelu, a konkretnie „U", bierze się od zastosowania dodatkowych połączeń skrótowych pomiędzy warstwami, co pozwala na przekazanie lokalnych informacji i zachowanie kontekstu przestrzennego, przekładające się na poprawę jakości segmentacji.

Z kolei metoda **osadzania słów** została wykorzystana w pracy [186] do odwzorowania korpusu językowego, i składających się na niego słów, w przestrzeni wielowymiarowej. Autorzy przedstawili kilka modeli, w tym model CBOW, który próbuje przewidzieć aktualne słowo na podstawie kontekstu (otaczających go słów), oraz model Skip-Gram, który robi to odwrotnie, przewidując kontekst na podstawie danego słowa. Reprezentacja słów w ciągłej przestrzeni wielowymiarowej pozwala nie tylko mierzyć podobieństwo między słowami, ale również wykonywać „działania arytmetyczne" na reprezentacjach słów, takie jak: Król − Mężczyzna + Kobieta = Królowa.

[13] Więcej o kontrowersji: https://blog.roboflow.com/yolov4-versus-yolov5/ – dostęp 07.06.2023.

Choć w pierwotnej pracy wykorzystywano stosunkowo proste sieci neuronowe, metoda osadzania słów stała się ważnym krokiem w rozwoju przetwarzania języka naturalnego. Służyła jako fundament dla wielu późniejszych prac związanych z głębokim uczeniem w dziedzinie NLP, czy to poprzez wykorzystanie wstępnie wytrenowanych osadzeń [7, 265], czy też przez rozbudowę samej idei osadzania słów [24, 48].

Architektura Transformer [290] wprowadza mechanizm atencji[14], który pozwala na lepsze wykorzystanie zależności obecnych w sekwencji danych wejściowych, umożliwiając „zważenie" znaczenia każdego elementu w szerszym kontekście całej sekwencji danych wejściowych. Dzięki swojej skalowalności oraz zdolności do efektywnego przetwarzania długich sekwencji, model Transformer zastąpił tradycyjne modele rekurencyjnych sieci neuronowych (jak LSTM) w dziedzinie przetwarzania języka naturalnego. Zgodnie ze źródłem [170]: o ile model Transformer jest uczony na wystarczająco dużym zbiorze danych, zwykle osiąga wyniki lepsze od klasycznych konwolucyjnych czy rekurencyjnych sieci neuronowych. Jednym z możliwych wyjaśnień tego fenomenu (oprócz podstawowej różnicy jaką jest większa pojemność) jest mniejsza liczba założeń co do struktury danych zawartych w samej architekturze, co przekłada się na znacznie większą elastyczność wykorzystania tego modelu, choć przewaga ta nie została jeszcze dostatecznie zbadana w sposób teoretyczny. Prace przeglądowe [170, 274] dotyczące tej architektury pozwalają zwrócić uwagę na liczne odmiany podstawowej architektury, nazywane zbiórczo jako „X-former". W tym określeniu zawierają się modele takie jak: „Reformer", „Linformer", „Performer", „Longformer". Odmiany te stanowią rozwinięcie oryginalnej architektury. Część z nich dostosowuje ją do zastosowań w innych dziedzinach niż przetwarzanie języka, takich jak widzenie komputerowe, przetwarzanie dźwięku czy uczenie ze wzmocnieniem. Znaczna część nowych odmian bazowego modelu polega na próbie zniwelowania podstawowego problemu Transformera, jakim jest problem ze skalowalnością tej architektury związanym z kwadratowym zapotrzebowaniem na pamięć i moc obliczeniową w kontekście długości sekwencji dla mechanizmu atencji. Modyfikacje modelu można pogrupować w rodziny określonych ulepszeń: a) pozwalające ograniczyć mechanizm atencji do ustalonego z góry okna, b) pozwalające na nauczenie się określonych wzorców okna dla mechanizmu atencji, c) wykorzystujące mechanizm pamięci pozwalający na dostęp do wielu obiektów jednocześnie (w tym całej sekwencji), d) wykorzystujące funkcje jądrowe lub inne techniki, które bazują na niskopoziomowym przybliżeniu macierzy atencji, e) inne techniki, takie jak wykorzystanie dodatkowego mechanizmu rekurencji,

[14] Alternatywne tłumaczenie to mechanizm uwagi.

rzadkie modele atencji lub połączenie kilku rodzajów okna atencji.

Ogromne modele językowe (ang. *Large Language Model*, LLM) to nowoczesne, złożone modele przetwarzania języka naturalnego, które pozwalają osiągnąć zaskakująco dobre wyniki w typowych zadaniach z zakresu przetwarzania języka naturalnego, w tym: wnioskowania z tekstu, odpowiadania na pytania, oceny podobieństwa semantycznego, kategoryzację tekstu czy – co najbardziej imponujące – symulowanie płynnej konwersacji. Do znanych przedstawicieli tej rodziny należą choćby modele: LaMDA [280], GPT-3 [43] i GPT-4 [204] (zastosowane w popularnym narzędziu ChatGPT), BERT [76] i jego rozwinięcie RoBERTa [172]. Sukces tych modeli sprawia, że są ciągle rozwijane i adaptowane do nowych wyzwań, które nie były przewidziane podczas ich tworzenia[15] – na przykład do rozwiązywania zadań matematycznych [58]. LLMy opierają się głównie na opisanej wyżej architekturze „Transformer" i charakteryzują się następującymi cechami: a) ogromną skalą, osiągającą setki milionów, a nawet dziesiątki miliardów parametrów; b) procesowi treningu możliwemu dzięki wykorzystaniu dużych zbiorów danych, często rozszerzanych przez automatyczną ekstrakcję z publicznie dostępnych internetowych źródeł, takich jak Wikipedia czy fora internetowe, np. Reddit; c) koniecznością wykorzystania rosnącej mocy obliczeniowej, umożliwiającej przeprowadzenie procesu uczenia i zapewnienie konwergencji modelu. Należy jednak zwrócić uwagę na poważne ograniczenia związane z praktycznym zastosowaniem modeli LLM [18, 163, 204]. Modele te są uczone na ogromnych zbiorach danych, które przez swoją skalę mogą zawierać błędy, sprzeczności wewnętrzne czy dezinformację, co może prowadzić do ich powielania w generowanych odpowiedziach. Ponadto, LLM są optymalizowane do generowania płynnych odpowiedzi z językowego punktu widzenia, co może skutkować tworzeniem treści naturalnie brzmiących, ale nieprecyzyjnych lub nieprawdziwych. Mimo że wykorzystanie LLM może znacznie ułatwić codzienną pracę, należy być świadomym ich ograniczeń i krytycznie weryfikować informacje uzyskane dzięki nim, korzystając z innych wiarygodnych źródeł. Badania [135] wskazują, że choć większa liczba parametrów, obszerniejsze zbiory danych i zwiększona moc obliczeniowa zazwyczaj przyczyniają się do lepszych wyników, istnieją punkty nasycenia, po których dodatkowe zasoby przestają przynosić proporcjonalne korzyści.

Modele z rodziny MobileNet [118] – czyli rodzaj konwolucyjnej sieci neuronowej dostosowanej do wykorzystania na platformach o ograniczonej mocy obliczeniowej (urządzeniach mobilnych). Model ten stanowił ważny

[15] Interesujące i nietypowe zastosowania LLM zostały zobrazowane w https://youtu.be/2A9PLW6BCx4. Dostęp na dzień 13.06.2023.

wkład w efektywne podejście do kompromisu pomiędzy jakością wyników modelu, a wydajnością modelu[16]. Było to możliwe dzięki zastosowaniu rozwiązań redukujących liczbę wymaganych operacji (inny rodzaj wykorzystanej konwolucji) czy też możliwości modyfikowania liczby parametrów poprzez zmianę szerokości i rozdzielczości wejściowej modelu.

Model GAN (Generatywna Sieć Konkurencyjna[17], od ang. *Generative Adversarial Network*) [95] to nowatorskie podejście do uczenia nienadzorowanego w modelach generatywnych. Celem takich modeli jest tworzenie realistycznych, generowanych komputerowo danych, takich jak obrazy, wideo, tekst czy audio. Kluczową ideą GAN jest rywalizacja dwóch sieci: generatora i dyskryminatora. Generator ma za zadanie tworzenie nowych, fikcyjnych danych. Początkowo generuje on dane losowe, ale w procesie uczenia produkowane próbki są coraz bardziej zbliżone do danych treningowych. Zadaniem generatora jest stworzenie próbek tak realistycznych, żeby oszukać dyskryminatora. Dyskryminator to klasyfikator trenowany w celu odróżnienia prawdziwych próbek (pochodzących ze zbioru treningowego) od fałszywych, wygenerowanych przez generator. Rywalizacje tych dwóch sieci można interpretować w ramach teorii gier za [234]: W idealnym stanie równowagi generator powinien doskonale naśladować rozkład danych treningowych, tak że dyskryminator nie będzie mógł lepiej rozróżniać próbek niż przez losowe zgadywanie. Mimo, że trening GAN-ów jest wyzwaniem, technika ta zyskała status ważnego narzędzia w generatywnych metodach uczenia maszynowego. Model ten jest nieustannie doskonalony i adaptowany do nowych zastosowań. Przykładowo, prace takie jak [41] pozwoliły na efektywne skalowanie treningu GAN-ów, umożliwiając ich aplikację na znacznie większą skalę, zaś badania przedstawione w [138] wprowadziły innowacyjne podejście do generowania obrazów, integrując tworzenie grafiki z transferem konkretnego stylu.

Ostatnią wymienioną dziedziną znaczących osiągnięć uczenia głębokiego są **metody generowania obrazów na podstawie dostarczonego opisującego je tekstu** (ang. *Text-to-Image Generation*, T2IG). Proces ten wymaga zrozumienia zarówno semantyki tekstu, jak i wizualnych aspektów generowania obrazów, które polegają na konwersji opisów językowych, zazwyczaj podanych w formie naturalnego języka, na odpowiadające im wizualne reprezentacje. Chociaż każde z tych zagadnień, rozpatrywane oddzielnie, stanowiło ogromne wyzwanie dla klasycznych metod sztucznej inteligencji, połączenie efektu skali wykorzystywanych danych oraz głębokich

[16] „Rozrost" modeli, poprzez ciągle rosnącą liczbę parametrów, aby osiągnąć mikroskopijne poprawy na znanych zbiorach danych nie powinien być celem samym w sobie w DL – praktyczność zastosowania modelu i odpowiedni dobór metod do celów ma znaczenie.

[17] Alternatywne tłumaczenie za [234]: Generatywna sieć kontradyktoryjna.

sieci neuronowych (takich jak sieci GAN [220, 310], sieci RNN, architektura Transformer [212] oraz ich kombinacje) pozwala na generowanie realistycznych obrazów nawet bez konieczności udziału „świadomego zrozumienia" zaawansowanej silnej sztucznej inteligencji (AGI). Co zaskakujące, zważywszy na trudność zadania, T2IG z powodzeniem łączy dwa obszary największych sukcesów DL (przetwarzanie języka i widzenie komputerowe) w jednym zastosowaniu.

Choć można było w tym miejscu pogłębić opis modeli T2IG, powołując się na prace przeglądowe [1, 214], to ze względu na dostępność platform pozwalających na generację obrazów przez zwykłych użytkowników – a także nieukrywaną atrakcyjność wizualną wyników – zdecydowano się przedstawić przykładowe różnice pomiędzy wybranymi modelami T2GI, generując lub wykorzystując wygenerowane obrazy dla zdań w języku angielskim:

- a) *Colorless green ideas sleep furiously* (pol. bezbarwne zielone idee śpią wściekle) na rysunku 4.9,
- b) *Schrödinger's cat gripping the latest research papers* (pol. kot Schrödingera trzymający najnowsze prace badawcze) na rysunku 4.10,
- c) *Writing a PHD dissertation as a form of suffering in Beksiński style* (pol. pisanie rozprawy doktorskiej jako forma cierpienia w stylu Beksińskiego) na rysunku 4.11.

Należy jednak zaznaczyć, że nie jest to porównanie „uczciwe". Model DALL·E mini[18] to uproszczona wersja oryginalnego modelu DALL·E w wersji pierwszej, która to modyfikacja jest używana, aby przyspieszyć generację wyników, ograniczając potrzebne zasoby do zastosowań w środowisku przeglądarki internetowej. Model projektu Midjourney[19] nie jest zaś publicznie dostępny, choć zgodnie z dostępnymi materiałami udostępnionymi przez twórców jest ulepszany, w ramach cykli kilkomiesięcznych. Dodatkowo porównanie stworzono 31 sierpnia 2022, kiedy model DALL·E 2 znajdował się w jeszcze w fazie zamkniętych testów, jedynie dla wybranych użytkowników – tym samym rezultaty zaprezentowane jako wyniki tego modelu opierają się na upublicznionych rezultatach tych testów[20]. Warto zwrócić również uwagę, że w związku postępem w ramach projektu DALL·E[21] porównanie nie jest aktualne już w momencie publikacji dysertacji – choćby ze względu na istnienie modelu w wersji nowszej, oznaczonej numerem trzy.

[18] Rezultaty wygenerowane na stronie projektu: https://huggingface.co/spaces/dalle-mini/dalle-mini

[19] Strona projektu: https://www.midjourney.com

[20] W tym relacji takich jak: https://youtu.be/Q9FGUii_4Ok oraz publicznych wpisów jak: https://twitter.com/twominutepapers/status/1537871684979642369 oraz https://twitter.com/twominutepapers/status/1538234269683810304

[21] Strona projektu: https://openai.com/dall-e-3

Rysunek 4.9: Obrazy wygenerowane na podstawie zdania *Colorless green ideas sleep furiously* przez wybrane modele I2TG.

Rysunek 4.10: Obrazy wygenerowane na podstawie zdania *Schrödinger's cat gripping the latest research papers* przez wybrane modele I2TG.

Rysunek 4.11: Obrazy wygenerowane dla zdania *Writing a PHD dissertation as a form of suffering in Beksiński style* przez wybrane modele I2TG.

Mimo tych wad można zaobserwować kilka ciekawych spostrzeżeń: a) Na rysunku 4.10, można zauważyć wyraźną różnicę pomiędzy modelami. DALL·E mini miał problem z wygenerowanym samego kota. Model Midjourney poprawnie wygenerował zarówno kota, jak i pracę naukową, ale miał problem z poprawnym odwzorowaniem relacji pomiędzy nimi (kot nie trzyma pracy naukowej, zamiast tego znajduje się na niej). Tymczasem model DALL·E 2 był w stanie oddać tę relację między obiektami. b) Na rysunku 4.11, można zauważyć, że modele były w stanie choć częściowo oddać styl charakterystyczny dla twórczości artysty Zdzisława Beksińskiego (choćby przez zastosowaną paletę kolorów, czy też dystopijny charakter wygenerowanych obrazów), co sugeruje dostęp do informacji o twórczości znanego artysty w danych wykorzystywanych podczas treningu. c) Pomimo wyraźnej sprzeczności w ramach zdania przedstawionego na rysunku 4.9 – żaden z modeli nie miał problemu aby wygenerować wynik choć częściowo związany z podanym zdaniem. Jest to tym bardziej zaskakujące, że tekst ten stanowi klasyczny przykład zdania semantycznie poprawnego ale pozbawionego sensu (choćby przez zaprzeczenie pomiędzy wymaganiem koloru zielonego, przy jednoczesnym oczekiwaniu braku koloru), zaproponowanego przez językoznawcę Noama Chomskiego. Można jedynie spekulować czy jest to związane z dostępem do informacji o sztuce w danych treningowych, a tym samym wykorzystaniu abstrakcyjnych, czy wręcz surrealistycznych połączeń pomiędzy obrazami, a ich „artystycznymi" tytułami.

Przed zakończeniem części tekstu poświęconej uczeniu głębokiemu, warto jeszcze omówić kwestię dotyczącą momentu pojawienia się tej techniki. W wielu ze źródeł (w tym: [56, 162, 314]) znaleźć można informacje, że początek wykorzystania dużych (czy głębokich jak w wypadku sieci neuronowych) modeli nie zależał od przypadkowych okoliczności. Źródło [56] wprost zadaje pytanie czemu uczenie głębokie stało się popularne dopiero po 2012 roku, skoro konwolucyjne sieci neuronowe i algorytm propagacji wstecznej był już dobrze znany w 1989 roku? Za kluczowe uznaje się rozwój technologiczny oraz wystąpienie wspierających warunków, takich jak:

- Dostęp do coraz większych i liczniejszych zbiorów danych – cyfryzacja spowodowała w sposób naturalny pojawienie się rosnącej ilości danych, które można poddać analizie (np. zapisów zachowań użytkowników), co pozwala następnie wyciągać z nich informacje przekładające się na realne zyski finansowe, napędzając tym samym proces wdrożenia technologii. Innym przykładem w ramach tego punktu są coraz bardziej imponujące rozmiary zbiorów danych wykorzystywanych w treningu modeli – w tym ogromnych korpusów językowych wykorzystywanych w treningu LLM.

- Rosnąca dostępność coraz większych zasobów obliczeniowych – w tym obliczeń chmurowych – zmniejsza „cennę" jaką trzeba płacić za konieczność dłuższego trenowania dużych modeli. Do tego punktu zaliczyć trzeba również rozwój powszechnie dostępnych kart graficznych. Choć pierwotnie zostały zaprojektowane do przetwarzania obrazów i renderowania grafiki (również w grach komputerowych) – karty te świetnie sprawdzają się w wykonywaniu równoległych operacji na macierzach, a operacje te są podstawą w treningu sieci neuronowych. Dodatkowo rosnąca popularność uczenia głębokiego doprowadziła do powstania sprzętu dedykowanego wprost do uczenia maszynowego i operacji macierzowych, dostosowanego do zastosowań profesjonalnych w większej skali (np. seria A od Nvidii)[22].
- Dostępność technologii i łatwość wykorzystania uczenia maszynowego – dzięki bibliotekom takim jak *TensorFlow*, *PyTorch*, *Keras* nie ma potrzeby implementacji sieci neuronowych od „zera" (a tym samym zredukowano liczbę możliwych do popełnienia błędów i przyspieszono każdy z projektów o ten właśnie wstępny etap). Zmniejszono próg wejścia z konieczności posiadania znacznej wiedzy specjalistycznej (w tym np. języka C++ oraz znajomości technologii CUDA) do podstaw języka wysokiego poziomu jak Python i wiedzy o właściwym ułożeniu gotowych „bloków sieci neuronowych", w procesie wręcz analogicznym do składania klocków LEGO. Tym samym dostęp do uczenia maszynowego stał się możliwy dla znacznie szerszego grona osób – proces ten określany jest jako „demokratyzacja uczenia głębokiego" [56]. Stąd też łatwiej wykorzystać uczenie maszynowe w skali startupów czy małych projektów badawczych, niekoniecznie związanych bezpośrednio z informatyką. Z innej, częściowo brutalnie biznesowej perspektywy, źródło [162] określa większą dostępność uczenia maszynowego jako swojego rodzaju: *zmniejszenie roli monopolu na najlepszych naukowców, na rzecz pracy sprawnych, ale niekoniecznie wybitnych inżynierów, wdrażających technologię.*
- Rozwój algorytmów – źródło [56] wymienia liczne pomysły (czasami stosunkowo proste), które pomogły w zwalczaniu uciążliwego problemu propagacji gradientu przez warstwy głębokie. W tym: wykorzystanie innych funkcji aktywacji, schematy inicjacji wag, algorytmy dedykowane takie jak wcześniej wspomniany Adam czy RMSProp. Dodatkowo usprawniono proces propagacji gradientu w trakcie treningu, przez metody takie jak: normalizacja wsadowa (ang. *batch normalization*), połączenia szczątkowe (ang. *residual connections*) i konwolu-

[22]Przykład rozwiązań dedykowanych ze strony producenta: https://resources.nvidia.com/l/en-us-gpu. Dostęp na dzień 11.09.2023.

cyjne oddzielenie w zależności od głębokości (ang. *depthwise separable convolutions*). Tego typu usprawnienia (i ich wspólne wykorzystanie) doprowadziły do przejścia od modeli składających się z kilku warstw ukrytych, do możliwości trenowania i praktycznego wykorzystania modeli składających się z ponad setki czy tysięcy warstw ukrytych (przykładem mogą być modele z architektury ResNet, z informacją o liczbie warstw w nazwie, choćby ResNet-152 [107]).

- Nowa fala inwestycji – rozwój sztucznej inteligencji bywa porównywany do kolejnej rewolucji przemysłowej, ze względu na potencjalne korzyści, zmiany i zagrożenia społeczne i gospodarcze jakie może przynieść. Firmy takie jak Meta, Google, Amazon czy Microsoft rywalizują ze sobą na tym froncie o miano lidera, przez działania takie jak inwestycje w badania, pozyskiwanie talentów i rozwijanie nowych produktów opartych o AI. Interesujący w tym kontekście jest konflikt amerykańsko-chiński, a także możliwości chińskich gigantów technologicznych takich jak: Baidu, Alibaba, Tencent do próby „równej" gry z amerykańskimi odpowiednikami [162]. Początkowe przykłady dotyczące zmiany skali finansowania, można przedstawić za [56]: Przed rewolucją uczenia głębokiego w roku 2011, suma inwestycji w rozwój sztucznej inteligencji wynosiła około 19 milionów dolarów. Podczas gdy już w 2013 firma Google przejęła start-up DeepMind za 500 milionów dolarów, a w 2014 Baidu otworzyło centrum badań nad uczeniem głębokim inwestując w ten projekt 300 milionów dolarów. W kontekście tym gorzej wypada Europa nie posiadająca korporacji tej samej skali, szczególnie o tak mocnym zaangażowaniu w rozwój AI. Mimo tego Europa pozostaje ośrodkiem innowacji i badań – czego również dowodem są programy granatowe takie jak: *The Digital Europe Programme (DIGITAL)*[23] czy *Horizon Europe Programme*[24] oraz pionierskie, choć defensywne regulacje: *the AI Act*[25].

Dopiero połączenie wyżej wypomnianych warunków doprowadziło do praktycznego wykorzystania rozbudowanych sieci neuronowych, dodatkowo po części zniwelowano znane wcześniej wady tego typu modeli – w tym: wolniejszego procesu treningu i skłonności do nadmiarowego dopasowania. Możliwym stało się praktyczne wykorzystanie większej pojemność modelu o ogromnej liczbie parametrów, znacznie odbiegającej od „klasycznych"

[23] https://digital-strategy.ec.europa.eu/en/activities/digital-programme

[24] https://digital-strategy.ec.europa.eu/en/activities/digital-technologies-and-research

[25] Skrót strategii: https://www.europarl.europa.eu/news/en/headlines/society/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence.

Dostęp do wszystkich wymienionych w punkcie źródeł na dzień 11.09.2023.

odpowiedników. Przykładem może być ponownie ChatGPT, gdzie liczba parametrów modelu liczona jest w miliardach, przy czym dodatkowo różni się (często rosnąc), pomiędzy poszczególnymi wersjami modelu. W tym miejscu warto jednak ponownie podkreślić, że rozwój sztucznej inteligencji historycznie przeplatał się pomiędzy okresami przyspieszonego postępu, połączonego z wręcz nieuzasadnionym optymizmem oraz okresami „zim AI" – momentami stagnacji, zniechęcenia i braku postępów. Choć obecnie DL jest, z nadzieją, postrzegane jako kluczowy katalizator do współczesnej rewolucji w dziedzinie sztucznej inteligencji – przyszłość jak zwykle pozostaje nieprzewidywalna, nawet mimo licznych sukcesów, opisanych zarówno w pracach naukowych, jak i w postaci licznych wdrożeń komercyjnych. Wystąpienie wyzwań i ograniczeń w ramach prac nad DL jest prawie pewne, natomiast pojawienie się w przyszłości kolejnej „zimy AI" (szczególnie przed osiągnięciem AGI) nie byłby niczym zaskakującym.

4.2.3 Dobór hiperparametrów

W ramach procesu uczenia sieci neuronowych należy choć skrótowo poruszyć dość istotną, ale czasem pomijaną przez teoretyków, kwestię: dobór hiperparametrów (czyli parametrów modelu, które muszą być wybrane z góry przed rozpoczęciem procedury treningiu). Choć większość algorytmów optymalizacji również wymaga ustalania parametrów mających wpływ na jakość uzyskanych rezultatów, to sytuacja jest szczególnie dotkliwa w wypadku sieci neuronowych. Wynika to z dużej liczby wspomnianych parametrów (dla przykładu sam algorytm optymalizacji wag sieci neuronowej Adam w popularnym oprogramowaniu `Keras` posiada 4 parametry numeryczne – jeśli dodatkowo uwzględnimy architekturę sieci w tym m.in. rodzaje warstw, liczbę neuronów w każdej z warstw, czy rodzaj wykorzystanej funkcje aktywacji – liczba hiperparametrów zaczyna gwałtownie rosnąć) oraz długiego czasu potrzebnego do treningu pojedynczego modelu (szczególnie problematyczne w wypadku modeli głębokich, gdzie trening modelu może trwać od godzin przez dni a nawet być liczony w miesiącach [21]).

Bazując na źródle [309] podane zostaną jedynie wybrane, stosunkowo popularne metody doboru hiper-parametrów, takie jak:

- Metoda prób i błędów – najprostszy sposób, czyli dobór bazujący na doświadczeniu lub po prostu zgadywaniu hiper-parametrów w ramach sekwencji powtórzeń eksperymentu. Po ocenie modelu, o ile nie zostało spełnione kryterium w postaci np. wyczerpania zasobów czasowych – podejmowana jest decyzja o kolejnej zmianie wartości hiperparametrów. Metoda ta nie jest efektywna z wielu powodów, takich

jak nieliniowe zależności pomiędzy hiper-parametrami, konieczność posiadania wiedzy o metodzie, aby skutecznie dobrać wartości z potencjalnie dużej przestrzeni hiper-parametrów, oraz brak automatyzacji, co spowalnia procedurę przez konieczność manualnych zmian między kolejnymi powtórzeniami badań. Wady te stanowiły motywację do rozwoju metod bardziej zaawansowanych.

- *Grid Search*, GS (przeszukiwanie wykorzystujące siatkę) – prosta metoda automatyzacji procesu przeszukiwania przestrzeni hiper-parametrów. Metoda ta polega na ocenie modelów w oparciu o iloczyn kartezjański zbiorów możliwych wartości hiper-parametrów (w praktyce często ograniczonych do wybranych wartości przez badacza). Zaletą takiego podejścia jest automatyzacja oraz łatwość zrównoleglenia. Wadami: a) wykładniczy wzrost potrzebnych ewaluacji modelu wraz ze wzrostem liczby hiper-parametrów b) brak wykorzystania informacji o obiecujących regionach przestrzeni c) konieczność powtarzania procesu ze stopniowo coraz mniejszym krokiem w wypadku poszukiwania wartości zbliżonych do optymalnych dla zdyskretyzowanych wartości ciągłych hiper-parametrów.
- Random Search, RS – przeszukiwanie losowe, podobne do GS, jednak ograniczające wadę związaną z marnowaniem zasobów na nierokujących regionach, spowodowanych regularnością siatki w GS. Metoda polega na wyborze losowych wartości pomiędzy z góry ustalonym zakresem minimalnymi i maksymalnymi wartościami hiper-parametrów – tak by następnie przeprowadzić określoną liczbę powtórzeń badania. Praca [20] wskazuje na przewagę podejścia RS w celu ustalenia hiper-parametrów sieci neuronowych, przy czym kryterium oceny było szybsze osiągniecie porównywalnych lub lepszych rezultatów.
- *Bayesian Optimization*, BO – metody bazujące na optymalizacji bayesowskiej – główna zmiana w stosunku do wyżej wymienionych podejść polega na wykorzystaniu rezultatów z wcześniejszych iteracji badań. Metoda ta posiada 2 kluczowe komponenty: model zastępczy (ang. *surrogate model*) oraz funkcję pozyskania (ang. *acquisition function*). Podejście to pozwala na pewien kompromis pomiędzy przeszukiwaniem obiecujących obszarów przestrzeni ale z jednoczesną eksploracją obszarów nieodwiedzonych. Wadą tego podejścia jest sekwencyjność utrudniająca zrównoleglenie. Źródło [309] wymienia ulepszenia tej metody wykorzystujące różne modele: od opartego na procesie gaussowskim, lesie losowym, poprzez estymator Parzen (ang. *Tree-structured Parzen Estimator*). Jednocześnie wskazując metodę *Bayesian Optimization HyperBand*, bazującą na BO jako podejście zalecane ze względu na osiągane rezultaty dla uczenia sieci neuronowych

(w rozumieniu terminu ang. *state-of-the-art*).

- Podejście wykorzystujące klasyczne metody optymalizacji – ze względu na tematykę dysertacji, podejście to nie będzie szerzej opisywane aby uniknąć powtórzeń. W ramach tego podejścia źródło [309] wskazuje na możliwość zastosowania opisanych już metod takich jak algorytmy populacyjne (3.2.3), w szczególności algorytm optymalizacji za pomocą roju cząstek.

Oprócz przedstawionego podejścia „systematycznego", trzeba przedstawić czasami pomijany „truizm": wiele, nawet wybitnych prac naukowych z dziedziny sieci neuronowych [21, 109, 294] unika wyczerpującego doboru hiperparametrów uczonych sieci, szczególnie udokumentowanego wykorzystaniem testów statystycznych. Czasami nie jest to konieczne by uzasadnić główne przesłanie pracy[26], czasami wystarczy jedynie dobrać i wskazać jako „wrażliwe" te parametry które mają duży wpływ na proces treningu [27], a czasami ze względu na skale badań – strojenie nie jest możliwe do uzyskania w „akceptowalnym czasie"[28]. Dodatkowo istnieje postrzeganie zadania doboru hiperparametrów jako „wyzwania czysto inżynierskiego", które z założeń nie prowadzi do „rewolucyjnej" poprawy otrzymanych wyników a stanowi bardzo czasochłonną „walkę o każdy, nawet najdrobniejszy" procent poprawy. Stan ten sprawia, że tematyka doboru hiperparametrów poruszana jest raczej w materiałach skoncentrowanych na osiągnięciu jak najlepszych wyników – np. w ramach zasobów dotyczących popularnych konkursów związanych z uczeniem maszynowym [11, 277], szczególne tych o ustalonych zbiorach danych (gdzie nie można uzyskać poprawy poprzez powiększanie zbioru treningowego) – za przykład może posłużyć serwis `Kaggle`[29]. W przypadku konkursów, kolejną popularną metodą poprawy wyników jest wykorzystanie wielu modeli uczenia maszynowego jednocześnie, poprzez ich połączenia w ramach metod zespołowych (ang. *Ensemble Learning*[235]) – jednak ze względu na ograniczenia co do liczby poruszanych wątków w dysertacji temat ten nie będzie dalej rozwijany.

[26] Za [294]: *Even though there are likely some gains to be obtained by tuning the model, we felt that having the same model hyperparameters operate on all the problems makes the main message of the paper stronger.*

[27] Za [109]: *The combinatorial space of hyper-parameters is too large for an exhaustive search, therefore we have performed limited tuning.*

[28] Za [21]: *As it is impossible for us to scan over any of these hyperparameters when our experiment is so large, we make no claim that the hyperparams used are optimal.*

[29] `https://www.kaggle.com/competitions` dostęp 28.10.2022.

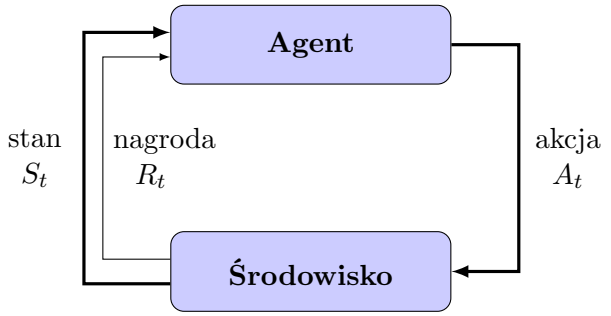
4.3 Uczenie ze wzmocnieniem

Uczenie ze wzmocnieniem (ang. *Reinforcement Learning*, RL) jest odmianą uczenia maszynowego, skupiającą się na podejmowaniu decyzji w warunkach niepewności, tak aby maksymalizować skumulowaną nagrodę. Decyzje są podejmowane przez agenta, który jest modyfikowany w trakcie treningu. Decyzje polegają na wyborze jaką następną akcję wybrać bazując na stanie środowiska, w którym się znajduje. Warto zauważyć, że agent nie posiada bezpośrednich informacji, które akcje są najkorzystniejsze. Musi samodzielnie „odkryć" zależności między akcjami a wynikającymi z nich nagrodami, korzystając z metody prób i błędów (ang. *trial-and-error search*). Co więcej, wykonanie akcji może nieść ze sobą konsekwencje nie tylko związane z określoną natychmiastową nagrodą bądź karą, ale może również doprowadzić do zmiany stanu środowiska, które w nowym stanie może nagradzać lub karać w inny sposób. Tym samym istotne staje się podejmowanie decyzji z perspektywą rozłożoną na przestrzeni wielu iteracji, w celu uwzględnienia odroczonej nagrody (ang. *delayed reward*).

Według [189], agent (ang. *agent*) w kontekście RL to program komputerowy odpowiedzialny za podejmowanie skomplikowanych decyzji w warunkach niepewności. W tym samym źródle zawarto przykład programu trenowanego do sterowania ramieniem robotycznym w celu podnoszenia przedmiotu. Ramię samo w sobie nie stanowi części agenta (pomimo że agent podejmuje decyzje na nie wpływające) – jedynie część kodu odpowiedzialna za podejmowanie decyzji określana jest jako agent.

Następnym podstawowym pojęciem jest środowisko (ang. *environment*) – wszystko, co znajduje się poza agentem, jest środowiskiem. Kontynuując przykład ramienia robotycznego z [189], środowiskiem są: obiekt, który ma być podniesiony; podłoga, na której obiekt się znajduje oraz samo ramię. Choć agent ma kontrolę nad ramieniem i może wymusić jego ruch, między innymi ze względu na istnienie zakłóceń oraz nieidealne odwzorowanie decyzji agenta na stan ramienia, ramię jest częścią środowiska.

Niestety, pełna informacja dotycząca stanu środowiska nie może być w praktyce wykorzystana (po części również dlatego, że nie cały stan środowiska ma znaczenie w ramach określonego zadania). Przez co samo środowisko operacjonalizuje się w postaci zmiennych liczbowych, które je opisują w postaci stanu (ang. *state*). Zbiór wszystkich możliwych stanów tworzy przestrzeń stanów (ang. *state space*). Kontynuując przykład ramienia robotycznego, w ramach pojedynczego stanu można zawrzeć informacje o: lokalizacji ramienia, prędkości przegubów, kątów pomiędzy przegubami oraz położeniu obiektu, który ramie ma podnieść. Natomiast stan środowiska, które agent jest w stanie bezpośrednio postrzegać w postaci danych wejścio-



Rysunek 4.12: Schematyczne przedstawienie podstawowej iteracji w ramach RL: Agent wchodzi w interakcję z środowiskiem poprzez podjęcie akcji A_t, ta akcja prowadzi do zmiany stanu środowiska na stan S_t oraz otrzymaniem przez agenta nagrody lub kary w postaci R_t – proces ten następnie powtarza się w kolejnych iteracjach.

wych, nazywany jest obserwacją (ang. *observation*). Co ważne, informacja w ramach obserwacji może być niepełna lub być zniekształcona wobec faktycznego stanu środowiska. Kontynuując przykład z [189], agent może nie mieć dostępu do bezpośrednich wartości lokalizacji czy prędkości ramienia robotycznego, a mieć jedynie dostęp do pośredniej reprezentacji stanu środowiska – np. jedynie wartości pikseli z obrazu kamery skierowanej na ramie robotyczne oraz obiekt, który należy ramieniem podnieść.

Uczenie ze wzmocnieniem służy do rozwiązywania problemów, które można opisać za pomocą agenta podejmującego decyzje w ramach środowiska, a decyzje podjęte przez agenta wiążą się z otrzymywaniem sygnału nagrody (co schematycznie pokazano na rysunku 4.12). W przeciwieństwie do innych typów uczenia maszynowego, agent otrzymuje pewną informację o jakości podejmowanych decyzji jedynie w postaci nagrody. Nie jest to więc przypadek uczenia nienadzorowanego, gdzie nie ma zupełnie informacji o oczekiwanym wyjściu (w tym wypadku agent nie miałby dostępu do informacji o nagrodzie). Jednocześnie nie jest to przypadek uczenia nadzorowanego, gdzie przy treningu sieci wykorzystuje się pary: oczekiwane wejście oraz przypisane oczekiwane wyjście (w tym wypadku agent trenowany nie byłby przez sygnał nagrody, ale przez przykłady idealnego zachowania w każdym z kroków decyzyjnych). Tym samym uczenie ze wzmocnieniem można stosować w przypadku środowisk, gdzie nieznane są konkretne kroki konieczne do osiągnięcia zadanego celu. Znany jest zamiast tego co najmniej sposób oceny finalnego rezultatu lub pewien pośredni sygnał nagrody lub kary. Przykładem pierwszej z możliwości jest nagradzanie agenta za rezultat rozgrywki, np. partii szachowej. Natomiast druga z opcji jest naturalnym

wyborem na przykład przy treningu agenta sterującego robotem mającym dotrzeć do zadanego celu. Można wtedy dostarczać niewielką nagrodę za każdym razem gdy robot stopniowo zbliża się do celu. Skuteczną może okazać się również odwrotna strategia, czyli dostarczanie stałej negatywnej nagrody (czyli kary) za każdy krok czasowy. Tworzy to zachętę agenta do jak najszybszego wykonania zadania i tym samym minimalizacji skumulowanej kary.

Jednym z największych wyzwań związanych z uczeniem ze wzmocnieniem jest dylemat pomiędzy wykorzystaniem zdobytej wiedzy, a eksploracją środowiska w celu jej pozyskania (ang. *exploration and exploitation dilema*). W celu maksymalizacji skumulowanej nagrody, agent musi preferować akcje które w przeszłości okazały się efektywne. Jednak aby móc „odkryć" które akcje są efektywne, musi najpierw je wykonać i estymować powiązaną z nimi nagrodę. Tym samym, próba nadmiernej priorytetyzacji jedynie jednego z tych celów prowadzi do gorszego zachowania agenta. Co więcej, w wypadku zadań stochastycznych, akcje muszą być podejmowane wielokrotnie aby estymacja nagrody była w ogóle możliwa. Za [266], dylemat pomiędzy wykorzystaniem zdobytej wiedzy, a eksploracją, był badany przez matematyków na przestrzeni dekad. Tym niemniej ciągle pozostaje nierozwiązany (bez przyjmowania silnych założeń), a jego występowanie jest koleją cechą odróżniającą uczenie ze wzmocnieniem od innych rodzajów uczenia maszynowego (uczenia nadzorowanego oraz uczenia nienadzorowanego).

4.3.1 Formalizm matematyczny

Bazując na podręczniku akademickim autorstwa R.S. Suttona i A.G. Barto [266], przedstawiony zostanie formalizm matematyczny opisujący uczenie ze wzmocnieniem[30]. Znaczną część technik RL formalizuje się za pomocą modelu procesów decyzyjnych Markowa (ang. *Markov Decision Processes*, MDP) z dyskretnym czasem. Podejście oparte na skończonych MDP umożliwia rozwiązanie znacznego zakresu problemów uczenia ze wzmocnieniem, wykorzystując zapis matematyczny typowy dla klasycznego podejścia do sekwencyjnych, dyskretnych procesów decyzyjnych.

Agent oraz środowisko wchodzą w interakcje uporządkowane sekwencją kroków czasowych $t = 0, 1, 2, 3, \ldots$. Dla każdego kroku czasowego t, agent otrzymuje informację o stanie środowiska $S_t \in \mathcal{S}$, na podstawie którego podejmuje akcję $A_t \in \mathcal{A}(s)$ oraz $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}(s)$. W następnym kroku czasowym, jako konsekwencja podjętej akcji, agent otrzymuje nagrodę $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ oraz informację o nowym stanie środowiska S_{t+1}.

[30] Przedstawiony paragraf tekstu jest tłumaczeniem własnym i streszczeniem treści z rozdziału 3 przytoczonego źródła z komentarzem.

Interakcja ta w ramach MDP prowadzi do postania sekwencji

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots,$$

nazywanej też trajektorią. W ramach skończonego MDP, zbiory stanów, akcji i nagród składają się ze skończonej liczby elementów. Dodatkowo zmienne losowe R_t oraz S_t mają zdefiniowany dyskretny rozkład prawdopodobieństwa zależny wyłącznie od staniu i akcji w poprzednim kroku decyzyjnym. Tym samym, dla określonych wartości zmiennych losowych $s' \in \mathcal{S}$ oraz $r \in \mathcal{R}$, istnieje prawdopodobieństwo ich wystąpienia w kroku czasowym t, uzależnione od wartości poprzedniego stanu i akcji,

$$p(s', r \mid s, a) \stackrel{\text{def}}{=} \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \quad \forall s', s \in \mathcal{S}, r \in \mathcal{R},\ \forall a \in \mathcal{A}(s). \tag{4.7}$$

Tak zdefiniowana funkcja $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0; 1]$ określa dynamikę MDP i zadaje rozkład prawdopodobieństwa następnego stanu oraz powiązanej z nim nagrody dla obecnego stanu s oraz akcji a, a więc

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \quad \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}(s). \tag{4.8}$$

Jak wcześniej wspomniano, W ramach MDP, prawdopodobieństwa określone dane przez p całkowicie charakteryzują dynamikę środowiska. Co za tym idzie, prawdopodobieństwa każdej pary wartości przyjętych przez S_t oraz R_t zależą wyłącznie od wartości z poprzedniego kroku czasowego $t-1$, czyli wartości przyjętych przez S_{t-1} oraz R_{t-1}; przy czym nie zależą od jeszcze wcześniejszych wartości stanów i akcji. Oznacza to, że stan musi stanowić jedyną (i pośrednią) informację o wszystkich wcześniejszych aspektach interakcji agenta z otoczeniem, które mają znaczenie dla przyszłych decyzji agenta. Jeśli tak jest, to stan spełnia własność Markowa. Należy przy tym podkreślić, że choć wiele metod zakłada konieczność spełnienia własności Markowa, to nie jest to założenie powszechne. Część metod uczenia ze wzmocnieniem obchodzi to wymaganie poprzez metody przybliżone oraz konstrukcję stanów spełniających własność, ale bazujących na obserwacjach które nie spełniają własności Markowa (przykładem może być wprowadzenie zadań pomocniczych (ang. *auxiliary tasks*) oraz wykorzystanie funkcji GVF (ang. *General Value Function*). Tym niemniej, ta tematyka wykracza poza zakres dysertacji.

Bazując na funkcji p, możliwe jest wyliczenie: a) prawdopodobieństwa przejścia pomiędzy stanami

$$p(s'|s, a) \stackrel{\text{def}}{=} \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a), \tag{4.9}$$

b) oczekiwanej nagrody dla pary stanu i akcji – jako dwuargumentowej funkcji $r : S \times A \to \mathbb{R}$,

$$r(s,a) \stackrel{\text{def}}{=} \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a), \tag{4.10}$$

oraz c) oczekiwanej nagrody dla trójki: stan, akcja, następny stan – jako trójargumentowej funkcji $r : S \times A \times S \to \mathbb{R}$,

$$r(s,a,s') \stackrel{\text{def}}{=} \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)}. \tag{4.11}$$

W ramach uczenia ze wzmocnieniem, celem agenta jest maksymalizacja skumulowanej nagrody. Tym samym, agent nie powinien priorytetyzować wyłącznie wysokiego sygnału nagrody w pojedynczym korku czasowym t, a raczej wartość oczekiwaną sumy przyszłych nagród w całej rozważanej trajektorii od momentu t do momentu T,

$$G_t^T \stackrel{\text{def}}{=} R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \tag{4.12}$$

Zabieg ten jest wykorzystywany szczególnie w tych przypadkach, gdzie można wyróżnić naturalne „zakończenie" trajektorii. To znaczy tam, gdzie interakcje agent-środowisko można podzielić na podsekwencje, nazywane epizodami – np. pojedyncza partia w ramach gry w szachy. Tym samym w ramach zadań epizodycznych (ang. *episodic tasks*) – epizod kończy się po osiągnięciu specjalnego stanu terminalnego, po którym następuje reset środowiska do stanu początkowego lub do stanu wylosowanego z dopuszczalnych stanów początkowych. Choć epizod może zakończyć się na wiele sposobów (kontynuując przykład: przegraną, remisem lub wygraną w ramach partii szachowej), nie wpływa to w żaden sposób na epizod kolejny – po prostu następny epizod nie zależy od swojego poprzednika.

Istnieją środowiska w ramach których nie można wyróżnić naturalnych podsekwencji, a które mogą tworzyć nieskończoną trajektorię (ang. *continuing tasks*). Wtedy niecelowe może być korzystanie ze wzoru (4.12), ze względu na potencjalną skumulowaną nieskończoną wartość nagrody spowodowaną nieskończoną liczbą kroków czasowych. W ramach zadań ciągłych zamiast tego często wykorzystuje się zdyskontowaną nagrodę (ang. *discounted return*)

$$G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{4.13}$$

gdzie γ to współczynnik dyskontowania jest parametrem z przedziału $[0; 1]$. Nagroda otrzymana w k-tym kroku czasowym w przyszłości staje się równoważna γ^{k-1} swojej wartości gdyby została otrzymana natychmiast. Wyrażenie (4.13) przyjmuje skończoną wartość dla $\gamma < 1$, pod warunkiem ograniczonej wartości nagrody $\{R_k\}$ w każdym kroku czasowym. Dla $\gamma = 0$, agent powinien zachowywać się w sposób zachłanny, podejmując akcję A_t maksymalizującą jedynie najbliższą otrzymaną nagrodę R_{t+1}. Dla wartości γ bliskich 1, agent powinien bardziej brać pod uwagę nagrodę z kolejnych kroków czasowych – tym samym zachowanie agenta przy takim sygnale nagrody powinno być bardziej „dalekowzroczne".

Znaczna część algorytmów uczenia ze wzmocnieniem wykorzystuje oszacowanie funkcji wartościującej (inaczej *funkcja wartości*, z ang. *value functions*). Jest to funkcja pozwalającej oszacować na ile pożądane jest przebywanie agenta w danym stanie środowiska. Jest to możliwe na podstawie oceny oczekiwanych nagród powiązanych z akacjami, które można wykonać z danego stanu. Aby określić jakich przyszłych nagród może oczekiwać agent, konieczna jest informacja o tym jakie akcje podejmie agent w danym stanie. Definiuje je polityka wykorzystywane przez agenta. Tym samym funkcje wartościujące są określane między innymi na podstawie tego, jaką politykę wykorzystuje agent.

Polityka (ang. *policy*)[31] to funkcja przyporządkowująca prawdopodobieństwo wyboru każdej możliwej akcji dla danego stanu środowiska. Jeśli agent wykorzystuję politykę π w chwili czasowej t, to $\pi(a \mid s)$ jest prawdopodobieństwem, że $A_t = a$ jeśli $S_t = s$. Metody uczenia ze wzmocnieniem określają, jak polityka agenta zmienia się w rezultacie pozyskiwania nowych informacji o środowisku na skutek eksploracji.

Funkcja wartościująca dla stanu s, w ramach polityki π, zapisywana jest jako $v_\pi(s)$ i określa oczekiwaną skumulowaną nagrodę, którą agent będzie w stanie zebrać od stanu s, a następnie wykorzystując politykę π,

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\Big|\, S_t = s\Big] \quad \forall s \ \in \mathcal{S}, \tag{4.14}$$

gdzie $\mathbb{E}_\pi[\cdot]$ oznacza oczekiwaną wartość zmiennej losowej, przy założeniu, że agent wykorzystuję politykę π, dla dowolnego kroku czasowego t. Funkcja v_π nazywana jest funkcją wartościującą stan w ramach polityki π (ang. *state-value function for policy* π). Wyznaczenie wartości funkcji v_π może być w praktyce niemożliwe i konieczne jest posługiwanie się oszacowaniem.

[31] Intuicyjnym tłumaczeniem byłby termin strategia – jednak to termin polityka występuje w polskiej literaturze, np. w źródle [87].

W stanie środowiska $s \in \mathcal{S}$, wartość podjęcia akcji $a \in \mathcal{A}(s)$ i dla polityki π opisuje funkcja

$$q_\pi(s,a) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\Big|\; S_t = s, A_t = a\Big]. \tag{4.15}$$

Tak zdefiniowana funkcja q_π jest nazywana funkcją wartościujacą akcję dla polityki π (ang. *action-value function for policy* π).

Zarówno funkcje v_π oraz q_π mogą być estymowane na podstawie doświadczenia w trakcie eksploracji środowiska. Jeśli agent wykorzystuje politykę π i zapisuje średnią wartość nagród uzyskanych z każdego stanu s, to te średnie powinny zbiegać do $v_\pi(s)$ wraz ze wzrostem liczby odwiedzeń s. W podobny sposób można oszacować $q_\pi(s,a)$. Metody estymacji należące do tej rodziny określa się metodami Monte Carlo, ponieważ wykorzystują uśrednianie wartości wielu losowych prób otrzymanej nagrody. Należy jednak podkreślić, że wraz ze wzrostem liczy stanów środowiska, koszt pamięciowy przechowywania oddzielnych średnich dla każdego ze stanów środowiska wzrasta i przestaje być możliwy w praktyczniej realizacji. W takim wypadku wykorzystuje się raczej aproksymację v_π oraz q_π, szczególnie w ramach charakterystycznej dla uczenia ze wzmocnieniem nauki „online", tak aby uzyskać politykę π która pozwala podjąć dobre decyzje dla często odwiedzanych stanów, nawet kosztem gorszych decyzji w stanach rzadziej odwiedzanych.

Przykładem polityki, może być polityka zachłanna (ang. *greedy policy*), prowadząca do wyboru jedynie tej akcji, która maksymalnie zwiększy potencjalną przyszłą nagrodę, estymowaną na podstawie dotychczas zebranej wiedzy. Warto zwrócić uwagę, że tego typu polityka nie pozwala eksplorować środowiska, co może doprowadzić do pominięcia (czy raczej nie odkrycia) obiecujących par akcja-stan. Dlatego równie prostym – ale bardziej praktycznym – jest podejście oparte o politykę ϵ-zachłanną. W wypadku tej polityki, z prawdopodobieństwem $1 - \epsilon$, podejmowana jest akcja zachłanna – ale z prawdopodobieństwem ϵ akcja losowa. Tym samym, agent zachowuje pewien balans pomiędzy wykorzystaniem zdobytej wiedzy, a eksplorację środowiska w celu jej pozyskania. Prawdopodobieństwo $\epsilon \in [0,1]$ staje się parametrem który należy dobrać. Istnieje wiele wykorzystywanych w praktyce metod doboru ϵ, na przykład ustalenie dużej wartości ϵ na początku treningu agenta, by stopniowo obniżać wartość tego parametru wraz z kolejnymi iteracjami.

Fundamentalną własnością funkcji wartościujących wykorzystywanych w uczeniu ze wzmocnieniem (jak i programowaniu dynamicznym) jest to, że spełniają one rekurencyjne zależności podobne do tych przedstawionych dla

oczekiwanej nagrody. Dla każdej polityki π i dowolnego stanu $s \in \mathcal{S}$, zachodzi następujący warunek spójności pomiędzy wartością stanu s, a wartością stanów następujących po nim:

$$\begin{aligned} v_\pi(s) &\stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \Big] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_\pi(s') \Big]. \end{aligned} \tag{4.16}$$

Wzór (4.16) jest równaniem Bellmana dla v_π (ang. *Bellman equation for* v_π) i przedstawia zależność pomiędzy wartością stanu oraz stanów będących jego następcami. Wzór ten pozwala na uśrednienie oczekiwanej nagrody po ważonym prawdopodobieństwem wystąpienia stanu, łącząc wartość stanu początkowego z zdyskontowaną wartością stanów następujących po nim i nagrodą oczekiwaną w ramach przejścia pomiędzy nimi. Funkcja v_π jest unikalnym rozwiązaniem powiązanego równania Bellmana, a metody uczenia ze wzmocnieniem wykorzystują różne sposoby na szacowanie jej wartości lub postaci.

Za cel uczenia ze wzmocnieniem można przyjąć odnalezienie polityki, która pozwala zmaksymalizować zebraną przez agenta nagrodę. Wykorzystując formalizm matematyczny MDP, możliwe jest zdefiniowanie pojęcia optymalnej polityki. Polityka π jest zdefiniowana jako lepsza lub równa polityce π', jeżeli oczekiwana nagroda jest większa lub równa polityce π' dla wszystkich stanów środowiska. Tym samym $\pi \geqslant \pi'$, wtedy i tylko wtedy, jeśli $v_\pi(s) \geqslant v_{\pi'}(s)$ dla każdego $s \in \mathcal{S}$. W ramach danego środowiska istnieje co najmniej jedna polityka lepsza lub równa innym politykom – jest to polityka optymalna (ang. *optimal policy*), oznaczana przez π_*. Jest ona powiązana z optymalną funkcją wartościującą stan v_*, zdefiniowaną jako

$$v_*(s) \stackrel{\text{def}}{=} \max_\pi v_\pi(s) \quad \forall s \in \mathcal{S}. \tag{4.17}$$

Dodatkowo, optymalna polityka ma powiązaną optymalną funkcję wartościującą akcję q_*,

$$q_*(s, a) \stackrel{\text{def}}{=} \max_\pi q_\pi(s, a), \tag{4.18}$$

dla każdego stanu $s \in \mathcal{S}$ oraz $a \in \mathcal{A}(s)$. Dla pary stanu i akcji (s, a), funkcja ta zwraca oczekiwaną nagrodę za podjęcie akcji a w stanie s, a następnie wykorzystywanie optymalnej polityki. Tym samym można zapisać q_* jako

$$q_*(s, a) = \mathbb{E}\Big[R_{t+1} + \gamma v_*(S_{t+1}) \,\Big|\, S_t = s, A_t = a \Big] \tag{4.19}$$

Ponieważ v_* jest funkcją wartościującą w ramach polityki, musi spełniać warunek podany w ramach równania Bellmana dla wartości stanów (4.16). Jednak ze względu, że jest to optymalna funkcja wartościująca v_*, warunek spójności może zostać zapisany w sposób nie odwołujący się do żadnej polityki – tworząc tym samym równanie optymalności Bellmana (ang. *Bellman optimality equation*)

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) && (4.20)\\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] && (4.21)\\
&= \max_a \mathbb{E}_{\pi_*}\Big[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a\Big] && (4.22)\\
&= \max_a \mathbb{E}\Big[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\Big] && (4.23)\\
&= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]. && (4.24)
\end{aligned}
$$

Intuicyjnie, wzór ten przedstawia fakt, że wartość stanu w ramach optymalnej polityki, musi być równa oczekiwanej nagrodzie dla podjęcia najlepszej możliwej decyzji z tego stanu. Wzór (4.24) odnosi się do v_*, ale w podobny sposób można zdefiniować równanie optymalności Bellmana dla $q*$,

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a\Big] && (4.25)\\
&= \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')]. && (4.26)
\end{aligned}
$$

Dla skończonych MDP, równanie (4.24) ma unikalne rozwiązanie niezależne od polityki. Równanie to jest w istocie zbiorem równań, po jednym dla każdego stanu i równie licznym zbiorze niewiadomych. Jeśli dynamika środowiska p jest znana, rozwiązanie v_* może być możliwe poprzez wykorzystanie metod rozwiązywania systemów równań nieliniowych. Jeśli v_* jest znane, łatwo określić optymalną politykę, która wybiera akcję dla których oczekiwane jest maksimum w ramach wzoru optymalności Bellmana. Wtedy można powiedzieć, że polityka ta jest zachłanna ale w stosunku do optymalnej wartości funkcji v_*, a więc jednocześnie optymalna. W wypadku $q*$ wybór optymalnej akcji staje się jeszcze prostszy – wystarczy wybierać akcje maksymalizujące $q_*(s,a)$.

Bezpośrednie rozwiązanie równania optymalności Bellmana stanowi jedną z możliwych metod znalezienia optymalnej polityki, a tym samym rozwiązania powiązanego problem uczenia ze wzmocnieniem. Jednak podejście to jest najczęściej niemożliwe do zastosowania, poprzez swoje podobieństwo

do przeszukiwania zupełnego, w tym ekstremalnie dużych kosztów obliczeniowych i pamięciowych. Dodatkowo w praktyce występują trudności takie jak: a) brak pełnej informacji o dynamice środowiska b) konieczność spełnienia własności Markowa. Za oczywisty przykład środowiska o znanej dynamice ale ogromnej przestrzeni stanów, dla której nie można zastosować rozwiązania optymalnego, jest gra w szachy. Tym samym w praktyce wykorzystywane są metody przybliżonego rozwiązywania równania optymalności Bellmana, z wykorzystaniem przybliżonych wartości prawdopodobieństw – szacowanych na podstawie przeszłych doświadczeń agenta w ramach określonego środowiska. Dodatkowo, część z metod RL wykorzystuje unikalne rozwiązania aby osiągnąć przydatne przybliżenia. Na przykład udaje się osiągnąć zachowanie agenta zbliżone do optymalnego pomimo wyboru nieoptymalnych akcji w stanach rzadko odwiedzanych (o niskim wpływie na łączną skumulowaną nagrodę). Zarówno wykorzystanie metod przybliżonych (takich jak przytoczona), jak i możliwość treningu „online" agenta przy jednoczesnej eksploracji i poznawaniu środowiska, są cechami wyróżniającymi uczenie ze wzmocnieniem od innych metod rozwiązywania MDP.

4.3.2 Wybrane algorytmy uczenia ze wzmocnieniem

Kolejna część rozdziału pokrótce przybliży dwa wybrane algorytmy uczenia ze wzmocnieniem. Tym samym pominięty zostanie wnikliwy opis zróżnicowanych metod należących do rodziny RL. Bardziej kompletny przegląd algorytmów tego typu można znaleźć między innymi w [189, 266, 314]. Wybrane algorytmy pozwolą na pokazanie ważnego, szczególnie w kontekście dysertacji, etapu rozwoju uczenia ze wzmocnieniem – t.j. połączenia sztucznych sieci neuronowych oraz uczenia ze wzmocnieniem.

Q-Learning

Zgodnie z [266], ważnym krokiem w rozwoju uczenia ze wzmocnieniem było powstanie metod różnic czasowych (ang. *temportal-difference*, TD). Metody te stanowią swojego rodzaju połączenie idei Monte Carlo oraz programowania dynamicznego. Podobieństwo do metod Monte Carlo polega na początkowym braku wiedzy o dokładnym modelu rozwiązywanego środowiska – agent może być trenowany na podstawie własnych doświadczeń, w tym przez początkowe podejmowanie losowych akcji (np. w ramach polityki: ϵ-zachłannej). Tym samym uwzględniony jest komponent prób i błędów. W wypadku programowania dynamicznego, rozwiązanie optymalne budowane jest na podstawie optymalnych rozwiązań podproblemów (problemy, które można w ten sposób rozwiązać, posiadają własność optymalnej podstruk-

tury). W podobny sposób metody TD polegają na aktualizacji estymacji wartości stanów, w oparciu o estymację stanów następujących po nich, bez konieczności „rozwiązania" całego problemu na raz.

Do rodziny TD należą między innymi metody takie jak *Q-Learning* [298, 299] oraz SARSA [231] wraz z modyfikacjami [288]. Zostanie jednak omówiona pokrótce jedynie pierwsza z nich – aby zobrazować rolę *Q-Learningu* w rozwoju RL – jako fundamentu dla metody omówionej w następnej sekcji, czyli *DQN*.

Metoda *Q-Learning* opiera się o procedurę aktualizacji funkcji Q,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]. \quad (4.27)$$

Trenowana funkcja Q stanowi przybliżenie q_*, czyli wzoru optymalności Bellmana dla $q*$ (wzory (4.25) i (4.26)), i jest w zasadzie niezależna od wykorzystanej polityki. Przy czym należy jednak zaznaczyć, że wykorzystana podczas treningu polityka wpływa na to, które z par stan-akcja i w jakiej kolejności zostaną odwiedzone, a ich wartość Q zaktualizowana. Tym samym wybór polityki początkowej wciąż pozostaje istotną decyzją, ze względu na wymóg aktualizacji wszystkich par stan-akcja. Przy spełnieniu tego założenia oraz zapewnieniu odpowiedniej wielkości kroku α, w [298] wykazano zbieżność Q do q_* z prawdopodobieństwem 1.

Należy zwrócić uwagę, że funkcja Q jest implementowana w praktyce jako zwykła tabela (określana jako tabela Q, od ang. *Q-Table*) o rozmiarze odpowiadającym liczbie możliwych akcji na liczbie możliwych stanów. W wypadku środowisk o małej przestrzeni stanów i akcji nie jest to problematyczne, jednak wraz ze wzrostem skomplikowania środowisk, w szczególności pojawieniem się stanów rzadko odwiedzanych, trening agenta zaczyna stawać się wyzwaniem. Dużą rolę odgrywa przy tym typowy proces trenowania agenta (uproszczony pseudokod algorytmu Q-Learning został na schemacie 2). Po resecie środowiska agent zaczyna w stanie startowym. Następnie, aż do osiągnięcia stanu końcowego – następuje pętla: agent podejmuje akcję zgodnie z obowiązującą polityką, co prowadzi do otrzymania nagrody i przejścia do kolejnego stanu. Po tym następuje aktualizacja funkcji Q i pętla się powtarza. Tym samym pojedyncza trajektoria (przejście agenta od stanu początkowego do końcowego) składa się głównie z aktualizacji często odwiedzanych par stan-akcja. Podkreślany wymóg aktualizacji wszystkich par stan-akcja, w praktyce oznacza wydłużenie czasu treningu agenta przez konieczność zebrania większej liczby trajektorii – tak, aby również rzadsze pary stan-akcja zostały odwiedzone, dzięki losowemu komponentowi polityki, a tym samym zostały uwzględnione podczas aktualizacji funkcji Q.

Algorytm 2: Uproszczony schemat algorytmu Q-Learning, tłumaczenie własne ze źródła [266]

Wejście: parametr $\alpha \in (0, 1]$ – współczynnik uczenia;
parametr $\gamma > 0$ – współczynnik dyskontowania;
polityka spełniająca założenia i jej parametry;
środowisko – model ze zdefiniowanymi stanami oraz akcjami przejścia pomiędzy nimi.

Wyjście: Tabela $Q(S, A)$ – ze zmienionymi wartościami, dostosowanymi do rozwiązywania zadanego środowiska.

1 Zainicjalizuj tabele $Q(s, a) \leftarrow 0$ dla każdego $s \in S^+$, $a \in A(s)$, poza stanami terminalnymi;
2 **for** *każdego epizodu* **do**
3 Rozpocznij od nowego stanu początkowego S;
4 **do** `// Dla każdego kroku w epizodzie`
5 Wybierz akcję A w stanie S przy pomocy polityki opartej o Q (np. może być to polityka ϵ-zachłanna);
6 Podejmij akcję A, otrzymując informację o powiązanych: nagrodzie R oraz kolejnym stanie S';
7 Uaktualnij tabelę, wykorzystując pozyskane informacje: $Q(S, A) \leftarrow Q(S, A) + \alpha\,[R + \gamma \max_a Q(S', a) - Q(S, A)]$;
8 Przejdź do kolejnego stanu, zgodnie z podjętą akcją: $S \leftarrow S'$;
9 **while** *stan S nie jest stanem terminalnym*;
10 **return** *Zmieniona tabla $Q(S, A)$.*

W tym miejscu podkreślić można praktyczne wyzwanie w implementacji metod RL, jakim jest zastosowanie wiedzy dziedzinowej o środowisku, tak aby zmniejszyć liczbę stanów i akcji – co pod warunkiem wybrania dobrej reprezentacji, prowadzi do uzyskania lepszych rezultatów (w tym wręcz decyduje o braku lub możliwości rozwiązania bardziej skomplikowanego środowiska[32]). Jednocześnie, jednak stanowi swojego rodzaju zaprzeczenie idei o treningu agenta bez wcześniejszej wiedzy o środowisku.

DQN

Klasyczne metody RL – w tym Q-Learning – charakteryzują się dość poważnymi ograniczeniami związanymi z koniecznością manualnego wybra-

[32] Za przykład wpływu doboru reprezentacji mogą służyć prace zespołu Tesauro nad grą Tryktrak, za źródłem [266].

nia sposobu reprezentacji stanów środowiska, aby ułatwić proces treningu agenta (czy nawet umożliwić osiągnięcie zachowanie zbliżonego do poprawnego). Jest to związane z lepszym działaniem klasycznych metod dla środowisk w pełni obserwowalnych, o niskowymiarowej przestrzeni stanów.

W tym kontekście ważnym postępem, stanowiącym przykład rozpoczęcia trendu wykorzystania (głębokich) sieci neuronowych jako części składowej agenta, stała się metoda DQN [188] (od ang. *Deep Q-network agent*). Potencjalną zaletą tego podejścia jest potencjalna możliwość wyeliminowania manualnego doboru reprezentacji stanów. Metoda DQN została zaprezentowana w ramach środowisk symulujących 49 prostych – jak na obecne standardy – gier z konsoli Atari 2600 (urządzenie z roku 1977)[33]. Zamiast „ułatwić" trening agenta poprzez dostosowanie stanów środowiska – wykorzystano konwolucyjne sieci neuronowe jako komponent agenta (zapewne, z powodu sukcesu wykorzystania sieci tego typu w ramach przetwarzania i analizy obrazów). Wybór ten pozwolił na trening zachowania agenta bazując bezpośrednio na wysoko wymiarowych danych wejściowych: wartościach pikseli w kolejnych klatkach obrazu gry (dokładnie o rozmiarze 210 na 160 pikseli, przy odświeżaniu 60Hz). Tym samym agent otrzymywał dane o stanie w sposób zbliżony do odbioru środowiska przez ludzkiego gracza (choć bez wcześniejszej wiedzy typowego gracza). Sygnał nagrody był zaś powiązany z punktacją poszczególnych gier.

Wcześniejsze próby wykorzystania sieci neuronowych jako aproksymacji funkcji q_* (wzór 4.25) nie były udane, ze względu na niestabilny proces uczenia. Problemy spowodowane były między innymi przez korelacje występującą w danych, złożonych z obserwacji kolejnych kroków czasowych tej samej trajektorii. Prowadzi to do sytuacji, gdzie nawet niewielkie zmiany Q mogą prowadzić do zbyt dużych zmian polityki. Problem ten ograniczono za pomocą dwóch usprawnień: a) wykorzystano bufor pamięci, przechowując wcześniej zebrane dane, by następnie pobierać je w sposób losowy; b) zmieniono sposób aktualizacji oczekiwanej wartości Q, tak aby nie zmieniać jej dla każdej iteracji treningu.

Metoda DQN przybliża q_* (wzór 4.25) wykorzystując głęboką sieć konwolucyjną, przy czym wagi tej sieci zapisywane się jako θ, parametr metody. Wagi te należy optymalizować w trakcie treningu sieci neuronowych. W celu wykorzystania danych, zebranych w ramach interakcji pomiędzy agentem a środowiskiem, zapisuje się „wspomnienia" $e_t = (s_t, a_t, r_t, s_{t+1})$ dla każdego kroku czasowego t w ramach zbioru danych $D = \{e_1, \ldots, e_t\}$. Trening

[33] Symulacje tego typu gier można obecnie w stosunkowo łatwy sposób przeprowadzić dzięki bibliotece Gym: https://www.gymlibrary.dev/environments/atari/. Data dostępu 13.05.2023.

agenta odbywa się w sposób podobny do wcześniej przedstawionej metody *Q-Learning*, jednak w ramach „paczek" (ang. *minibatch*) przeszłych doświadczeń losowanych z jednostajnym prawdopodobieństwem $(s, a, r, s') \sim U(D)$. Trening wykorzystuje funkcję straty (ang. *loss function*), w postaci:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right], \quad (4.28)$$

gdzie θ_i oznacza wagi sieci neuronowej dla iteracji i, a wagi θ_i^- są wykorzystywane aby obliczać wartość oczekiwaną w iteracji i. Wagi θ_i^- są aktualizowane co C kroków (dodatkowy hiperparametr metody DQN), a tym samym utrzymują tą samą wartość na przestrzeni wielu iteracji.

Podejście oparte o metodę DQN pozwoliło uzyskać lepsze rezultaty od wybranych, klasycznych metod RL w aż 43 z 49 gier dla których dokonano porównania. Dodatkowo, dla 29 z 49 gier, uzyskano zachowanie agenta zbliżone lub nawet przewyższając ludzkiego odpowiednika. Przyjęto próg 75% skumulowanej nagrody ludzkiego gracza jako punkt odniesienia.

Sama metoda DQN oczywiście nie rozwiązywała wszystkich wyzwań uczenia ze wzmocnieniem. W ramach testowanych środowisk problematyczne okazały się szczególnie te gry (np. Montezuma's Revenge), dla których uzyskanie nagrody było znacznie odroczone w czasie (nagroda uzyskana dopiero po wykonaniu dłuższej sekwencji poprawnych decyzji). Nie przeszkodziło to jednak w adaptacji kolejnych technik [15, 84, 104, 297] poprawiających możliwość zastosowania sieci neuronowych w połączeniu z RL. Tym samym, wraz z postępami, osiągnięto coraz lepsze wyniki [109] i z biegiem lat stopniowo upowszechniono wykorzystanie sieci neuronowych w uczeniu ze wzmocnieniem.

4.3.3 Zastosowania RL

Analizując literaturę (w tym wykorzystując źródła takie jak: [189, 266]) można z powodzeniem przedstawić przykłady stopniowego rozwoju obszaru uczenia ze wzmocnieniem – w tym sukcesów osiągniętych dzięki połączeniu RL oraz sztucznych sieci neuronowych. Rozwój tego podejścia na przestrzeni 25 lat pozwolił przejść od rozwiązania gry Tryktrak (ang. *Backgammon*, alternatywna nazwa polska: Nardy) [276] – czyli środowiska o przestrzeni rzędu 10^{20} stanów o pełnej informacji. Przez rozwiązanie bardziej skomplikowanych, klasycznych gier takich jak szachy (10^{45} stanów o pełnej informacji) [157, 250] czy GO (10^{170} stanów o pełnej informacji) [249, 252], aż do współczesnych gier strategicznych takich jak StarCraft II (10^{270} stanów o niepełnej informacji, decyzje podejmowane w czasie rzeczywistym)

[293] czy DOTA 2 (zbliżony stopień skomplikowania, jednak oprócz czasu rzeczywistego, potrzeba podejmowania decyzji z uwzględnieniem zachowania innych agentów – zarówno współpracy i rywalizacji w ramach gry 5 na 5) [21]. Wykorzystanie gier w badaniach RL jest naturalne, każda z gier z samej definicji jest ograniczona przez skończoną liczbę reguł, które z powodzeniem można modelować w postaci środowiska RL – zaś decyzję gracza przedstawić jako akcję, którą podejmuje agent. Jednak zastosowanie RL nie ogranicza się wyłącznie do mniej lub bardziej skomplikowanych gier[34].

W ramach prac badawczych [124, 181] z powodzeniem zaprojektowano kontroler pamięci oparty o RL, udało się uzyskać przyspieszenie wykonywania programów w porównaniu do tradycyjnych kontrolerów. Tradycyjne kontrolery wykorzystywały strategie nieuwzględniające zarówno wcześniejszych doświadczeń w harmonogramowaniu, jak i długoterminowych konsekwencji podjętych decyzji. Wykorzystanie własnego kontrolera, z możliwością nauki „online", pozwoliło uzyskać 8% poprawę w stosunku do kontrolera, wykorzystującego ustalony zbiór reguł. Niestety rozwiązanie ze względu na koszt nie zostało wdrożone w praktyce.

W roku 2004 z powodzeniem zastosowano jedną z metod RL (w ramach podejścia opartego o *Policy-gradient*, [149]) aby w sposób automatyczny wytrenować prawidłowy sposób poruszania się robotów (fizycznych urządzeń, a nie samych modeli), które następnie wystawiono do gry w piłkę nożną w ramach konkursu RoboCup. Autorzy podkreślili, że ich podejście pozwoliło osiągnąć szybszy sposób poruszania, co stanowiło oczywistą przewagę – dodatkowo wyraźnie odróżniając się przez wykorzystanie uczenia maszynowego w kontraście do stanu literaturowego, gdzie dominowały algorytmy dedykowane teorii sterowania.

Kolejnym obszarem zastosowania RL jest personalizacja marketingu, w tym dopasowywanie reklam w zależności od profilu upodobań oraz przeszłych zachowań internautów. Uczenie ze wzmocnieniem jest przydatne w tym obszarze ze względu na zdolność do poprawy strategii rekomendacji, dzięki uwzględnieniu informacji zwrotnej w postaci rzeczywistej reakcji użytkownika – np. „kliknięć" w określone reklamy. Już prosta metoda testów A/B, pozwala na zbadanie preferencji użytkowników. Wystarczy zbadać reakcję pomiędzy podobnymi, choć różniącymi się wersjami stron inter-

[34] W tym miejscu jednak warto zaznaczyć przewrotne zastosowanie RL: czyli zmianę parametrów gier komputerowych, w celu dostosowywania ich poziomu trudności, tak aby były przyjemniejsze w odbiorze przez użytkowników, ani zbyt łatwe przez co nudne, ani zbyt trudne przez co wzbudzające frustrację – przykładem może być firma *Riot Games* w materiale https://youtu.be/r6ErUh5sjXQ czy raport *Google* https://ai.googleblog.com/2021/03/leveraging-machine-learning-for-game.html. Dostęp: na dzień 29.04.2023.

netowych w celu dopasowania ich zawartości – np. wyboru, które wiadomości powinie zostać przedstawione, tak aby zmaksymalizować liczbę kliknięć i odwiedzeń [167]. Oczywiście badania tego typu nie kończą się na prostych metodach. Firma *Adobe Inc.* (znana z oprogramowania *Photoshop*) w ramach prac [278], postawiła za cel zmaksymalizowanie liczby kliknięć, ale nie w perspektywie krótkoterminowej (pojedynczej wizyty na stronie internetowej), lecz długoterminowej (na przestrzeni wielu osobnych odwiedzeń strony) – projekt obejmował dostarczenie zautomatyzowanej infrastruktury wykorzystującej wiedzę o użytkowniku. Zadanie to zrealizowano przy użyciu RL oraz sformułowaniu problemu jako MDP – co interesujące, udało się to nawet pomimo wyzwań związanych z wykorzystaniem danych charakteryzujących się trudnościami, takimi jak komponent losowy, wynikający z wysokiej wariancji danych oraz rzadko występująca nagroda (w przeważającej części zapisów użytkownik nie podejmuje interakcji). Wynik okazał się na tyle zachęcający, że został zintegrowany w ramach produktu *Adobe Marketing Cloud*, czyli usługi wspierającej zarządzanie oraz analitykę kampaniami reklamowymi[35].

Podobnym, ale bliskim przykładem wykorzystania uczenia ze wzmocnieniem są działania popularnego w Polsce e-commerce'u *Allegro*, gdzie wykorzystano RL w projekcie, aby dostosowywać treści wyświetlane na stronie internetowej, do preferencji użytkownika[36]. Co interesujące wykorzystano RL w trybie „offline" – co oznacza, że nie symulowano bezpośrednich interakcji między agentem a środowiskiem. Zamiast tego, zastosowano archiwalne zapisy akcji, które rzeczywisty agent podejmował w środowisku. Podejście to jest praktyczne dla firmy, ze względu na łatwą dostępność informacji o przeszłych zachowaniach klientów, jak i kosztowność (i ryzyko) wdrażania rzeczywistych zmian na platformie. Warto zaznaczyć, że aby osiągnąć bliżej niesprecyzowane poczucie „wygody" u użytkowników, firma oprócz RL stosuje uczenie maszynowe na szeroką skalę – zarówno do przetwarzania języka naturalnego, jak i w ramach systemów wizyjnych (w tym do: rankingowania i wybór istotnych przedmiotów, sugerowania podobnych przedmiotów na podstawie zdjęć, scalania zbliżonych ofert na podstawie ich opisów czy też poprawiania źle wypełnionych kategorii).

[35]Implikacje konkurencji o uwagę użytkownika przez reklamodawców, jak i platformy społecznościowe są fascynujące ale wykraczają poza tematykę dysertacji. Istnieją prace przeglądowe [22] obrazujące, że uwaga poświęcona „używkom cyfrowym" ogranicza czas, jaki użytkownik poświęca na obowiązki zawodowe, studiowane, sen, relację z rodziną i znajomymi – co prowadzi do problemów psychologicznych m.in. wzrostu odczuwanego niepokoju, czy nasilenia stanów depresyjnych.

[36]Informacja na podstawie prezentacji pracowników *Allegro* (Jacka Płocharyczka oraz Mishy Zanka): „It ain't much but you can use offline data in RL" podczas konferencji „ML in PL Conference 2021".

Choć zastosowanie sieci neuronowych w ramach uczenia ze wzmocnieniem, nie jest tak powszechne, jak w ramach przetwarzania języka naturalnego czy systemów wizyjnych – to stopniowo nabiera tempa i zaczyna być wykorzystywane do popularyzacji RL w kolejnych obszarach zastosowań np. finansów – gdzie RL może służyć do doboru i rebalansowania portfela inwestycyjnego [217]. Trend ten zapewne będzie się nasilał wraz z kolejnymi usprawnieniami i powstawaniami nowych metod, choćby ze względu na tak prozaiczną zaletę RL, jak brak wymogu manualnego etykietowana danych.

W ramach niniejszego rozdziału zdecydowano się jednak nie przedstawiać kolejnych przykładów obszarów wykorzystania RL, ani nie wymienić sukcesów związanych z zastosowaniem uczenia ze wzmocnieniem do rozwiązywania problemów dyskretnych. Po części ze względu na obszerność rozdziału – ale również ze względu na poruszenie tego drugiego zagadnienia w przeglądzie literatury z rozdziału 9. Zawarty tam opis stanu wiedzy wprowadza do badań własnych autora dysertacji, w których wykorzystano uczenie ze wzmocnieniem.

4.4 Wnioski i uwagi

Rozdział zawiera poglądowy opis wybranych metod i terminów związanych z sieciami neuronowymi – w tym ważnych powiązanych pojęć jak: „uczenie maszynowe" oraz „sztuczna inteligencja". Następnie przybliżono architekturę oraz sposób trenowania sieci neuronowych, wraz z celami i wyzwaniami tej procedury. Dodatkowo opisano wybrany rodzaj uczenia maszynowego, czyli ucznie ze wzmocnieniem, gdzie coraz powszechniejsze staje się wykorzystanie sieci neuronowych.

Ponownie, jak w wypadku pozostałych rozdziałów opisujących stan wiedzy, opracowanie nie jest wyczerpujące. Zostało przemyślane tak, aby przybliżyć – lub choćby wymienić – wybrane terminy powiązane z sieciami neuronowymi, wraz z odwołaniami do obszerniejszych i bardziej wyczerpujących źródeł. Zadanie to było tym trudniejsze, że obszar sztucznej inteligencji jest dziedziną rozwijającą się niezwykle dynamicznie – żartobliwie można wręcz powiedzieć, że większość bieżących materiałów z tego zagadnienia jest przestarzała, już w momencie publikacji.

Rozdział 5

Analiza krajobrazu przestrzeni rozwiązań

Rozdział stanowi wprowadzenie do tematyki związanej z analizą krajobrazu rozwiązań (FLA, z ang. *Fitness Landscape Analysis*). Obok skróconego opisu podstawowych terminów i metryk związanych z tą tematyką, pogłębiony zostanie opis metody sieci lokalnych optimów ze względu na jej wykorzystanie w badaniach autora dysertacji. Opracowanie opiera się w znaczącym stopniu o prace [122, 176, 202, 208]. Cześć polskich terminów i nazw metod stanowi tłumaczenie własne, stąd za każdym razem przytoczono również bardziej ukonstytuowane nazwy angielskie.

5.1 Podstawowe pojęcia

Przestrzeń rozwiązań (ang. *Fitness Landscape*, FL) tworzona jest poprzez przyporządkowanie każdemu z rozwiązań (punktowi w przestrzeni) wysokości, determinowanej przez jakość rozwiązania (czyli na przykład wartość funkcji celu). Tak zdefiniowany obiekt może być interpretowany jako metafora rzeczywistego krajobrazu. Wtedy proces przeszukiwania można przyrównać do poruszania się po pewnym „krajobrazie", na który oczywisty wpływ ma zarówno struktura, jaki i inne cechy tego krajobrazu; w tym: czy powierzchnia jest płaska, czy składa się z wielu szczytów, na ile strome są szczyty oraz czy „droga" pomiędzy szczytami nie cechuje się trudnościami w postaci nagłych ostrych klifów.

Co istotne z perspektywy optymalizacji, metafora przestrzeni rozwiązań nie jest tylko elegancką abstrakcją, ale ma również praktyczne zastosowanie. Znane jest co najmniej kilka cech przestrzeni rozwiązań, które przekładają się na efektywność procesu przeszukiwania z wykorzystaniem heurystyk.

Tym samym możliwe jest zarówno przewidywanie skuteczności procesu przeszukiwania, jak i modyfikacja istniejących algorytmów w celu uzyskania lepszych wyników dla konkretnych instancji problemów. Należy przy tym podkreślić, że trudność tych celów jest zróżnicowana – dla przykładu czasami wykorzystane miary nie są wystarczające do szacowania trudności instancji problemów [191]. Jednak nawet takie przypadki nie zniechęcają badań nad krajobrazem przestrzeni rozwiązań, a sam opis przestrzeni rozwiązań za pomocą metryk może być pierwszym krokiem w dalszych badaniach zarówno dotyczących tworzenia lepszych metryk opisujących przestrzeń jak i nowych algorytmów optymalizacji. Zgodnie z [202], przykładami cech mającymi bezpośredni wpływ na proces przeszukiwania są między innymi: liczba ekstremów lokalnych – liczba szczytów w krajobrazie, ich rozkład, zależność pomiędzy wartościami funkcji celu dla sąsiadujących rozwiązań, topologia zagłębień (lub wzgórz) przyciągających proces przeszukiwania do konkretnego lokalnego ekstremum; czy też istnienie oraz stopień neutralności – gdzie różne rozwiązania mają tą samą wartość funkcji celu.

Zgodnie z [262] – formalnie – przestrzeń rozwiązań to krotka (S, f, N), gdzie S jest zbiorem rozważanych rozwiązań, $N : S \to 2^S$ jest operatorem sąsiedztwa, a funkcja celu $f : S \to \mathbb{R}$ przyporządkowuje wartość „wysokości" każdemu z rozwiązań z S. Dzięki istnieniu sąsiedztwa można też zdefiniować w sposób naturalny odległość pomiędzy rozwiązaniami $d : S \times S \to \mathbb{R}^+$. Wykorzystując przykład z [122], dla algorytmów przeszukiwania lokalnego o binarnym sposobie kodowania rozwiązań, przestrzeń rozwiązań S stanowi boolowską hiperkostkę $B = \{0, 1\}^l$. Przestrzeń składa się z 2^l rozwiązań, możliwych do zapisania jako ciąg znaków o długości l wraz z przypisaną wartością funkcji celu. Sąsiedztwo pojedynczego rozwiązania x jest tworzone poprzez zmianę pojedynczego bitu tworząc zbiór $y \in B$, odległość pomiędzy rozwiązaniami w ramach takiej przestrzeni rozwiązań to odległość Hamminga.

Za [122], bazując na pojęciu sąsiedztwa, można zdefiniować lokalne ekstremum (tutaj – maksimum) jako rozwiązanie x, dla którego spełnione jest $\forall y \in N(x)\ f(y) \leqslant f(x)$. Globalne optimum można zdefiniować podobnie, jako absolutne maksimum (lub minimum) dla całej przestrzeni rozwiązań S. Można też zdefiniować chód, który tworzy ścieżkę p z rozwiązania s_1 do rozwiązania s_m jako sekwencję $p = (s_1, s_2, \ldots, s_m)$ rozwiązań należących do S, gdzie $\forall i \in \{2, \ldots, m\}$, s_i jest sąsiadem s_{i-1}. Chód może być losowy (błądzenie losowe), wtedy rozwiązanie jest wybierane przypadkowo spośród rozwiązań w ramach sąsiedztwa. Rozważane są różne odmiany chodu, zarówno poprzez zmiany określania prawdopodobieństwa przy wyborze rozwiązania z sąsiedztwa, jak i poprzez zastosowanie wielu powtórzonych operatorów

ruchu – metoda ta jest normą w ramach perturbacji rozwiązania, tak aby opuścić ekstremum lokalne (ang *kick operator*).

W ramach analizy przestrzeni rozwiązań kolejnym ważnym obiektem jest basen przyciągania (ang. *basin of attraction*). Jest on definiowany jako taki zbiór rozwiązań, dla których algorytm zstępujący zwraca to samo optimum lokalne. Basen przyciągania jest pojęciem analogicznym do tego wykorzystywanego w terminologii układów dynamicznych oraz teorii chaosu, gdzie jako atraktor określa się stan do którego proces zmierza z wielu innych stanów. Zbiór takich stanów określa się jako obszar przyciągania lub właśnie basen przyciągania.

Ostatnim pojęciem związanych z FL opisanym w rozdziale jest bariera (ang. *barrier*, [262]). Termin został zapożyczony z fizyki, gdzie bariera powoduje konieczność dostarczenia określonej energia do układu aby możliwe stało się przejście z jednego stanu do drugiego. Bariera w ramach FLA to minimalna zmiana wartości funkcji celu konieczna do przejścia z jednego rozwiązania do drugiego 5.1. Choć definicja ta obejmuje dowolne dwa rozwiązania, to za „najciekawsze" przyjmuje się bariery pomiędzy optimami lokalnymi. Dla ustalonej definicji otoczenia, niech p będzie dowolną ścieżką z rozwiązania x do rozwiązania y. Wtedy, dla funkcji celu f, bariera między rozwiązaniami x i y jest opisana równaniem

$$B_f(x,y) \stackrel{\text{def}}{=} \min\{\max\{f(x) \mid x \in p\} \mid p \text{ to ścieżka z } x \text{ do } y\}. \tag{5.1}$$

Pomimo że termin bariery jest stosunkowo intuicyjny i prosty, to nie zawiera w sobie komponentu „odległości" pomiędzy rozwiązaniami. Z tego powodu, nawet stosunkowo niska wartość bariery może oznaczać wiele iteracji procesu przeszukiwania w ramach zadanego sąsiedztwa (długa ścieżka pomiędzy rozwiązaniami). Dodatkowo, termin bariery nie określa trudności wynikającej z kierunkowości (metody) przeszukiwania. Na przykład, ze względu na umiejscowienie rozwiązań w ramach basenów przyciągania możliwe jest, że przejście z jednego rozwiązania do drugiego jest znacznie prostsze w ramach wybranego procesu przeszukiwania niż „powrót" z rozwiania drugiego do pierwotnego.

5.2 Wybrane cechy przestrzeni rozwiązań

W ramach tego podrozdziału opisane zostaną bardziej szczegółowo wybrane, najpopularniejsze miary przestrzeni rozwiązań: szorstkość [289], epistaza [192], neutralność [221] oraz zdolność do ewolucji [285]. Oczywiście istnieje wiele innych miar jak i podejść do analizy krajobrazu przestrzeni

rozwiązań, część z nich zostanie przywołana w sposób zbiorczy aby choć poglądowo przedstawić zróżnicowanie w podejściu do ekspozycji różnorodnych cech przestrzeni rozwiązań. Nowe miary powstają zarówno: a) z przyczyn praktycznej użyteczności (jak np. pomoc w szacowaniu trudności instancji specyficznego problemu, bez perspektyw generalizacji); b) jako „naprawa" wad miar starszych, kiedy po dokładniejszych badaniach okazuje się, że wcześniejsze miary miały ograniczenia, które z łatwością mogą być rozwiązane poprzez nowe miary; c) jako skuteczne przeniesienie metodologii z innych dziedzin do FLA.

Zgodnie z [176], można przy tym wyróżnić pewną chronologię. W późnych latach 1980 i do połowy 1990 głównym obiektem badań była analiza liczby i rozkładu optimów lokalnych. Popularną miarą była więc szorstkość oraz – ze względu na próby wyjaśnienia działania algorytmów genetycznych – epistaza. W późnych latach 1990 uwagę zwrócono na neutralności. Po 2000 najpopularniejszą badaną metryką stała się zdolność do ewolucji.

Ciekawe spojrzenie na analizę przestrzeni rozwiązań (a więc i miary z nią związane) zapewnia praca [176]. Podkreślono tam, że FLA ma sens nawet wobec braku istnienia darmowych obiadów (ang. *No Free Lunch Theory*) – czyli twierdzeniem, że żaden z algorytmów optymalizacyjnych nie może być lepszy od innych algorytmów dla każdego możliwego przypadku. Nie wyklucza to potrzeby tworzenia i dostosowywania algorytmów do określonych przypadków problemów, szczególnie kiedy istnieje o nich pewna wiedza a priori, która może być wykorzystana.

5.2.1 Klasyczne miary teorii grafów

Przestrzeń rozwiązań może być reprezentowana przez graf $G = (S, E)$, gdzie S to zbiór wierzchołków reprezentujących rozwiązania, a

$$E = \{(s_1, s_2) \in S \times S \mid s_2 \in N(s_1)\} \tag{5.2}$$

to zbiór łuków odzwierciedlających wszystkie możliwe ruchy w ramach wybranego otoczenia. Z wierzchołkami można wiązać wagi, wynikające z definicji funkcji celu. Nie są one jednak istotne z punktu widzenia opisanych dalej klasycznych miar teorii grafów: średnicą oraz centrum.

Średnica grafu to odległość łącząca dwa najodleglejsze wierzchołki grafu. Innymi słowy, jeżeli średnica grafu wynosi a, to najkrótsza ścieżka łącząca dowolne dwa wierzchołki grafu ma długość co najwyżej a. Dla grafu G, jest ona wyrażona wzorem

$$\mathrm{dia}(G) = \max\{d(s_1, s_2) \mid s_1, s_2 \in S\}. \tag{5.3}$$

Jeżeli istnieją wierzchołki które nie łączy żadna ścieżka (graf nie jest spójny), często przyjmuje się, że średnica grafu jest nieskończona. W kontekście FLA, niewielkie wartości średnicy grafu można interpretować jako możliwość dotarcia przez procedurę przeszukiwania lokalnego do dowolnego rozwiązania, niezależnie od rozwiązania początkowego. Z kolei nieskończona średnica grafu oznacza, że dla niektórych rozwiązań początkowych osiągnięcie rozwiązania optymalnego dla danego otoczenia może nie być możliwe (brak własności *connectivity* otoczenia).

Centrum grafu to z kolei zbiór wierzchołków centralnych, czyli wierzchołków z których można dotrzeć do dowolnego wierzchołka możliwie najkrótszą drogą. Z oczywistych względów, miara jest definiowana dla grafów spójnych. Formalnie, centrum grafu można opisać wyrażeniem

$$\mathrm{cen}(G) = \arg\min_{s_1 \in S} \max_{s_2 \in S} d(s_1, s_2). \tag{5.4}$$

Intuicyjnie, rozwiązania z centrum grafu mogą stanowić dobre rozwiązania początkowe dla algorytmów przeszukiwania lokalnego.

5.2.2 Szorstkość

Szorstkość (ang. *ruggedness*, inaczej – nieregularność) jest jedną z podstawowych metryk opisujących przestrzeń rozwiązań. Miara ta opiera się o liczbę oraz rozkład lokalnych optimów i ma reprezentować zmienność krajobrazu; stanowić reprezentacje „pofałdowania" przestrzeni, z odpowiadającymi „wzniesieniami" oraz „dolinami". Dla przykładu, sąsiedztwo składające się z rozwiązań o znacznie różniących się od siebie wartościach funkcji celu sugeruje szorstką przestrzeń. Tymczasem zupełnym przeciwieństwem jest przestrzeń „płaska", o wielu rozwiązaniach z identyczną wartością funkcji celu. Innym przykładem przestrzeni o niewielkiej szorstkości jest przestrzeń z pojedynczym ekstremum, której rozwiązania tworzą pojedynczy basen atrakcji wokół globalnego optimum – tym samym znacznie ułatwiając przeszukiwanie z wykorzystaniem algorytmów lokalnej poprawy.

Pomimo prostoty idei stojącej za metryką, istnieje wiele sposobów jej praktycznego szacowania. Wynika to zarówno z pewnej dowolności formułowania założeń co do samej metryki, jak i ze zróżnicowanych sposobów kodowania rozwiązań badanych problemów oraz przyjętego sposobu mierzenia odległości pomiędzy rozwiązaniami. Prawdopodobnie pierwszym sposobem pomiaru szorstkości była metryka autokorelacji (ang. *autocorrelation*). Jest to średnia wartość korelacji wartości funkcji celu w ramach sąsiedztwa podczas błądzenia losowego (ang. *random walk*), przy zachowaniu stosunkowo małego kroku podczas poruszania się przez przestrzeń [300]. Ten sposób

rozumienia szorstkości może być wyrażony wzorem:

$$\rho(\epsilon) = \frac{E(f_t \cdot f_{t+\epsilon}) - E(f_i)E(f_{t+\epsilon})}{\mathrm{Var}(f_t)}, \tag{5.5}$$

gdzie f_t oznacza wartość funkcji celu rozwiązania odwiedzonego przez błądzenie losowe w kroku t, a ϵ to rozważany dystans (horyzont czasowy). W podobny sposób można zdefiniować dystans korelacji (ang. *correlation length*), jako średnią odległość pomiędzy rozwiązaniami, zanim te przestaną być ze sobą skorelowane. Analogicznie do autokorelacji wykorzystywany jest również współczynnik korelacji, z korekcją opartą o wariancję (ang. *variance-corrected correlation coefficient*) [178].

Innym przykładem podejścia do pomiaru szorstkości, może być rozważanie analizy korelacji w ramach błądzenia losowego jako serii czasowej w [116]. W tych badaniach przyjęto znacznie bardziej rygorystyczne założenia dla autokorelacji, gdzie uznawano korelacje za statystycznie istotną tylko w wypadku gdy przekracza podwojoną wartość odchylenia standardowego. Kolejnym podejściem do szacowania szorstkości są badania [289] gdzie przedstawiono nowe metryki, bazujące na entropii. Wykorzystano przy tym pomiar entropii w stosunku do rozkładu prawdopodobieństwa rozwiązań nietypowych w ramach sekwencji stworzonej przez błądzenie losowe po przestrzeni rozwiązań.

Należy też podkreślić za [208], że przytoczone sposoby szacowania szorstkości przeważnie wykorzystują błądzenie losowe. Prowadzi ono proces przez rozwiązania które są częściej nieskorelowane, przez co może być zbyt pesymistycznym opisem krajobrazu rozwiązań – szczególnie w kontraście do ukierunkowanego przeszukiwania (przez na przykład algorytm, przeszukiwania lokalnego), które prowadzi zazwyczaj przez „gładszą" część krajobrazu rozwiązań. Pomimo tego, pod warunkiem spełnienia pewnych wymagań (jak np. wystarczająco duża wielkość próbki), możliwe jest wykorzystanie przedstawionych sposobów szacowania szorstkości do szacowania poziomu trudności instancji dla problemów optymalizacji [208] – takich jak: QAP [184], wielowymiarowy problem plecakowy [273] czy też problemy marszrutyzacji [64]. W literaturze można jednak znaleźć również kontrprzykłady, jak choćby nieudana próba szacowania trudności dla gniazdowego problemu szeregowania zadań w pracy [182].

5.2.3 Epistaza

Epistaza (ang. *epistasis*) to termin pierwotnie opisujący zjawisko typowo biologiczne – interakcję pomiędzy genami, w wyniku której zmieniają się obserwowalne cechy organizmu (fenotyp) kodowane przez jeden gen, pod

wpływem obecności innego genu lub genów. Bardzo daleko idącym (i nie do końca poprawnym) uproszczeniem tego procesu jest dominacja jednego genu nad innymi (choć dominacja ma inne podłoże biologiczne) – czy jak czasami jest opisywana epistaza – maskowanie cech jednego genu przez inny lub inne. Przykład z pracy opisującej biologiczną epistazę [281]: *nieistotny jest gen koloru futra u myszy jeśli w innym genie – odpowiedzialnym za produkcję melaniny – dojdzie do mutacji, to osobnik zostanie albinosem, a efekty genu determinującego barwę sierści zostaną zamaskowane.* Co ważne z perspektywy optymalizacji, nawet w czysto biologicznym podejściu epistaza sprawia, że określenie cech organizmu na podstawie pojedynczych mutacji genów staje się niemożliwe przez nieliniowe współdziałanie genów.

Epistaza w ramach przestrzeni rozwiązań związana jest z „trudnością" rozwiązania problemu przez algorytmy genetyczne, dla których jakość przeszukiwania zależy od sposobu kodowania rozwiązania. Zgodnie z [192], algorytmy genetyczne działają w sposób „ślepy", z tendencją do optymalizacji funkcji celu poprzez zmiany w pojedynczym lub zaledwie kilku bitach kodowanego rozwiązania. Tym samym funkcje które „silnie" zależą od wielu bitów rozwiązania jednocześnie, okazują się trudne do rozwiązania z wykorzystaniem algorytmów genetycznych. Zachowanie to zostało zaobserwowane w ramach pracy [83] nad funkcjami „drogi królewskiej" (ang. *royal road functions*), gdzie próbowano określić jakie cechy problemów wpływają na proces przeszukiwania za pomocą algorytmów genetycznych.

Zgodnie z [192], dla funkcji $f : S = \{0,1\}^l \rightarrow \mathbb{N}^+$, epistaza opisuje jak bardzo funkcja $f(s)$ jest uzależniona od wartości na pojedynczym bicie s. Za [69], można też określić pewne spektrum – od minimalnej epistazy, gdzie każdy gen (bit) jest niezależny od każdego z pozostałych genów (czyli funkcję f można wyrazić jako liniową kombinację funkcji, z których każda zależy wyłącznie od pojedynczego genu), jak i maksymalną epistazę, gdzie nie istnieje żaden właściwy podzbiór genów niezależny od żadnego innego genu (co odpowiada funkcji f o wartościach ustalonych przypadkowo). Epistazę można wyznaczyć zgodnie z [69] za pomocą wariancji epistazy – liczonej w oparciu o kompozycję liniową ciągu reprezentującego rozwiązanie na podstawie bitów kodujących rozwiązanie. Uzyskana wartość liczbowa reprezentuje stopień niezależności pomiędzy genami. Alternatywnym sposobem jest epistaza bitowa, zaproponowana przez [82]. Wyznacza się ją za pomocą różnic w wariancji funkcji celu rozwiązań tworzących przez zmianę każdego pojedynczego bitu genu kodującego rozwiązanie. Obliczenie epistazy bitowej wymaga rozważenia wszystkich rozwiązań w ramach przestrzeni – lub próbki, jeśli ma być to wartość aproksymowana. Stad też istnieją liczne metryki, które stanowią przybliżoną wartość epistazy ale są łatwiejsze do

policzenia, jak np. rozstrzyganie bitowe (ang. *bit decidability*) [192].

5.2.4 Neutralność

Zgodnie z [176], neutralność (ang. *neutrality*) jest miarą przestrzeni rozwiązań przeciwną do miary szorstkości i reprezentuje regiony przestrzeni rozwiązań gdzie wartość funkcji celu jest identyczna lub prawie taka sama. Ta cecha przestrzeni może doprowadzić do „ukrycia" informacji o lokalnym optimum i przez to wpłynąć na proces przeszukiwana przestrzeni. Przykładem kiedy proces ten jest znacznie utrudniony, są metody oparte o sąsiedztwo w wypadku gdy całe sąsiedztwo ma tą samą wartość funkcji celu. Jednym z problemów optymalizacji dla którego neutralność szczególnie często wpływa na proces optymalizacji jest problem kolorowania grafu [179]. Pojęcie neutralności zostało początkowo zasugerowane w ramach pracy [142] nad zmianami cząstek w ramach badań nad ewolucją molekularną (procesem ewolucji dokonującym się na poziomie DNA, RNA i białek) – gdzie zaobserwowano, że większość zmian jest albo neutralna (ma bardzo mały wpływ na zdolność adaptacji) albo wręcz przeciwnie – jest śmiertelna.

Neutralność może być mierzona na wiele sposobów i zależy od przyjętych kryteriów. Jedną z metod jest sieć neutralna (ang. *neutral network*[1]), czyli graf gdzie węzły są pojedynczymi rozwiązaniami problemu z tą samą wartością funkcji celu, a krawędzie określają relacje sąsiedztwa pomiędzy takimi rozwiązaniami. W celu określenia neutralności danej przestrzeni rozwiązań, można policzyć ile neutralnych sieci posiada, ich rozmiar czy też średnice tak zdefiniowanych grafów. Są to jednak dla większych przestrzeni informacje trudne lub niemożliwe do wyznaczenia. Innym podejściem do pomiaru neutralności jest stopień neutralności (ang. *neutral degree*), czyli liczba rozwiązań o tej samej wartości funkcji celu w ramach sąsiedztwa – co stanowi dodatkową informacje i może być wykorzystane w ramach poszerzenia informacji o sieci neutralnej, np. poprzez obliczenie stopnia rozkładu wierzchołków w ramach sieci neutralnej. W pracy [221] autorzy zaproponowali neutralny spacer (ang. *neutral walk*), by mierzyć metrykę neutralności bazując na generycznym określeniu sąsiedztwa, z założeniem o jego dyskretnej reprezentacji. Technika ta opiera się na znalezieniu jednego neutralnego sąsiada, dla którego dystans od punktu początkowego zwiększa się z każdą iteracją – proces ten trwa aż do znalezienia otoczenia które nie jest neutralne, ale jednocześnie zwiększa łącznie pokonany dystans.

Neutralność nie stanowi problemu sama w sobie, wskaźnikiem o bardziej bezpośrednim wpływie na efektywność działania algorytmów lokalnej

[1] Wtrącenie ze względu na podobieństwo nazw: jest to sieć neutralna, a nie neuronowa – pojęcia te oznaczają zupełnie inne metody.

poprawy i genetycznych byłaby możliwość adaptacji („ewolucji") [122] rozwiązania – co można rozumieć jako możliwość znalezienia lepszego rozwiązania w sąsiedztwie. Miara taka mogłaby uzupełnić informacje o otoczeniu sieci neutralnej i odpowiedzieć na pytanie czy neutralność jest na tyle duża, że ma negatywny wpływ na proces przeszukiwania przestrzeni rozwiązań. Tego typu rozważania doprowadziły do opracowania nowej metryki, jaką jest zdolność do ewolucji.

5.2.5 Zdolność do ewolucji

Zdolność do ewolucji (ang. *evolvability*) ma z założenia pokazywać na ile dany proces przeszukiwania jest zdolny do „przejścia" do regionów o lepszej wartości funkcji celu. Metryka ta jest silnie powiązana z samym sposobem przeszukiwania, nie opisuje więc wyłącznie cechy przestrzeni rozwiązań danego problemu ale zawiera również komponent oceny „jakości" procesu przeszukiwania rozważanego algorytmu. Tak jak w wypadku pozostałych metryk, istnieje wiele podejść do sposobu mierzenia zdolności do ewolucji – choć w wypadku tej metryki, oprócz różnic wynikających z przyjętych założeń, pojawiają się różnice wynikające bezpośrednio z wykorzystanego algorytmu przeszukiwania.

W kontekście biologicznym, zgodnie z [208], zdolność do ewolucji można określić jako *zdolność populacji do tworzenia wariantów osobników lepiej przystosowanych niż dotychczas istniejące*. Choć nie jest jasne dlaczego biologiczna zdolność do ewolucji jest tak ważną cechą mechanizmu ewolucji [180], to istnieją badania [285] wskazujące na jej pożądany wpływ w kontekście algorytmów ewolucyjnych, szczególnie w dłuższej perspektywie (kiedy liczba iteracji jest wystarczająco duża). W kontekście algorytmów ewolucyjnych – zdolność do ewolucji jest zapewniana poprzez odpowiednie zróżnicowanie osobników w ramach populacji. Należy jednak podkreślić, za [208], że zdolność do ewolucji nie ogranicza się do metod opartych o populację. Można mówić o niej również w wypadku mechanizmów innych heurystyk, wszędzie tam gdzie zwiększana jest (szczególnie długoterminowo) *szansa na poprawę proponowanego rozwiązania*. Do takich mechanizmów można zaliczyć mechanizm restartów w algorytmach przeszukiwania z zabronieniami, a dla symulowanego wyżarzania – mechanizm akceptacji rozwiązania o gorszej wartości funkcji celu z pewnym prawdopodobieństwem.

Można wymienić wiele metod bezpośredniego mierzenia zdolności do ewolucji. Przykładem jednej z nich jest miara portretów przystosowania ewolucyjnego (ang. *Fitness Evolvability Portraits*) [254]. Służy ona do zobrazowania relacji pomiędzy zdolnością do ewolucji, a szeregiem innych miar; w tym również szorstkością i neutralnością. Metryka ta opiera się

o średnie wartości innych miar w ramach grup rozwiązań o tej samej wartości funkcji celu. Inną metodą jest chmura przystosowania (ang. *Fitness Cloud*) [59]. Jest to sposób przedstawienia na wykresie punktowym zależności pomiędzy funkcją celu rozwiązań „rodzica", a funkcją celu stworzonego na jego podstawie „potomka"; którą określa mianem graniczącego przystosowaniem – zakłada się przy tym relację 1 do 1 w ramach przyjętego algorytmu bądź operatora. Tak uzyskane dane obrazuje się w postaci wykresu, gdzie wartości na jednej z osi są wartościami funkcji celu rodzica, a na drugiej osi oznacza się graniczące przystosowanie. Istnieją też modyfikacje tego podejścia, pozwalające uwzględnić dodatkowo trudność ucieczki z lokalnego optimum (metoda ang. *Accumulated escape probability*) [175].

5.2.6 Inne, wybrane metryki przestrzeni rozwiązań

Jak już wspomniano wcześniej, istnieje bardzo wiele miar przestrzeni krajobrazu rozwiązań. Opisane wcześniej miary, choć należące do najbardziej popularnych, nie stanowią pełnego przekroju narzędzi FLA. W tym podrozdziale w skrócie opisane zostaną rzadziej rozważane, ale interesujące miary.

Przykładem przeniesienia metodologii z innej dziedziny jest analiza spektralna krajobrazu przestrzeni rozwiązań (ang. *Spectral Landscape Analysis*). W pracy [117] wykorzystano transformację analogiczną do transformacji Fouriera w celu rozłożenia przestrzeni rozwiązań na bazowe składowe, dzięki transformacji funkcji celu. Zabieg ten ułatwia dalszą analizę, niestety ze względu na koszt obliczeniowy jest często niepraktyczny [261]. Z kolei analiza informacji przestrzeni rozwiązań (ang. *Information Analysis*) [289] opiera się o narzędzia teorii informacji. Zasada działania metody polega na określaniu liczby bitów koniecznych do opisu procesu błądzenia losowego przez wybrany krajobraz przestrzeni rozwiązań. Samą przestrzeń traktuje się jako funkcję rozkładu elementów w ramach systemu – im więcej bitów informacji jest potrzebne do opisu, tym bardziej złożony krajobraz.

Wiele miar wykorzystuje informacje o rozwiązaniach spełniających konkretne warunki odnośnie wartości funkcji celu – w szczególności są to optima lokalne. Metryki te mogą być stosunkowo proste, jak w wypadku modalności (ang. *Modality*) przestrzeni rozwiązań, gdzie mierzona jest częstość występowania lokalnych optimów w odniesieniu do rozmiaru samej przestrzeni [208]. Przykładem bardziej złożonej miary jest rozkład przystosowania (ang. *Fitness Distribution*), gdzie mierzona jest częstość występowania każdej z wartości funkcji celu. Pewną odmianą tej techniki jest gęstość stanów (ang. *Density of States*) [227], bazująca na liczbie rozwiązań z określoną wartością funkcji celu w ramach pewnej próbki przestrzeni rozwiązań. Metoda wykorzystuje strategię Boltzmana by szacować optimum globalne

oraz szansę znalezienia go poprzez błądzenie losowe.

Miary zwodzenia (ang. *Deception*) z założenia mają odzwierciedlać trudność przeszukiwania przestrzeni rozwiązań z wykorzystaniem zadanego algorytmu przeszukiwania. Tym samym mają dostarczać informację czy dana przestrzeń rozwiązań nie zawiera cech bezpośrednio wpływających negatywnie na dany algorytm. Można wyróżnić tutaj badania bezpośrednio związane z algorytmami genetycznymi (ang. *GA-deception* w [74], ale również inne metryki w [302]) oraz korelację między wartością funkcji celu, a odległością (ang. *Fitness Distance Correlation* [130]). Druga z wymienionych miar, pomimo iż wymaga znajomości globalnego optimum (co jest jej znaczną wadą), jest wykorzystywana wraz z miarą dyspersji (ang. *Dispersion*) [177] do opisu globalnych relacji pomiędzy funkcją celu, a odległością miedzy rozwiązaniami. Tym samym pozwala opisać topologię przestrzeni rozwiązań z perspektywy globalnej.

5.3 Sieć lokalnych optimów

Sieć lokalnych optimów (ang. *Local Optima Network*, LON) to model FLA, który przedstawia informacje o globalnej strukturze przestrzeni rozwiązań w postaci kompaktowego grafu skierowanego $G_{\mathrm{LON}} = (N_{\mathrm{LON}}, E_{\mathrm{LON}})$. Wierzchołkami tego grafu, których zbiór oznaczono przez N_{LON}, są optimami lokalnymi. Krawędzie grafu, ze zbioru E_{LON}, wraz z wagami reprezentują sposób przejścia pomiędzy tymi optimami. Sam model został wykorzystany w wielu pracach badawczych przez zespół Ochoa, Verel, Doalio i Romassini [53, 65, 67, 200, 202, 291]. Analizowali oni między innymi przestrzenie rozwiązań klasycznych problemów optymalizacji, takich jak problem komiwojażera [201] czy problem przepływowy [68, 108]. Oprócz tego zespół ten rozważał również problemy często wykorzystywane przy ocenie metod FLA – QAP [66, 279] oraz problem NK [203, 292]. LON to model ważny w kontekście dysertacji, ze względu na jego wykorzystanie w badaniach własnych. Przedstawiony opis będzie opierał się w znacznej mierze o [202], gdzie metoda została przedstawiona.

5.3.1 Wierzchołki LON

Węzły grafu LON są optimami lokalnymi. Formalnie, rozwiązanie $s \in S$ jest lokalnym optimum (w wypadku problemów minimalizacji) wtedy i tylko wtedy, gdy $\forall a \in N(s)\ (f(a) \geqslant f(s))$. Stąd,

$$N_{\mathrm{LON}} = \{s \in S \mid \forall a \in N(s)\ (f(a) \geqslant f(s))\}. \tag{5.6}$$

Konstrukcja LON jest uzależniona od przyjętego rodzaju sąsiedztwa, a więc pośrednio jest związana z określonym algorytmem przeszukiwania przestrzeni rozwiązań. Dla przykładu, w pracy [202] wykorzystano metodę największego wzrostu (ang. *Hill Climbing*, odpowiednik metody zstępowania dla problemów maksymalizacji). Dla analizy istotne znaczenie miał sposób wyboru sąsiada: można było albo wybrać najlepsze rozwiązanie z otoczenia (sposób wybrany w przytoczonej pracy); albo pierwsze poprawiające rozwiązanie z otoczenia. Każdy z wyborów prowadzi do utworzenia innego grafu LON. W celu uniknięcia niejednoznaczności, przez $h(s)$ oznaczono lokalne optimum dla rozwiązania s.

Graf LON może zostać skonstruowany zarówno w oparciu o informacje o wszystkich rozwiązaniach w ramach zadanej instancji, jak i może też być wyznaczony na podstawie zadanego „wycinka" przestrzeni rozwiązań. Wycinek taki powstaje w wyniku próbkowania części rozwiązań, co jest szczególnie użyteczne w wypadku dużych instancji problemu. Wtedy zwykle analiza całej przestrzeni rozwiązań jest niemożliwe w akceptowalnym czasie.

5.3.2 Krawędzie LON

Istnieją co najmniej dwa sposoby określania połączeń pomiędzy optimami lokalnymi: model krawędzi oparty o przejścia pomiędzy basenami przyciągania (ang. *basin-transition*) oraz model oparty o opuszczanie optimum (ang. *escape edges*). Oba sposoby zostaną przedstawione, wykorzystując bezpośrednio opis z pracy [202].

Krawędzie oparte o basen przyciągania

Dla zdefiniowania krawędzi opartych o basen przyciągania (ang. *basin-transition*), niezbędne jest wprowadzenie pojęcia basenu przyciągania. Zgodnie z [202], dla lokalnego optimum $s \in N_{\text{LON}}$ jest on zbiorem rozwiązań $b(s) = \{s' \in S \mid h(s') = s\}$. Rozmiar basenu przyciągania to liczność zbioru $|b(s)|$. Każde rozwiązanie należy do któregoś z basenu przyciągania $S = \cup_{s \in N_{\text{LON}}} b(s)$ oraz baseny są rozłączne $\forall s, s' \in N_{\text{LON}}$ $(s \neq s' \Rightarrow b(s) \cap b(s') = \emptyset)$. O ile przestrzeń rozwiązań nie cechuje się silną neutralnością, składa się z wielu basenów przyciągania.

Dla każdej z par rozwiązań s oraz s', $P(s \rightarrow s')$ to prawdopodobieństwo przejścia z rozwiązania s do rozwiązania s' w ramach wybranego sąsiedztwa. W pracy [202] przedstawiono przykład dwóch sąsiedztw – otrzymanego z wykorzystaniem:

- zmiany bitowej pojedynczego znaku (ang. *single bit-flip*) dla rozwiązania kodowanego w postaci łańcucha binarnego (dla rozwiązania o roz-

miarze n, rozmiar sąsiedztwa to n, a rozmiar przestrzeni rozwiązań 2^n; przykład problemu bazowego: problem NK),
- zamiany ze sobą kolejności wykonywania dwóch operacji dla rozwiązania kodowane w postaci permutacji (dla rozwiązania o rozmiarze n, rozmiar sąsiedztwa to $n(n-1)/2$, a rozmiar przestrzeni rozwiązań $n!$; przykład problemu bazowego: problem QAP).

Rozważmy reprezentację rozwiązania o rozmiarze n dla błądzenia losowego, gdzie przejście do każdego rozwiązania z sąsiedztwa jest tak samo prawdopodobne. Dla rozwiązania kodowanego jako łańcuch binarny, prawdopodobieństwo przejścia z rozwiązania s do s' można obliczyć ze zbioru

$$P(s \to s') = \begin{cases} \frac{1}{n} & \text{jeżeli } s' \in V(s), \\ 0 & \text{jeżeli } s' \notin V(s), \end{cases} \tag{5.7}$$

a dla rozwiązania w postaci permutacji

$$P(s \to s') = \begin{cases} \frac{1}{N(N-1)/2} & \text{jeżeli } s' \in V(s), \\ 0 & \text{jeżeli } s' \notin V(s). \end{cases} \tag{5.8}$$

Prawdopodobieństwo $P(s \to b(s'))$ przejścia z rozwiązania $s \in S$ do rozwiązania należącego do basenu przyciągania $b(s')$, wyrażone jest wzorem

$$P(s \to b_j) = \sum_{s' \in b_j} P(s \to s'). \tag{5.9}$$

Tym samym, łączne prawdopodobieństwo przejścia z basenu przyciągania $b(s')$ do basenu przyciągania $b(s'')$, czyli waga krawędzi od wierzchołka s' do s'', jest wyrażone wzorem

$$P(b(s') \to b(s'')) = \frac{1}{|b(s')|} \sum_{s \in b(s')} P(s \to b(s'')). \tag{5.10}$$

Krawędzie oparte o opuszczenie optimum

Za [202], krawędzie oparte o opuszczenie optimum (ang. *escape edges*) zdefiniowane są w oparciu o wybraną funkcję odległości d. Przykładem takiej funkcji może być minimalna liczba perturbacji permutacji reprezentującej rozwiązanie, za pomocą wybranego operatora (np. typu wstaw), konieczna do przejścia z optimum s' do s''. Kontynuując przykład: krawędź pomiędzy optimum s' oraz s'' istnieje, jeżeli jest takie rozwiązanie s, że $d(s, s') \leqslant D$ oraz $h(s) = s''$. Waga takiej krawędzi wynosi

$$w_{s' \to s''} = |\{s \in S \mid d(s, s') \leqslant D \wedge h(s) = s''\}|. \tag{5.11}$$

Waga ta może zostać znormalizowana przez podzielenie przez liczbę rozwiązań oddalonych o dystans D od s',

$$w_{s' \to s''} = \frac{|\{s \in S \mid d(s, s') \leqslant D \wedge h(s) = s''\}|}{|\{s \in S \mid d(s, s') \leqslant D\}|}. \tag{5.12}$$

Ponieważ metoda tworzenia krawędzi opiera się o dystans pomiędzy optimami lokalnymi, w ramach własnych badań zdecydowano się odejść od rozwiązania zaproponowanego wyżej, pochodzącego z [202], na rzecz własnego rozwiązania. Zamiast tego przyjęto, że krawędź pomiędzy optimami lokalnymi s' oraz s'' istnieje tylko i wyłącznie wtedy gdy można przejść do rozwiązania s'' poprzez wykonanie perturbacji „kopnięcia" (podwójnego zastosowania jednego ze standardowych ruchów – opisanych w 3.2.2, w tym przedstawionych wybranych na rysunku 3.1) na rozwiązaniu s', a następnie zastosowaniu zachłannego algorytmu poprawy. Waga łuku (s', s'') jest liczbą reprezentującą prawdopodobieństwo przejścia z s' do s'' i jest estymowana w ramach procesu próbkowania przestrzeni rozwiązań.

5.3.3 Miary LON

Skonstruowanie sieci lokalnych optimów (LON) umożliwia wyznaczenie dla niej wartości licznych miar, opisujących wybrane jej cechy. Znaczna część z wykorzystanych metryk, ze względu na strukturę sieci LON, to po prostu typowe miary służące do opisu grafów. Zdecydowano się jednak na przedstawienie wybranych miar LON, szczególnie tych powszechnie używanych jak i tych które zostały wykorzystanie w badaniach własnych opisanych w kolejnych rozdziałach.

edgeToNode – stosunek liczby łuków do wierzchołków, wzór: $edgeToNode = \frac{|E_{LON}|}{|N_{LON}|}$;

escRate – średnia liczba perturbacji typu „kopnięcie" potrzeba do opuszczenia minimum lokalnego;

numSubSinks – liczba wierzchołków określanych jako *subsinks*. Wierzchołek uznawany jest za *subsink* wtedy i tylko wtedy, gdy nie ma żadnych wychodzących łuków do wierzchołków o niższej wartości funkcji celu, tj. $\forall (s', s'') \in E_{LON}\ (f(s'') \geqslant f(s'))$;

average shortest path to optimum – średni najkrótszy dystans między dowolnym wierzchołkiem, a najbliższym wierzchołkiem reprezentującym optimum globalne $s^\star$. Wierzchołki nie połączone z $s^\star$ są pomijane. Odległość między wierzchołkami, w wypadku grafów ważonych, jest odwrotnością wagi łuku;

conRel – stosunek liczby połączonych z optimum globalnym $s^\star$, do liczby wierzchołków nie połączonych z $s^\star$;

clustering – miara opisująca stopień w jakim wierzchołki grafu tworzą podobne do siebie zgrupowana (klastry) – w badaniach własnych wykorzystano *global clustering coefficient* (inaczej *transitivity*) na podstawie pracy [196], miara ta opisuje prawdopodobieństwo istnienia krawędzi (a, c) jeśli istnieją krawędzie (a, b) and (b, c), tak aby węzły a, b, c utworzyły trójkąt. Obliczana za pomocą wzoru:

$$C = 3 \times \frac{\text{liczba trójkątów}}{\text{liczba trójek}}, \tag{5.13}$$

gdzie za „trójkę" przyjmuje się węzeł połączony krawędziami z nieuporządkowaną parą innych węzłów;

minimum as percent – procentowy stosunek liczby węzłów reprezentujących rozwiązania będące optimum globalnym do wszystkich węzłów (nazwa miary ze względu na wykorzystanie w problemie minimalizacji) – ze wzoru:

$$O = \frac{|N_{\text{LON}}|}{|N|} \cdot 100\%; \tag{5.14}$$

assortativity – miara opisująca na ile połączone są ze sobą wierzchołki o podobnych parametrach – miara rozumiana zgodnie z badaniami przedstawionymi w [195] (występujące tłumaczenie: asortatywność). Przyjmując rozwiązanie w postaci permutacji $s \in N$ jako wierzchołek LON, można przedstawić następujące parametry x wierzchołka:

- **assortativity-in** – liczba krawędzi kończących się w wierzchołku

$$x_{\text{in}}(s) = |\{(s', s'') \in E_{\text{LON}} : s'' = s\}|; \tag{5.15}$$

- **assortativity-out** – liczba krawędzi zaczynających się w wierzchołku

$$x_{\text{out}}(s) = |\{(s', s'') \in E_{\text{LON}} : s' = s\}|; \tag{5.16}$$

- **assortativity-total** – suma krawędzi zaczynających i kończących się w wierzchołku

$$x_{\text{total}}(s) = x_{\text{in}}(s) + x_{\text{out}}(s); \tag{5.17}$$

- **assortativity-bin** – rozmiar basenu przyciągania reprezentowaną przez wierzchołek

$$x_{\text{bin}}(s) = |b(s)|; \tag{5.18}$$

- **assortativity-of** – wartość funkcji celu rozwiązania reprezentowanego przez wierzchołek

$$x_{\text{of}}(s) = f(s). \tag{5.19}$$

Formalnie miara *assortativity* definiowana jest jako:

$$A = \frac{\sum_i e_{i,i} - \sum_i \left(\sum_j e_{i,j} \sum_j e_{j,i}\right)}{1 - \sum_i \left(\sum_j e_{i,j} \sum_j e_{j,i}\right)}, \tag{5.20}$$

gdzie $e_{i,j}$ jest stosunkiem krawędzi łączących wierzchołki typu i oraz j – to jest takich krawędzi $(s', s'') \in E_{\text{LON}}$, że $x(s') = i$ oraz $x(s'') = j$ – do liczby wszystkich krawędzi.

Wyliczenie miar LON stanowiących typowe miary grafów możliwe jest w sposób stosunkowo prosty, dzięki istnieniu wyspecjalizowanych bibliotek do analizy statystycznej grafów. Przykładem takiego narzędzia jest biblioteka: `graph-tool`[2] [206].

5.4 Wnioski i uwagi

Zgodnie z zaplanowanym zakresem i przyjętymi ograniczeniami, rozdział stanowi zwięzłe i skoncentrowane wprowadzenie do tematyki analizy krajobrazu przestrzeni rozwiązań. Podano w nim część z najważniejszych pojęć oraz wymieniono wiele metryk powszechnie wykorzystywanych w ramach przytoczonej tematyki – rozszerzono przy tym opis metryk takich jak: szorstkość, epistaza, neutralność czy zdolność do ewolucji, bazując na ich popularności w cytowanych badaniach. Ze względu na wykorzystanie w badaniach własnych przedstawiono model grafowy sieci lokalnych optimów. Opis ten uzupełniono o sposoby konstrukcji oraz metryki służące do pomiaru ilościowego tego modelu, znaczna część z których to miary również wykorzystywane przy obrazowaniu cech grafu. Choć informacje zawarte w rozdziale stanowią jedynie niewielki, specyficzny wycinek informacji z dziedziny FLA, to uzupełniono je o liczne odniesienia do wartościowych prac, które w razie potrzeby z pewnością mogą posłużyć do dalszego pogłębienia wiedzy o analizie krajobrazu przestrzeni rozwiązań.

[2]Biblioteka dostępna pod adresem https://graph-tool.skewed.de/ na dzień 29.06.2022.

Część II

Opis przeprowadzonych badań

Rozdział 6

Sieci lokalnych optimów jako metoda analizy krajobrazu przestrzeni rozwiązań

Problemy NP-trudne są jednymi z najczęściej analizowanych problemów w ramach badań operacyjnych. Do ich rozwiązania w znacznej mierze wykorzystywane są algorytmy heurystyczne. Wynika to z braku możliwości otrzymania rozwiązania optymalnego poprzez zastosowania algorytmów dokładnych dla większych instancji problemów (póki nie wykazane zostanie czy P $\neq$ NP), które często stanowią odzwierciedlenie praktycznych wyzwań planistyczno-produkcyjnych. Jak przedstawiono w rozdziale 2, istnieje wiele algorytmów heurystycznych z powodzeniem wykorzystywanych do rozwiązywania problemów optymalizacji. Jako przykłady mogą posłużyć udane badania wykorzystujące: przeszukiwanie z zabronieniami [26, 29, 30] oraz symulowane wyżarzanie [38]. Jednak nie zawsze istnieje zgoda co do ustalenia który z licznych algorytmów jest najlepszy dla wybranego problemu. Jako przykład może posłużyć przegląd prac [50] dla elastycznego problemu przepływowego, gdzie skategoryzowano 14 rodzajów algorytmów optymalizacji wykorzystywanych do rozwiązywania problemu – między innymi: algorytmy ewolucyjne, hybrydowe, przeszukiwanie z zabronieniami, optymalizacja za pomocą roju cząstek czy też przeszukiwanie oparte o sąsiedztwo. Dodatkowo dla każdej z kategorii można wyróżnić wiele wariantów bazowych algorytmów (dobrym przykładem przeglądu wielu odmian algorytmów genetycznych dla klasycznego problemu komiwojażera jest praca [286]).

Wspomniany wyżej wybór najlepszego w danym kontekście algorytmu nazywany jest w literaturze problemem wyboru algorytmu (ang. *Algorithm Selection Problem*, ASP [225]). Aby taki wybór był dobry, konieczne jest

zebranie tak wielu informacji o rozwiązywanym zadaniu (czyli problemie bazowym) jak to możliwe. Uzyskanie danych opisujących pojedyncze instancje problemu w sposób dokładny jest możliwie, między innymi dzięki analizie przestrzeni rozwiązań (FLA, pojęcie opisano w rozdziale 5). Tak zebrane dane można wykorzystać do rozwiązania ASP, wykorzystując techniki uczenia maszynowego lub nawet proste metody bazujące na korelacji. Wyzwaniem, podjętym w niniejszym rozdziale, pozostaje jednak zebranie wartościowych informacji, przydatnych w kontekście rozwiązywanego problemu. W ramach przedstawionych badań wybrano jedną z metod FLA, czyli sieć lokalnych ekstremów (ang. *Local Optima Network*, LON) – nowatorską metodę opisu powiązań pomiędzy rozwiązaniami w ramach instancji problemów optymalizacji. LON to grafowy model, opisujący zbiór lokalnych ekstremów (węzłów) wraz z prawdopodobieństwem przejścia pomiędzy nimi (w postaci krawędzi) w trakcie procesu przeszukiwania przestrzeni rozwiązań (szczegółowy opis modelu w sekcji 5.3). Badania przeprowadzono dla problemu TSP, opisanego w podrozdziale 2.2.

W kontekście przedstawionej motywacji, celem tego rozdziału jest odpowiedź na następujące pytania:

1. czy metoda oparta o LON może dostarczyć informacji pozwalających skutecznie rozwiązywać ASP dla wybranego portfolio algorytmów, na przykładzie TSP?
2. czy metoda rozwiązywania ASP oparta o LON może dostarczyć rezultatów konkurencyjnych do używanych w praktyce rozwiązań, na przykładzie biblioteki `Google Optimization Tools`[1]?
3. które z miar LON warto stosować w konteście omawianego problemu?

Pierwsze z pytań bezpośrednio odnosi się do jednej z tez doktoratu: *korzystając ze sztucznych sieci neuronowych oraz analizy krajobrazu przestrzeni rozwiązań możliwe jest rozwiązywanie problemu wyboru algorytmu.* Pozostałe z nich pozwalają skonkretyzować wiedzę, ułatwiając zastosowanie badanych metod w praktyce. Rozdział zawiera wyniki badań własnych, zaprezentowanych na konferencji o zasięgu międzynarodowym, a następnie opublikowanych w ramach materiałów pokonferencyjnych [28].

6.1 Powiązane badania

Wyczerpujący przegląd literatury w tematyce ASP został przedstawiony w pracy [153]. Przykładem badań ASP w kontekście jednego problemu –

[1]Biblioteka dostępna pod adresem `https://developers.google.com/optimization` na dzień 16.04.2022.

konkretnie kwadratowego problemu przydziału (ang. *Quadratic Assignment Problem*, QAP) – jest publikacja [209]. Wykorzystano tam dwa rodzaje metaheurystyk: odporne przeszukiwanie z zabronieniami (ang. *Robust Tabu Search*) oraz metodę zmiennych otoczeń (ang. *Variable Neighborhood Search*). W celu wyboru algorytmu najlepiej nadającego się do rozwiązywania każdej z instancji problemu, wykorzystano wiele znanych i popularnych metod klasyfikacji, między innymi: mechanizm oparty o regułę pojedynczej zmiennej (ang. *One Variable Rule Learner*, OneR), sekwencyjną optymalizację minimalną (ang. *Sequential Minimal Optimization*, SMO), maszynę wektorów nośnych (ang. *Support Vector Machine*, SVM), proces gaussowski (ang. *Gaussian Processes*, GProc), czy regresję liniową (ang. *Linear Regression*, LR). Rezultaty okazały się bardzo obiecujące. Podejście pozwoliło na wskazanie najlepszego algorytmu (w sensie uzyskanej wartości funkcji celu) w przypadku blisko 80% instancji. Niestety czas potrzebny do zebrania koniecznych informacji o instancjach był znacznie większy niż łączny czas potrzebny do rozwiązania bazowego problemu z wykorzystaniem każdej z rywalizujących metod. Jest to oczywista wada uniemożliwiająca praktyczne zastosowanie tego typu FLA do rozwiązywania ASP (bardziej efektywne jest bezpośrednie porównanie wyników działania algorytmów). Inne badania nad ASP dla QAP zostały przedstawione w pracy [256], gdzie wykorzystano proces uczenia z wykorzystaniem prostej, jednokierunkowej sieci neuronowej w połączeniu z sieciami samoorganizującymi się (sieć Kohonena) [255]. Sieci neuronowe zostały również z powodzeniem wykorzystane [256] do przewidywania algorytmu wygrywającego spośród portfolio 3 algorytmów heurystycznych, wykorzystując jako dane 9 miar FLA. Średnia wartość dokładności wyboru wygrywającego algorytmu w ramach pięciokrotnie powtórzonej procedury walidacji krzyżowej na zbiorze testowym osiągnęła wartość 94%. Mimo to autorzy zalecają jednak ostrożność z wyciąganiem pochopnych wniosków ze względu na przeprowadzenie badań na zaledwie 28 instancjach problemu QAP. W pracy [127] przetestowano wyniki sześciu algorytmów metaheurystycznych dla problemu przewidywania struktury białka – za pomocą korelacji udało się zaobserwować związek pomiędzy kilkoma metrykami FLA oraz jakością wyników algorytmów.

Choć przytoczone rezultaty mogą przemawiać na korzyść wykorzystania FLA jako narzędzia w rozwiązywaniu ASP, istnieje ku temu szereg przeszkód. Po pierwsze, konieczne jest zebranie i przetworzenie ogromnych ilości danych, często w skali przekraczającej możliwości dostępnego sprzętu. Po drugie, brak jest ujednoliconego sposobu przetwarzania danych – co znacząco utrudnia możliwość wymiany i ponownego wykorzystania zebranych próbek FL w społeczności naukowej. Na część z tych wyzwań odpowiadają

sieci lokalnych optimów (LON, [202], patrz podrozdział 5.3), przy jednoczesnym zapewnieniu „świeżego spojrzenia" na sposób opisu oraz ekstrakcji cech instancji problemu. LON to sieć składająca się z węzłów reprezentujących ekstrema lokalne połączonych za pomocą krawędzi odpowiadających prawdopodobieństwu przejścia pomiędzy tymi ekstremami w procesie przeszukiwania przestrzeni rozwiązań. Tym samym model LON zapewnia usystematyzowany sposób kondensacji danych na temat przestrzeni rozwiązań który został z powodzeniem wykorzystany do zebrania danych dla problemów optymalizacji, takich jak: problem komiwojażera [200, 201], QAP [123, 202, 279] oraz NK [202, 282]. Pomimo przytoczonych przykładów wykorzystania metod FLA, analiza LON nie stała się jeszcze powszechnie wykorzystywaną metodą uzyskiwania cech instancji dla ASP.

Na podstawie przeglądu literatury, zasadnym zdaje się zbadanie możliwości wykorzystania analizy LON w kontekście ASP dla jednego z klasycznych problemów, jakim jest TSP. Algorytmy spośród których następował wybór w ramach ASP ograniczono do implementacji z popularnej biblioteki `Google Optimization Tools`, `ORTools`. Badania przeprowadzono dla instancji wygenerowanych w sposób losowy oraz dla wybranych instancji ze zbioru danych `TSPLib` (oczekiwano zróżnicowanych cech instancji w związku z takim wyborem).

6.2 Konstrukcja sieci lokalnych optimów

Z powodu bardzo dużych przestrzeni rozwiązań, węzły (rozwiązania) i krawędzie (prawdopodobieństwa przejścia pomiędzy rozwiązaniami) LONów są wyznaczane poprzez próbkowanie, podobnie jak w pracy [123]. Metoda ta została jednak zmodyfikowana aby możliwe było zastosowanie jej przy mniejszym budżecie mocy obliczeniowej. Zamiast poszukiwać „elitarnych" ekstremów lokalnych (czyli tych, których nie można poprawić za pomocą dwóch ruchów typu zamień), postanowiono ograniczyć próbkowanie do rozwiązań, których nie można poprawić za pomocą pojedynczego operatora typu zamień. Dodatkowo, w ramach przedstawionych badań, liczba otrzymanych węzłów została przyjęta jako parametr, podczas gdy w metodologii przyjętej w pracy [123] parametrem tym była liczba prób generacji węzłów.

Algorytm 3 opisuje sposób próbkowania węzłów LON. Pierwszym krokiem jest wygenerowanie losowego rozwiązania $s \in S$. Następnie rozwiązanie to jest optymalizowane z wykorzystaniem zachłannego algorytmu spadku – ponieważ badania przeprowadzono dla problemu komiwojażera, wykorzystano w tym celu klasyczną heurystykę 2-opt [63]. Jeżeli rozwiązanie nie może zostać dalej poprawione jedynie z wykorzystaniem metody

Algorytm 3: Metoda tworzenia węzłów LON, za [123]

Wejście : I_{nmax} – docelowa liczba węzłów;
I_{natt} – liczba prób generacji węzłów;
instancja TSP
Wyjście: N_{LON} – zbiór węzłów LON

```
1 N_LON ← ∅;
2 for i = 1, 2, ..., I_nmax do
3     for i = 1, 2, ..., I_natt do
4         s ← generujLosoweRozwiązanie();
5         s ← 2-opt(s);
6         if s jest lokalnym optimum then
7             if s ∉ N_LON then
8                 N_LON ← N_LON ∪ {s};
9                 break
```

2-opt i jest unikalne, zostaje uznane za węzeł. W wypadku przeciwnym generowane jest kolejne rozwiązanie losowe i proces zostaje powtórzony. Parametry I_{nmax} oraz I_{natt} odpowiadają oczekiwanej liczbie węzłów w ramach budowanej sieci LON oraz liczbie prób generowania każdego z węzłów.

Algorytm 4 podsumowuje proces próbkowania krawędzi w sieci LON. Metoda ta została opisana w pracy [123], jednak w ramach przedstawionych badań została zmodyfikowana. Zdecydowano się mierzyć dodatkowo liczbę ruchów typu „kopnięcie" które prowadzą do takich ekstremów, które nie są reprezentowane w ramach rozważanej, niekompletnej sieci LON. Pomiar tego parametru pozwala ocenić na ile dobrze próbkowano przestrzeń rozwiązań za pomocą algorytmu 3. Dla każdego z węzłów $s \in N_{\text{LON}}$ w LON stosuje się ruch typu „kopnięcie" (ang. *kick-move*). Ruch ten stanowi perturbację (intensyfikującymi eksplorację) i jest definiowany jako $k = 2$ losowych operatorów zamiany wykonywanych po sobie. Tak otrzymana trasa s' jest następnie optymalizowana za pomocą wariantu metody zstępowania. Jest to zmodyfikowana wersja metody 2-opt, dla której w każdej iteracji wybierane jest pierwsze rozwiązanie które poprawia rozwiązanie początkowe. Odróżnia to ją od tradycyjnego podejścia, gdzie wybierane jest rozwiązanie najlepsze z otoczenia. Wykorzystano tę strategię bazując na pracy [123], gdzie pozwoliła ona otrzymać lepsze wyniki. Jeśli tak znalezione rozwiązanie znajduje się wśród węzłów N_{LON}, tworzona jest krawędź (s, s') która dodawana jest do zbioru krawędzi LON. W przeciwnym wypadku (otrzymane rozwiązanie nie jest węzłem LON) ruch rozpatrywany jest jako wio-

Algorytm 4: Metoda wyznaczania krawędzi LON, za [123]

Wejście: N_{LON} – zbiór węzłów LON;
I_{eatt} – liczba ruchów *kick* dla każdego węzła;
instancja TSP

Wyjście: E_{LON} – zbiór łuków LON;
wagi łuków LON

```
1  E_LON ← ∅;
2  zainicjuj wagę każdego możliwego łuku LON wartością 0;
3  foreach s ∈ N_LON do
4      for i = 1, 2, ..., I_eatt do
5          s' ← zastosujLosowyKick(s);
6          s' ← 2-optPierwszejPoprawy(s');
7          if s' ∈ N_LON then
8              E_LON ← E_LON ∪ {(s, s')};
9              zwiększ wagę łuku (s, s') o 1;
10         else
11             zwiększ liczbę ruchów kick z s prowadzących do
                rozwiązań spoza N_LON o 1;
```

dący do nieznanego ekstremum lokalnego (gdyby badania były prowadzone z większym budżetem mocy obliczeniowej, można byłoby dodać to ekstremum do węzłów N_{LON}). Proces ten jest powtarzany I_{eatt} razy dla każdego z węzłów. Waga krawędzi (s, s') jest równa liczbie dodań (s, s') krawędzi do E_{LON}. Tym samym im większa jest waga krawędzi (s, s'), tym bardziej prawdopodobne jest przejście pomiędzy rozwiązaniem (węzłem) s oraz s'.

6.3 Eksperymenty obliczeniowe

W ramach przeprowadzonych badań zdecydowano się wykorzystać kanoniczny zbiór danych: *TSPLib*[2] dla problemu TSP. Jednak ze względu na ograniczoną moc obliczeniową ograniczono się do 19 instancji o relatywnie małym rozmiarze: `eil51`, `berlin52`, `st70`, `pr76`, `eil76`, `rat99`, `rd100`, `kroA100`, `kroB100`, `kroC100`, `kroD100`, `kroE100`, `eil101`, `lin105`, `pr107`, `pr124`, `ch130`, `pr136`, `pr144`. Tym samym instancje posiadały od 51 do 176 miast. Aby nieco zdywersyfikować zbiór danych wygenerowano po 30

[2]Zbiór danych dostępny pod adresem http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ na dzień 16.04.2022.

Tabela 6.1: Średnia względna jakość rozwiązań dla różnych algorytmów.

Instancje	Średnia Δ [%]					
	Auto	GD	GLS	SA	TS	OTS
rnd30	2,525	2,525	0,000	2,513	0,152	0,206
rnd30, 50, 100	2,304	2,304	0,093	2,408	0,532	0,381
TSPLib	3,527	3,527	0,172	3,458	2,822	1,063

losowych instancji problemów dla rozmiarów: 30, 50 oraz 100 równomiernie rozłożonych miast. Uzasadnieniem tej decyzji jest fakt, że zbiory danych składają się przeważnie z instancji trudnych do rozwiązania, które niekoniecznie muszą być reprezentatywne dla problemu. Potencjalnie ustalenie cech odróżniających instancje trudne od łatwych, bez konieczności przeszukiwania przestrzeni rozwiązań, byłoby istotnym rezultatem w kontekście wykorzystania praktycznego metody FLA.

Aby porównać różne procesy przeszukiwania przestrzeni rozwiązań zdecydowano się na wykorzystanie powszechnie dostępnej biblioteki `Google Optimization Tools` (OR-Tools)[3], zawierającej implementacje wybranych algorytmów metaheurystycznych. Decyzja ta pozwoliła na porównanie różnorodnego zbioru algorytmów bez obawy o poprawność ich implementacji, a dodatkowo biblioteka OR-Tools posiada tryb automatycznego doboru odpowiedniego algorytmu do danego problemu czy zadania. Tym samym początkową ideą było aby ten mechanizm wyboru uznać za punkt odniesienia dla metody wyboru zaproponowanej w ramach przeprowadzonych badań. Dla każdej z instancji problemu wykorzystano następujące opcje: automatyczny dobór algorytmu przeszukiwania (tryb *Auto*), algorytm zstępujący (*ang. Greedy Descent*, GD), nadzorowane przeszukiwanie lokalne (*ang. Guided Local Search*, GLS), symulowane wyżarzanie (*ang. Simulated Annealing*, SA), przeszukiwanie z zabronieniami (*ang. Tabu Search*, TS) oraz zmodyfikowana wersja przeszukiwania z zabronieniami (*ang. Objective Tabu Search*, OTS). Zgodnie z dokumentacją biblioteki, zalecanym w rozwiązywaniu problemów z rodziny marszrutyzacji algorytmem jest GLS.

W ramach porównywania procesu przeszukiwania przestrzeni przez algorytmy, ograniczono czas działania każdego z algorytmów do 1 sekundy dla każdej instancji (ze względu na mały rozmiar badanych instancji). Relatywna jakość rozwiązania była liczona za pomocą wzoru:

[3]Biblioteka dostępna pod adresem https://developers.google.com/optimization na dzień 16.04.2022.

Tabela 6.2: Porównanie parami efektywności algorytmów.

	Wygrane	Pary algorytmów, A vs. B														
		1v2	1v3	1v4	1v5	1v6	2v3	2v4	2v5	2v6	3v4	3v5	3v6	4v5	4v6	5v6
TSPLib	alg. A	0	0	**4**	4	3	0	**4**	4	3	**11**	**10**	**8**	2	1	1
	remis	19	5	**7**	6	5	5	**7**	6	5	**2**	**2**	**2**	15	10	11
	alg. B	0	14	**8**	9	11	14	**8**	9	11	**6**	**7**	**9**	2	8	7
md	alg. A	0	0	**34**	19	15	0	**34**	19	15	69	48	**36**	1	1	8
	remis	90	25	**28**	27	15	25	**28**	27	15	14	32	**34**	59	30	43
	alg. B	0	65	**28**	44	60	65	**28**	44	60	7	10	**20**	30	59	39

Kolumna 1v2 pokazuje porównanie najlepszych rezultatów algorytmów 1 i 2, 1v3 algorytmów 1 i 3, etc.... Algorytmy: 1-Auto, 2-GD, 3-GLS, 4-SA, 5-TS, 6-OTS. Rezultaty sugerujące komplementarność pogrubiono.

$$\Delta(s) = \frac{f(s) - f(s^*)}{f(s^*)} \cdot 100\%, \quad \text{zakładając, że } \forall s \in S \ (f(s) > 0), \tag{6.1}$$

gdzie s^* jest najlepszym znanym rozwiązaniem dla instancji, a f to funkcja celu. Zagregowane wyniki rozwiązań uzyskanych przez algorytmy przedstawiono w tabeli 6.1.

Wyniki uzyskane za pomocą badanych algorytmów wykluczyły możliwość porównania zaproponowanej metody wyboru algorytmów z tą zaimplementowaną w ramach OR-Tools. Automatyczny wybór algorytmu z biblioteki prowadził do uzyskania takich samych wyników jak z wykorzystaniem algorytmu GD, który niestety okazał się najgorszym z badanych algorytmów (dla warunków przyjętych w badaniu). Brak różnicy między wynikami trybu Auto a wynikami GD można zaobserwować w tabeli 6.1. Dodatkowo algorytm GLS okazał się znacznie lepszy od reszty badanych algorytmów dla większości instancji, przez co problem wyboru algorytmu spośród dostępnych w ramach biblioteki OR-Tools dla problemu komiwojażera był trywialny.

Fakt ten jednak nie eliminuje możliwości wykorzystania analizy opartej o LON w zadaniu wyboru odpowiedniego algorytmu w ramach mniejszego portfolio algorytmów. Aby rozwiązanie ASP dostarczało użytecznych informacji, portfolio algorytmów powinno składać się z komplementarnych metod rozwiązania (t.j. powinny być skuteczne dla różnych klas instancji, uzupełniając wzajemnie swoje braki). W celu ustalenia które

algorytmy są komplementarne, dla dowolnej pary algorytmów A i B obliczono: liczbę instancji w której algorytm A ma przewagę nad algorytmem B – co jest zapisane jako $w(A, B)$, liczbę remisów, czyli instancji dla których otrzymano taki sam wynik zapisywana jest jako $d(A, B)$. Definiujemy algorytmy A i B jako komplementarne jeśli $w(B, A) + w(A, B) \geqslant 1,5 \cdot \max\{w(B, A), w(A, B)\}$ oraz $d(A, B) \leqslant w(A, B) + w(B, A)$. Wyniki takiego porównania algorytmów można znaleźć w tabeli 6.2. Komplementarne okazały się algorytmy: (GA i SA), (GLS i OTS), (tryb auto i SA) oraz w wypadku instancji z biblioteki TSPLIB (ale już nie instancji losowych): (GLS i SA) oraz (GLS i TS).

6.3.1 Sposób budowy sieci lokalnych optimów

Sposób generowania sieci lokalnych ekstremów został przedstawiony w podrozdziale 6.2 oraz w algorytmach 3 oraz 4. Jako parametry LON dla losowych, stworzonych na potrzeby badania instancji, dla $n = 30$ i $n = 50$ miast przyjęto wymóg $I_{\text{nmax}} = 1\,000$ węzłów oraz $I_{\text{eatt}} = 10\,000$ prób generacji krawędzi. Wartości zostały dobrane w ten sposób aby zbiory lokalnych optimów najmniejszych, losowych instancji były dokładnie spróbkowane. W wypadku większych instancji oraz instancji z biblioteki TSPLib ustalono z kolei $I_{\text{nmax}} = 10\,000$ oraz $I_{\text{eatt}} = 10\,000$.

W literaturze znaleźć można wiele definicji miar opisujących różne aspekty LON. Część z nich jest wzajemnie skorelowana, co prowadzi do nadmiarowości i potencjalnej nadreprezentacji niektórych własności sieci. Aby zminimalizować ten efekt, przeprowadzono analizę z wykorzystaniem współczynnika korelacji Spearmana. Obliczono go dla każdej z par rozważanych miar LON oraz dla każdej z testowanych instancji. Wykorzystano miary: *edgeToNode*, *escRate*, *numSubSinks*, *average shortest path to optimum*, *conRel*, *assortativity*, *clustering* – ich dokładny opis można znaleźć w sekcji 5.3.3. Macierz uzyskanych korelacji pokazano na rysunku 6.1.

Wartości dla miar z rodziny *assortativity* korelują ze sobą w sposób istotny statystycznie, stąd do dalszych badań wytypowano jedynie jedną z nich. Dodatkowo wyniki wskazują na znaczną korelację pomiędzy rozmiarem instancji n i większością z badanach miar LON. Wynika to z mniejszej zmienności wartości miar pomiędzy instancjami tego samego rozmiaru. Stan ten może być spowodowany nie tyle inherentnymi cechami LON instancji, a raczej wybranym sposobem próbkowania. Na przykład: średnia wartość miary *edgeToNode* dla instancji wygenerowanej w sposób losowy, składającej się z $n = 30$ miast oraz sieci składającej się z 1 000 węzłów wynosi aż 282. Tymczasem w wypadku losowej instancji o $n = 50$ miastach, miara ta przyjmuje wartość 30. Dla instancji o rozmiarze $n = 100$

oraz 1 000 węzłów, praktycznie nie istnieją krawędzie nie będące pętlami. Jednak już zwiększenie liczby węzłów do 10 000 przekłada się na wartość $edgeToNode = 2,6$. Tymczasem intuicja sugeruje, że wraz ze zwiększaniem się liczby miast n, wartość miary *edgeToNode* powinna rosnąć (zwiększa się liczba możliwych przejść pomiędzy ekstremami lokalnymi). Podobny problem został zaobserwowany między innymi w pracy [201], gdzie pokazano wpływ wartości miar w zależności od stopnia spróbowania przestrzeni. Niestety, w momencie przeprowadzania badań nie istniała jedna, powszechnie akceptowana metoda próbkowania przestrzeni, która pozwalałaby wyeliminować ten niekorzystny efekt.

6.3.2 Problem doboru algorytmu dla TSP

W wyniku przeprowadzonych badań, jako zbiór algorytmów dla ASP wykorzystano głównie te pary algorytmów z biblioteki OR-Tools, które okazały się być komplementarne (tabela 6.2). Dla każdej z par algorytmów A-B, problemem było przewidzenie (bez uruchamiania algorytmów) czy:

- algorytm A uzyskuje lepsze wyniki niż algorytm B,
- algorytm B uzyskuje wyniki lepsze niż algorytm A,
- zarówno algorytm A jak i algorytm B pozwalają na uzyskanie zbliżonego wyniku,

jest to więc problem klasyfikacji dla 3 klas. Klasyfikacja odbywa się na podstawie miar LON opisanych w ramach podrozdziału 5.3.3. Do klasyfikacji wybrano kilka metod zaimplementowanych w ramach narzędzia WEKA[4]. Narzędzie to pozwoliło na łatwe przebadanie efektywności kilku popularnych klasyfikatorów, zaczynając od bardzo prostych metod takich jak: naiwny klasyfikator Bayesa lub klasyfikator bazujący na pojedynczej cesze (ang. *Decision Stump*), aż do metod bardziej zaawansowanych jak: wielowarstwowy perceptron czy losowy las decyzyjny. Dodatkowo jako punkt odniesienia wykorzystano klasyfikator zeroR – najprostszą metodę, polegającą na przewidywaniu klasy najbardziej licznej. Wybór ten pozwolił również na wyraźne zobrazowanie na ile niezbalansowany jest zbiór danych. Dla każdego z klasyfikatorów mierzono procent instancji dla których przewidywanie było poprawne (wybrano poprawnie algorytm wygrywający lub wskazano na faktyczny remis).

Wyniki eksperymentu przedstawiono w tabeli 6.3. Dla instancji ze zbioru *TSPLib* oraz pary algorytmów GLS i TS, różnica procentowa poprawnych

[4]Narzędzie dostępne pod adresem https://www.cs.waikato.ac.nz/ml/weka/ na dzień 13.05.2022.

Tabela 6.3: Porównanie efektywności klasyfikatorów

.

	Klasyfikator	Procent poprawnych klasyfikacji dla par algorytmów									
		2v3	2v4	2v5	2v6	3v4	3v5	3v6	4v5	4v6	5v6
TSPLib	najlepszy	73,7	**57,9**	**52,6**	57,9	**68,4**	52,6	**57,9**	79,0	**73,7**	**68,4**
	zeroR	73,7	42,1	47,4	57,9	57,9	52,6	42,1	79,0	52,6	57,9
rnd	najlepszy	61,7	48,3	40,0	**60,0**	73,3	68,3	51,7	86,7	51,7	61,7
	zeroR	63,3	46,7	40,0	55,0	73,3	68,3	51,7	86,7	60,0	58,3

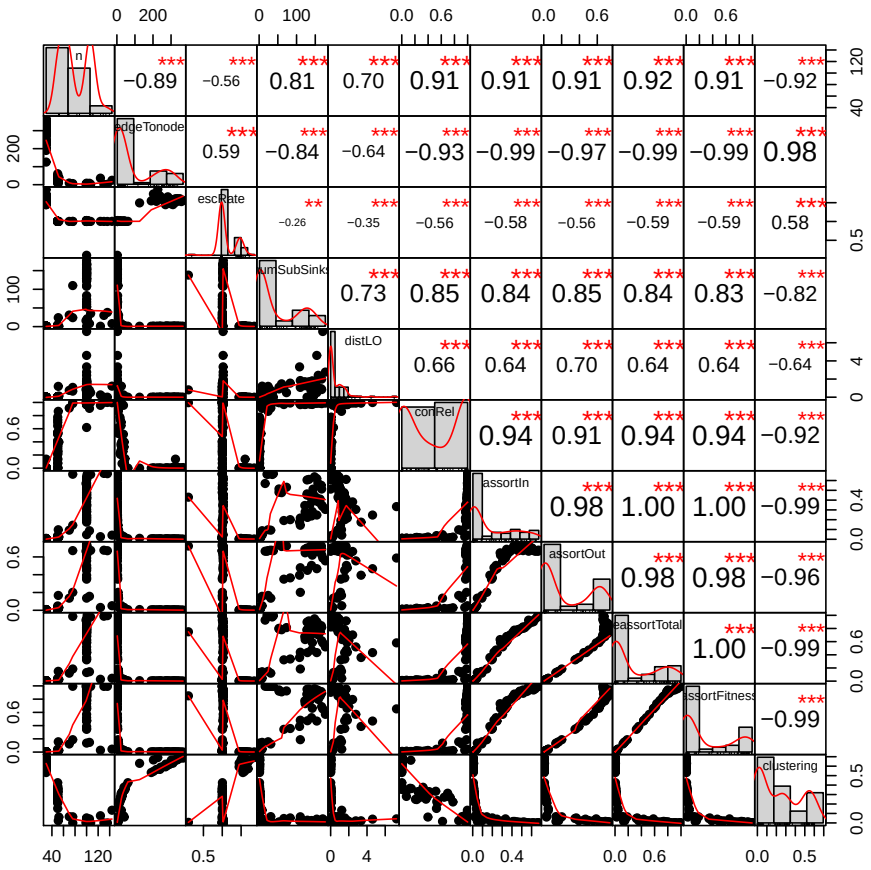
W kolumnach pokazano rezultaty najskuteczniejszego klasyfikatora oraz klasyfikatora odniesienia zeroR dla par algorytmów. Rezultaty wyznaczono dla instancji losowych i z TSPLib. Algorytmy: 2-GD, 3-GLS, 4-SA, 5-TS, 6-OTS. Rezultaty wykazujące poprawę co najmniej 5% w stosunku do zeroR pogrubiono. Rezultaty par algorytmów komplementarnych podkreślono.

predykcji pomiędzy „klasyfikatorem" odniesienia zeroR, a innymi klasyfikatorami była mniejsza niż 5%. Jednak dla par GD oraz SA (oraz trybu Auto i SA), najlepszy z klasyfikatorów uzyskał 57,9% poprawnych predykcji, podczas gdy zeroR jedynie 42,1%. Dla pary GLS-OTS bazowe zeroR zostało pokonane o 15,8% – 57,9% do 42,1%. W podobny sposób zeroR zostało pokonane dla par: SA-OST oraz TS-OST, gdzie najlepszy klasyfikator osiągnął wyniki odpowiednio 73,7% oraz 68,4%, podczas gdy wynik zeroR wynosił 52,6% oraz 57,9%. Niestety ostatnie z tych par (SA-OST oraz TS-OST) zostały wcześniej określone jako niekomplementarne (w kontekście ustalonego w tym podrozdziale kryterium). Analiza wyników pozwoliła zauważyć, że klasyfikatory były w stanie jedynie wskazać dla których instancji nastąpi remis lub poprawnie wskazywano jako zwycięzcę ten algorytm, który dawał lepsze rezultaty dla większej liczby instancji. Tym samym największym problemem było niedoszacowanie wygranych algorytmu „słabszego", który miał przewagę jedynie dla mniejszej liczby instancji. Dodatkowo instancje wygenerowane w sposób losowy okazały się znacznie trudniejsze w klasyfikacji – jedynie dla jednej z par algorytmów (i to niestety nie komplementarnej) najlepszy z klasyfikatorów był w stanie poprawić o co najmniej 5% predykcję zeroR. Rezultat ten prawdopodobnie wynika ze znacznie mniejszych różnic pomiędzy losowymi instancjami wygenerowanymi w ten sam sposób, a których cechy nie musiały zostać dostatecznie odzwierciedlone w procesie próbkowania wykorzystanym w ramach opisanych badań.

6.4 Wnioski i uwagi

Na podstawie przedstawionych badań zaprezentowano metodę wykorzystania analizy przestrzeni rozwiązań do rozwiązywania problemu wyboru algorytmu dla problemu komiwojażera. Najbardziej znaczące były badania dotyczące praktycznej próby wykorzystania analizy sieci lokalnych optimów (LON) jako sposobu na ekstrakcję cech (ang. *feature extraction*) instancji problemu. Wyniki badań wskazują, że sposób ten może być z powodzeniem wykorzystany pod warunkiem wystarczająco zróżnicowanych instancji (co odpowiada różnicy wartości miar ich LON) oraz komplementarnym portfolio algorytmów (t.j. powinny być skuteczne dla różnych klas instancji, uzupełniając wzajemnie swoje braki). Tym samym uzyskano odpowiedź pozytywną na pierwsze z pytań postawionych we wstępie do rozdziału oraz potwierdzono jedną z tez doktoratu: *korzystając ze sztucznych sieci neuronowych oraz analizy krajobrazu przestrzeni rozwiązań możliwe jest rozwiązywanie problemu wyboru algorytmu.* Główną wadą przedstawionej metody jest nakład obliczeniowy potrzebny do próbkowania przestrzeni rozwiązań, w ramach budowy grafu LON, który jest znacznie większy niż podczas bezpośredniej próby rozwiązania instancji problemu. Zaskoczeniem była nieoczekiwanie słaba jakość automatycznego wyboru algorytmu zaimplementowana w ramach biblioteki *OR-Library*, która nie mogła zostać wykorzystana jako punkt odniesienia dla mechanizmu przedstawionego w ramach badań. Macierz korelacji pozwoliła z kolei na określenie zbioru miar LON pozbawionego nadmiarowości.

Obiecującym obszarem dla dalszych badań okazało się ustalenie związku pomiędzy metodą próbkowania przestrzeni LON, a wartościami uzyskanych miar. W szczególności, istotne byłoby znalezienie miar których wartości nie byłyby uzależnione od stopnia kompletności próbkowania przestrzeni rozwiązań, a jednocześnie dobrze „oddawałyby" cechy tej przestrzeni. Opracowanie takiego zestawu miar byłoby sukcesem świadczącym o możliwości zastosowania praktycznego metody. W ramach przytoczonej literatury zagadnienie to wydaje się być częściowo przemilczane. Tymczasem, bardzo duży nakład obliczeniowy konieczny do dokładnego odwzorowania cech przestrzeni w miarach LON (w tym stworzenia grafu LON) utrudnia skuteczne wykorzystanie analizy LON w rzeczywistych zastosowaniach. Temat ten został podjęty w kolejnym rozdziale 7. Dodatkowo interesujące wydaje się przetestowanie zaprezentowanej metodologi dla innych problemów optymalizacji dyskretnej, w połączeniu z alternatywnymi metodami ich rozwiązywania, również na różniących się od siebie danych syntetycznych.



Rysunek 6.1: Macierz korelacji dla wszystkich rozważanych miar LON. Pod przekątną pokazano wykres punktowy wartości. Na przekątnej pokazano histogramy. Ponad przekątną pokazano wartości współczynników korelacji Spearmana. Rysunek opracowany w programie Weka, gdzie kropka jest wykorzystywana do rozdzielenia części dziesiętnej liczby. Wartości p oznaczono przez gwiazdki: *** dla $p < 0{,}001$, ** dla $p < 0{,}01$ oraz * dla $p < 0{,}1$.

Rozdział 7

Konstrukcja sieci lokalnych optimów w praktyce

Na przestrzeni ostatnich lat analiza przestrzeni rozwiązań (FLA) umożliwiła opracowanie wielu nowych, zróżnicowanych metod analizy i opisu instancji problemów optymalizacji. Pozwoliło to na pełniejsze zrozumienie problemów, które są rozwiązywane w ramach badań operacyjnych. Wiele z najbardziej obiecujących metod FLA bazuje na sieciach lokalnych optimów (LON), czyli skompresowanej reprezentacji przestrzeni rozwiązań w postaci grafu, opisującego instancję z perspektywy algorytmu optymalizującego. Aby stworzyć reprezentację LON, przestrzeń rozwiązań musi zostać poddana próbkowaniu. Niestety, trudno jest znaleźć informacje na temat właściwego sposobu próbkowania – w tym kluczowej informacji o minimalnym nakładzie obliczeniowym, wymaganym, aby „wystarczająco" dobrze zbadać przestrzeń rozwiązań. W ramach przedstawionych badań, zbadano wpływ liczby próbek na jakość uzyskanych miar opisujących LON dla cyklicznego problemu przydziału w niepermutacyjnym problemie przepływowym szeregowania zadań. Proces próbkowania został podzielony na etapy, w których przestrzeń była stopniowo próbkowana w coraz pełniejszym zakresie, aż do przeprowadzenia analizy obejmującej całą przestrzeń rozwiązań. Dla każdego z tych etapów wyznaczano wartości miar LON.

Główną przeszkodą w szerokim zastosowaniu analizy LON w praktyce, postulowaną również w rozdziale 6, jest relatywnie duży koszt obliczeniowy. Co za tym idzie, z perspektywy szerszych rozważań opisanych w ramach dysertacji, bardzo ważne jest znalezienie odpowiedzi na pytanie: czy możliwe jest ograniczenie kosztów obliczeniowych LON poprzez mniejszy stopień próbkowania przestrzeni rozwiązań? Gdyby miary LON okazały się „odporne" na zróżnicowany stopień próbkowania przestrzeni rozwiązań, ozna-

czałoby to, że możliwe jest ich zastosowanie jako pomocniczych metod opisu instancji bezpośrednio podczas przeszukiwania przestrzeni rozwiązań. Nawet jeśli wyznaczenie wartości miar LON byłoby związane ze znacznymi kosztami, skompresowana informacja o instancji mogłaby stanowić świetny sygnał wejściowy dla sieci neuronowych lub innych metod sztucznej inteligencji. Zaś te metody mógłby być wykorzystane dla przykładu do „nadzorowania" procesu przeszukiwania przestrzeni rozwiązań. Z drugiej strony znaczne zmiany w wartościach miar LON w zależności od stopnia próbkowania przestrzeni rozwiązań oznaczałyby niską opłacalność wykorzystania mocy obliczeniowej i sugerowałby użycie znacznie prostszych metod FLA w ramach bardziej skomplikowanych systemów.

W odróżnieniu od istniejących publikacji, badania nad tym zjawiskiem przedstawione w niniejszym rozdziale zostały przeprowadzone na realistycznym, rozbudowanym problemie optymalizacji – cyklicznym problemie przepływowym z gniazdami produkcyjnymi z pojedynczymi operatorami. W ramach kontekstu całej dysertacji, pozwalają one na odpowiedzenie (lub chociaż sformułowanie pewnych przypuszczeń) na pytania takie jak:

1. jaka jest zmienność miar LON w zależności od stopnia próbkowania przestrzeni rozwiązań?
2. czy pomimo zaobserwowania zmienności miar, uzasadnione jest stosowanie metody opartej o LON, czy też wystarczające jest wykorzystanie prostszych miar FLA?
3. czy zasadne jest stosowanie „drogich" obliczeniowo LON w ramach większych systemów?

Tym samym głównym celem rozdziału jest wykazanie jednej z tez pracy: *sposób przeprowadzenia próbkowania przestrzeni rozwiązań może mieć istotny wpływ na wartości estymacji miar tej przestrzeni.* Rozdział zawiera wyniki badań zaprezentowanych na konferencji o zasięgu międzynarodowym, opublikowanych w materiałach pokonferencyjnych [92].

7.1 Perspektywa analizy krajobrazu przestrzeni rozwiązań

Krajobraz przestrzeni rozwiązań (FL) jest ustrukturyzowaną reprezentacją procesu eksploracji zbioru rozwiązań przez dany algorytm przeszukiwania (lokalnego) dla konkretnej instancji problemu. Dzięki analizie FL, możliwe jest „uchwycenie" unikalnych własności instancji problemu, które następnie można wykorzystać, np. w celu rozwiązania problemu doboru algorytmu

(ang. *Algorithm Selection Problem*, ASP [153] – tak jak w badaniach własnych przedstawionych w poprzednim rozdziale 6) lub w celu oszacowania na ile trudno będzie uzyskać wystarczająco dobre rozwiązanie z wykorzystaniem określonego algorytmu.

Jedną z metod analizy FL jest zdefiniowanie, a następnie obliczanie metryk ją opisujących, a tym samym opisujących instancję problemu z perspektywy danej klasy algorytmów. Istnieje wiele dobrze znanych i powszechnie wykorzystywanych metryk, zarówno tych ogólnych jak i dobranych bezpośrednio do wybranych problemów – dla przykładu: rozkład przystosowania (ang. *fitness distribution*[227]), epistaza (ang. *epistasis*[192]), szorstkość (ang. *ruggedness*[289]), neutralność (ang. *neutrality*[221]). Wyczerpujący przegląd metryk wykorzystywanych w ramach FL można znaleźć w pracach: [122, 208, 176] – skrócony opis wybranych miar był przedstawiony w ramach sekcji 5.2.

Pomimo znacznego potencjału analizy FL, jest to proces dość trudny do przeprowadzenia, między innymi ze względu na ogromne zapotrzebowanie na dane oraz moc obliczeniową. W odpowiedzi na te wyzwania, Ochoa [202] zaproponowała sieć lokalnych optimów (ang. *Local Optima Networks*, LON). LON zapewnia kompaktową reprezentacje FL, poprzez zachowanie jedynie informacji o lokalnych optimach, wraz z informacją o ich powiązaniach. Model ten został z powodzeniem zastosowany do wielu, zróżnicowanych problemów optymalizacji, takich jak: QAP [123, 202, 279], TSP [200, 201] oraz NK [202, 282].

Bezpośrednie stworzenie kompletnego grafu LON dla dużych instancji nie jest możliwe w praktyce ze względu na zbyt długi czas wymagany do analizy całej przestrzeni. Tym samym konieczne jest częściowe próbkowanie przestrzeni rozwiązań. Wiąże się to zwykle z kompromisem pomiędzy kosztem obliczeniowym, a jakością opisu przestrzeni rozwiązań – tym samym powstaje ważne pytanie: *Jaki jest minimalny nakład obliczeniowy potrzebny aby zbudować LON, który reprezentuje instancje bazowego problemu w sposób wystarczający?*. Choć istnieją badania na ten temat [201, 202, 247], konieczne było przeprowadzenie badań własnych w kontekście oceny potencjału wykorzystania LON jako elementu bardziej rozbudowanego systemu.

7.2 Problem bazowy

Problem którego przestrzeń rozwiązań miała zostać poddana analizie został wybrany zgodnie z trzema kryteriami: problem powinien być złożony, istotny oraz relatywnie słabo zbadany w kontekście FLA. Potrzeba ciągłego zwiększania zdolności produkcyjnej wraz z rosnącym zapotrzebowaniem na

coraz bardziej zróżnicowane produkty wymusza stosowanie nowych rozwiązań, w tym systemów wytarzania z równoległymi maszynami [32, 50, 228]. Co więcej, rozwój technologii komputerowej pozwolił na znacznie bardziej efektywne zarządzanie elastycznymi systemami produkcji, w których zadania mogą zostać wykonane na wiele alternatywnych sposobów. Jednak ze względu na nieprzewidywalność wymagań oraz względnie zróżnicowany profil produkcji często konieczne jest wykorzystanie maszyn o zróżnicowanych parametrach, co prowadzi do powstawania alternatywnych marszrut technologicznych. Optymalizacja tak złożonych procesów wymaga zastosowania równie wyrafinowanych modeli oraz dedykowanych algorytmów.

Koncepcja produkcji opartej o gniazda łączy w sobie dwie różne metody organizacji systemu wytwarzania. Jedna skupiona jest na jak największej wydajności w produkcji jednorodnych wyrobów (podejście ang. *product layouts* – charakterystyczne dla linii produkcyjnej). Druga z metod stawia nacisk na elastyczność procesu wytwarzania, w tym wykorzystanie różnych funkcjonalności każdego z wykorzystanych zasobów (podejście ang. *functional layouts* – charakterystyczne dla problemów szeregowania zadań, w tym klasycznego problemu przepływowego czy też gniazdowego). Gniazda są zazwyczaj skupiskami maszyn, charakteryzujących się zbliżonymi możliwościami produkcyjnymi. Tym samym możliwe jest, z ich wykorzystaniem, wytwarzanie produktów o podobnych wymaganiach technologicznych. Dla zdefiniowanych w taki sposób gniazd produkcyjnych, do transportu elementów podawanych obróbce pomiędzy maszynami, bardzo często wykorzystuje się roboty autonomiczne – więcej na temat produkcji gniazdowej (w tym z wykorzystaniem robotów transportujących) można znaleźć w pracy [72].

Cykliczny problem przepływowy z gniazdami robotycznymi jest jednym z najczęściej badanych wśród problemów optymalizacji gniazda robotycznego. Gniazdo takie składa się z m maszyn i 1, 2 lub r robotów wykorzystywanych do transportu przetwarzanych elementów (zdań) pomiędzy maszynami, gdzie wykonywane są właściwe operacje. Produkcja jest powtarzana w sposób cykliczny – w modelowym przypadku nieskończoną liczbę powtórzeń. Rozważa się przy tym podział wariantów problemu (już od wczesnych prac poświęconych tej tematyce) na dwie grupy [245] pod względem zróżnicowania wytwarzanych produktów: tam gdzie występuje jedynie jeden typ, oraz tam gdzie występuje wiele różnych rodzajów produktów (zadań). Wariant jednozadaniowy jest wariantem częściej badanym, przegląd problemów z tej kategorii można znaleźć w pracach [62, 71, 102]. W zależności od występowania dodatkowych ograniczeń, część z tych problemów można rozwiązać w czasie wielomianowym. Optymalizacja wariantu gniazda robo-

tycznego dla wielu zróżnicowanych zdań jest często trudniejszym wyzwaniem. Przegląd literatury na ten temat można znaleźć w [99], gdzie badano wariant dla pojedynczego gniazda robotycznego oraz pojedynczego robota transportowego. Zaproponowano rozwiązania oparte o metodę dokładną (programowanie całkowitoliczbowe) oraz podejście przybliżone (równoległe przeszukiwanie z zabronieniami oraz algorytm genetyczny). Dodatkowo badano wpływ definicji cykliczności na efektywność produkcji.

Zgodnie z przytoczonymi badaniami, wariant problemu rozważany w ramach badań można określić jako system produkcji z jednym operatorem (maszyną) w każdym z gniazd produkcyjnych. System ten jest hybrydą pomiędzy cyklicznym problemem przydziału [27] oraz niepermutacyjnym problemem przepływowym szeregowania zadań. Przykład algorytmu optymalizacji dla tego problemu został wcześniej opisany w ramach pracy [26], gdzie wykorzystano podejście dwuetapowe. Przydział maszyn do operacji wyznaczano z wykorzystaniem zarówno metod przybliżonych jak i algorytmów dokładnych, podczas gdy harmonogram optymalizowano z wykorzystaniem przeszukiwania z zabronieniami.

7.3 Eksperymenty obliczeniowe

Jak już wspomniano, analiza całej przestrzeni rozwiązań nie jest możliwa dla dużych instancji problemów. Tym samym koniecznie należy skorzystać z wybranej metody próbkowania przestrzeni tak aby stworzyć reprezentatywny „wycinek" przestrzeni rozwiązań. Określenie na ile częściowe próbkowanie przestrzeni wpływa na jakość uzyskanych informacji jest kluczowe dla zasadności wykorzystanie FLA do analizy instancji problemów optymalizacji. W celu zbadania tego zagadnienia zdecydowano o wygenerowaniu całej przestrzeni rozwiązań dla małych instancji problemów. Następnie porównano uzyskane wartości referencyjne miary do wartości miar uzyskanych w ramach częściowego próbkowania przestrzeni tych instancji.

Eksperymenty były przeprowadzone na losowych instancjach, których rozmiar wahał się od 6 do 9 operacji. Dla każdego z tych rozmiarów wygenerowano 30 instancji. Przestrzeń rozwiązań instancji była próbkowana w ramach stopniowo rosnących „wycinków". Dla wybranej instancji wycinek S_c stanowi dokładnie S_c procent rozwiązań (wybranych losowo) z całej przestrzeni rozwiązań, które zostały poddane dalszej analizie. W celu zmierzenia wpływu próbkowania na wartość miar LON wykorzystano względny błąd oszacowania (*względne odchylenie* od wartości prawdziwej)

$$\Delta M_i(S_c) = \frac{M_i(100\%) - M_i(S_c)}{M_i(100\%)}, \tag{7.1}$$

gdzie $M_i(S_c)$ to wartość miary i dla wycinka S_c. Dla każdej z miar LON oraz instancji, względny błąd został policzony dla wycinków obejmujących różny procent uwzględnionych rozwiązań: $S_c = 5, 10, \ldots, 95\%$.

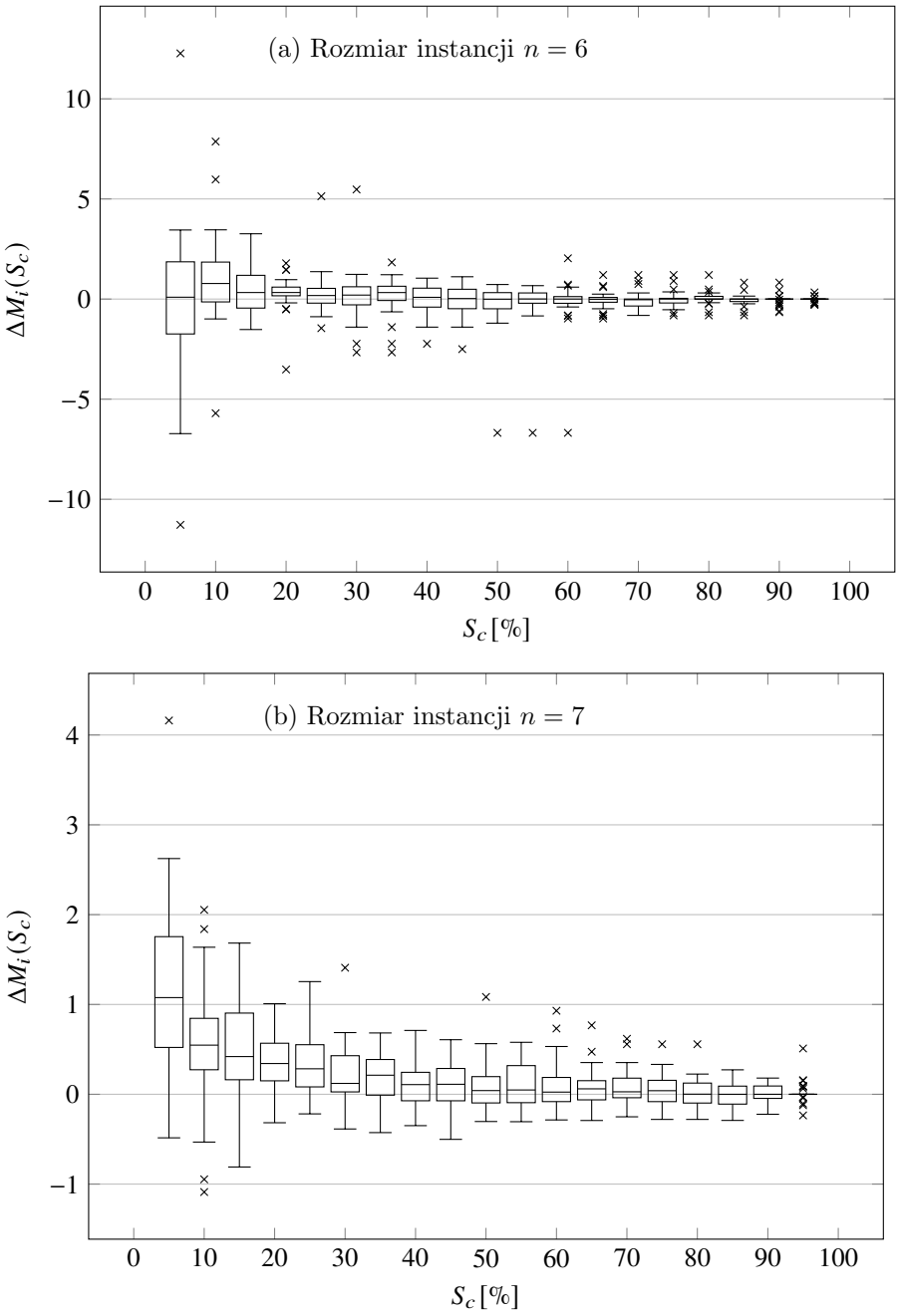
W ramach przeprowadzonych eksperymentów wykorzystano takie miary analizy przestrzeni rozwiązań jak: *assortativity* (miara ta rozbita została ze względu na wykorzystany parametr wierzchołka na: *assortativity-in*, *assortativity-out*, *assortativity-total*, *assortativity-bin* oraz *assortativity-of*), *global clustering coefficient*, *average shortest path to optimum*[1], *minimum as percent*. Miary te zostały szczegółowo opisane w ramach sekcji 5.3.3. Uzyskane wyniki eksperymentów obliczeniowych zostały przedstawione na rysunkach 7.1–7.8, które zostaną wykorzystane do omówienia zaobserwowanej zmienności miar.

7.3.1 Wyniki dla metryki *assortativity-in*

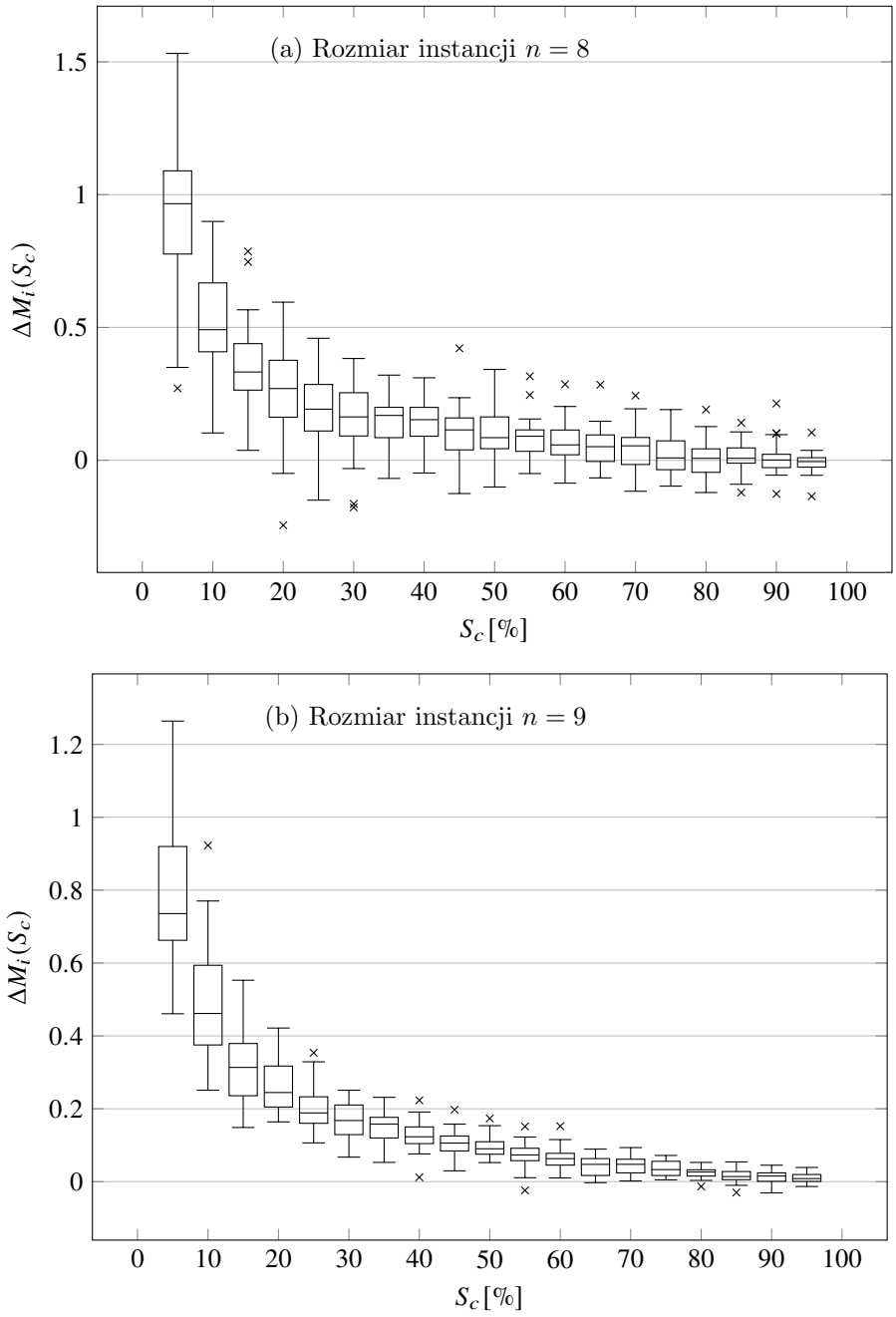
Rezultaty eksperymentów dla dużej część metryk były podobne, stąd zostaną one podsumowane zbiorczo w następnej sekcji. Do pogłębionej analizy wybrano metrykę *assortativity-in*. Średnie względne odchylenie miary *assortativity-in*, w zależności od stopnia próbkowania przestrzeni rozwiązań, zostało przedstawione na rysunkach 7.1a oraz 7.1b. Każdy z rysunków przedstawia wyniki dla instancji o określonym rozmiarze.

Dla najmniejszych instancji o 6 operacjach ($n = 6$, rysunek 7.1a), wartość mediany odchylenia miary znajdowała się w zakresie od 0 do 1. Wartość zbliżona do 0 charakteryzowała najmniejsze, 5% próbkowanie przestrzeni rozwiązań ($S_c = 5\%$). Dla kolejnego wycinka $S_c = 10\%$ zaobserwowany został skok do wartości 1 by następnie stopniowo maleć i ponownie zbliżać się do wartości zbliżonych do 0 dla $S_c = 45\%$. Co ciekawe zależność tą można zaobserwować w jeszcze większym stopniu dla większych instancji problemu. Rozstęp międzykwartylowy jest stosunkowo duży (ponad 2 standardowe odchylenia, co świadczy o dużym zróżnicowaniu miary) dla małych S_c i maleje wraz ze wzrostem stopnia spróbkowania przestrzeni. Dla $S_c = 15\%, \ldots, 60\%$ rozstęp stabilizuje się około 1 odchylenia względnego, po czym można zaobserwować kolejny spadek. Rozkłady empiryczne odchyleń miary mają różne kształty, część z nich cechuje się znaczną skośnością (wydłużone ramię rozkładu). Dane można określić jako heterogeniczne (zróżnicowane), z licznymi wartościami odstającymi w zakresie do 6–10 względnych odchyleń od 0.

[1] Zdecydowano się pozostać przy konwencji angielskich nazw metryk – dlatego nie wykorzystano polskiego odpowiednika, czyli *średnia długość najkrótszej ścieżki do optimum* nawet w wypadku tak bardzo „opisowej" nazwy metryki.



Rysunek 7.1: Względne odchylenie miary *assortativity-in* w zależności od stopnia próbkowania przestrzeni rozwiązań dla instancji problemu o rozmiarze 6 i 7 operacji.



Rysunek 7.2: Względne odchylenie miary *assortativity-in* w zależności od stopnia próbkowania przestrzeni rozwiązań dla instancji problemu o rozmiarze 8 i 9 operacji.

Dla instancji o 7 operacjach ($n = 7$, rysunek 7.1b) największą różnicą są znacznie mniejsza wartość mediany odchylenia miary, z pozostającym wyraźnym trendem spadku wraz ze wzrostem stopnia próbkowania (S_c). Liczba wartości odstających zmniejsza się, choć pozostaje stosunkowo duża. Rozkłady empiryczne odchyleń miary są ponownie zbliżone do rozkładu normalnego, z wyjątkami charakteryzującymi się dużą skośnością.

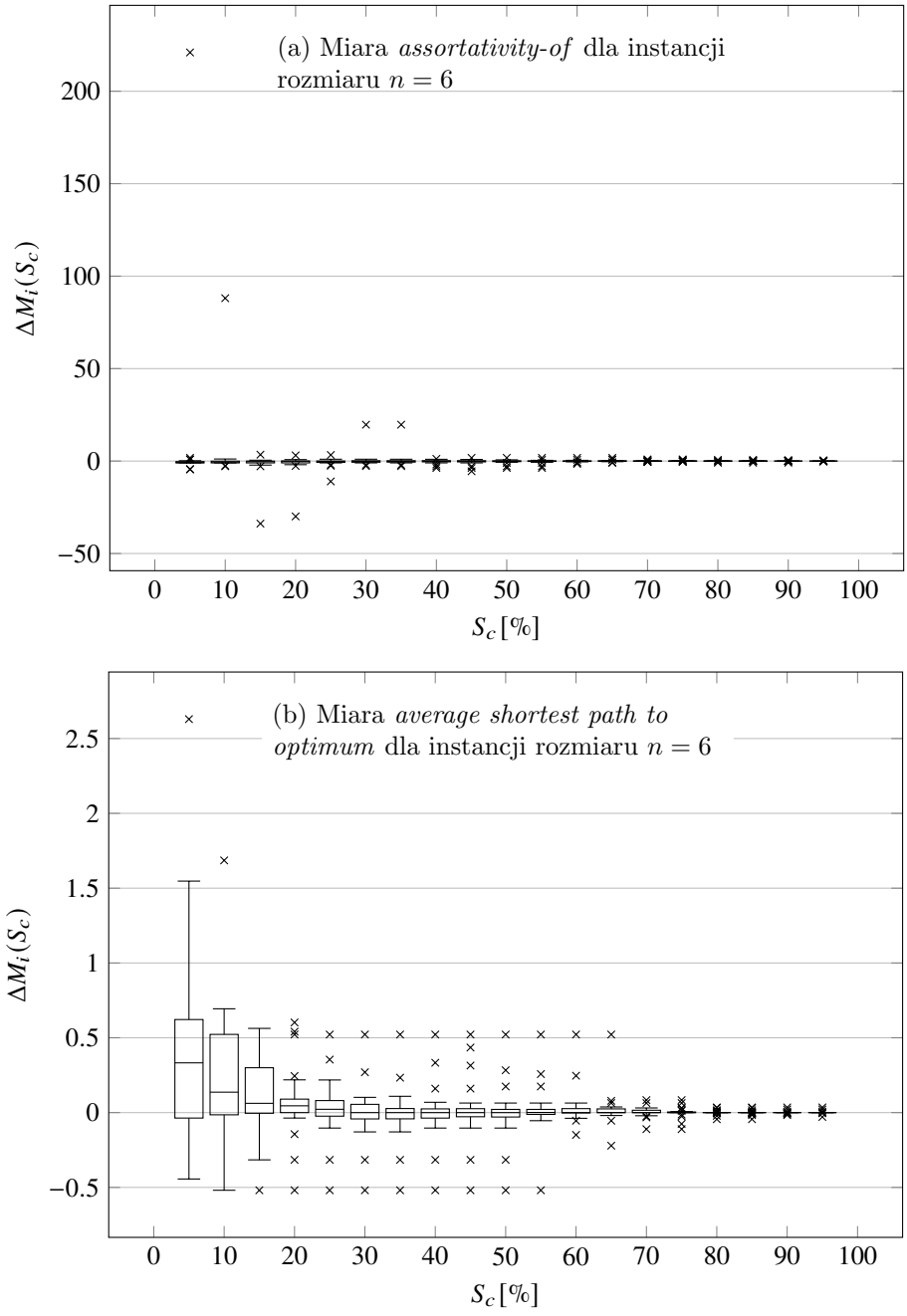
Następny rysunek 7.2a, obrazuje względne odchylenie miary *assortativity-in* dla instancji o rozmiarze $n = 8$. Ponownie mediana odchyleń jest mniejsza, ze znacznie bardziej widocznym trendem spadkowym wraz ze stopniem próbkowania przestrzeni. Dodatkowo, podobny trend spadkowy można zaobserwować dla wariancji odchyleń miary. Oba z przedstawionych trendów utrzymują się dla wyników instancji o rozmiarze $n = 9$ (co zostało przedstawione na rysunku 7.2b).

7.3.2 Wyniki dla pozostałych metryk

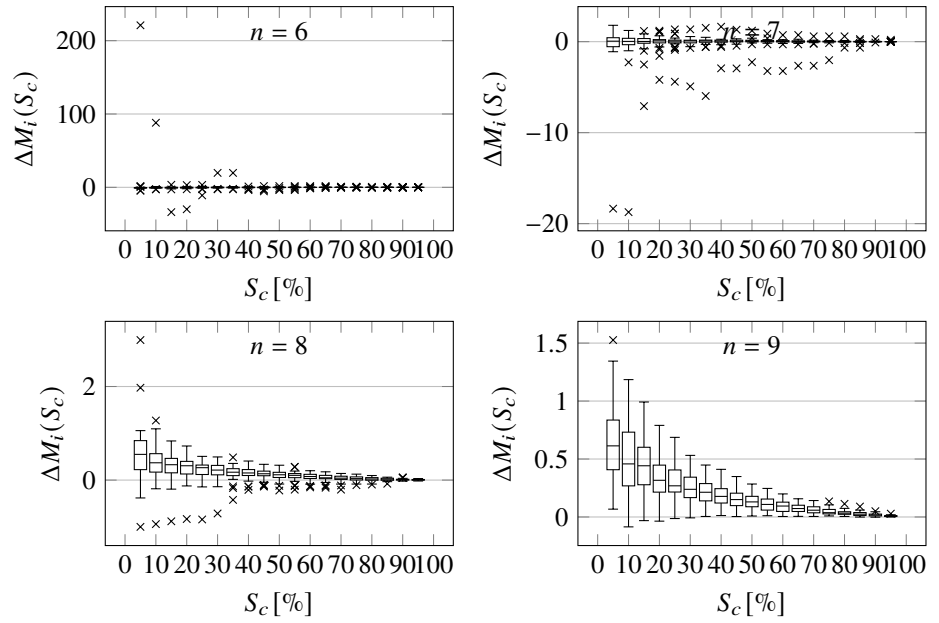
Ze względu na podobieństwo wyników otrzymanych dla pozostałych metryk (rysunki 7.4, 7.5, 7.6, 7.7, 7.8), zostaną one omówione łącznie. Zmiany odchyleń dla metryk *assortativity-out*, *assortativity-total*, *assortativity-of*, *global clustering*, *average shortest path to optimum* czy też *minimum as percent* są podobne do przedstawionych w ramach opisu metryki *assortativity-in*, jednak można zauważyć pewne różnice. Dla metryki *assortativity-out*, w wypadku instancji o rozmiarze $n = 6$ (rysunek 7.3a) wartości odstające znajdują się znacznie dalej od 0 niż dla jakiejkolwiek innej metryki. Można to zaobserwować szczególnie dla wycinków o mniejszym stopniu próbkowania przestrzeni rozwiązań. Tym samym można przypuszczać, że metryka ta charakteryzuje się wysoką podatnością na niewystarczający stopień próbkowania, szczególnie dotkliwą w wypadku mniejszych instancji problemu. Kolejna interesująca obserwacja dotyczy metryki *average shortest path to optimum* (rysunek 7.3b) dla instancji o rozmiarze $n = 6$. Wartości odstające w tym wypadku wykazują tendencję do przyjmowania określonych, „zdyskretyzowanych" wartości. Wynika to prawdopodobnie z małego rozmiaru analizowanych instancji. Odpowiadające im grafy LON są również stosunkowo małe, przez co charakteryzują się dość krótkimi ścieżkami.

7.3.3 Omówienie wyników

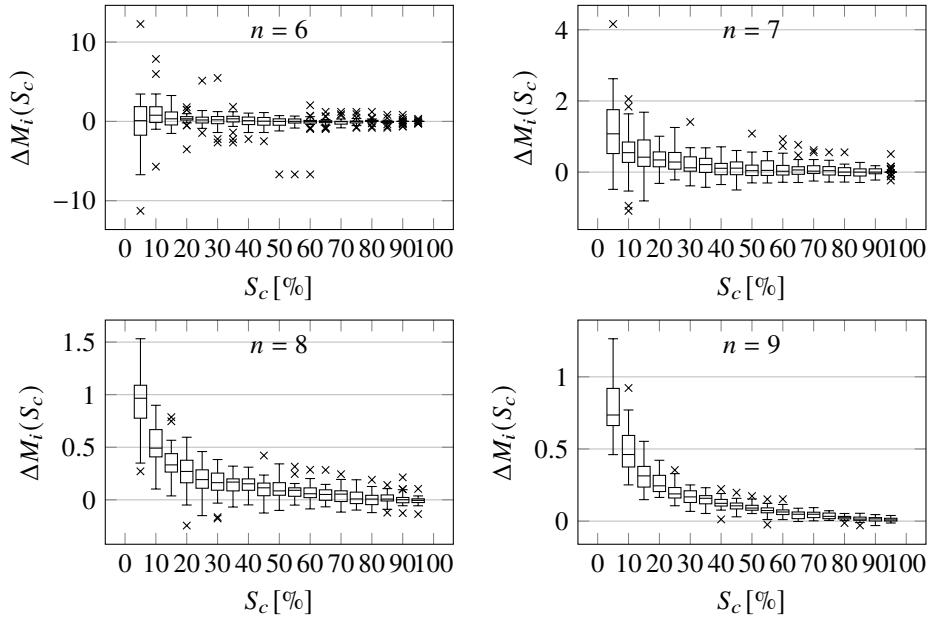
Przedstawione wyniki eksperymentów wskazują na zmienność wartości metryk nie tylko w zależności od stopnia próbkowania przestrzeni S_c ale niestety również od rozmiaru instancji problemu. Wraz ze wzrostem rozmiaru instancji, dla zadanej wartości S_c można zaobserwować spadek wariancji



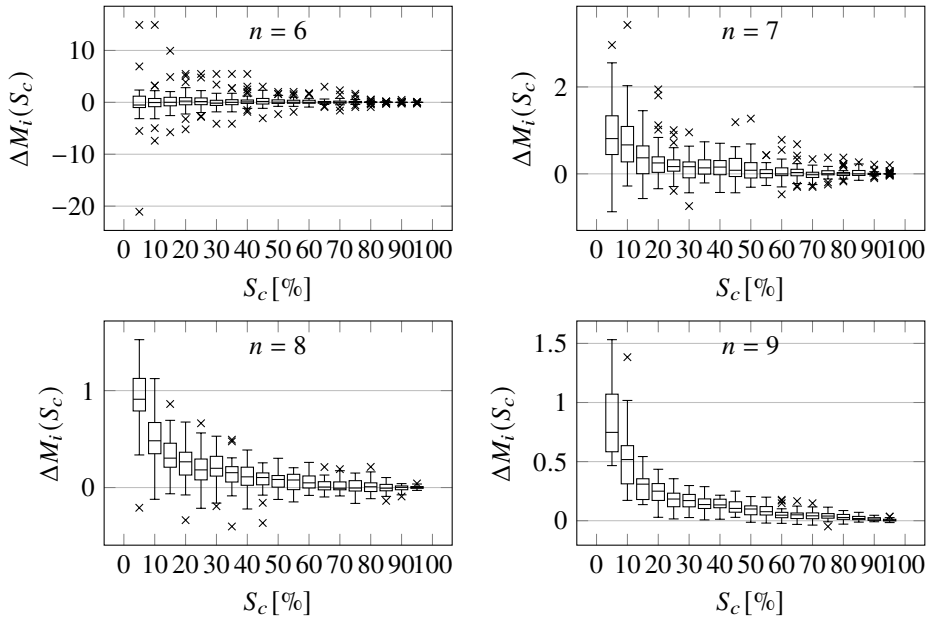
Rysunek 7.3: Przykłady interesujących anomalii w ramach badanych miar.



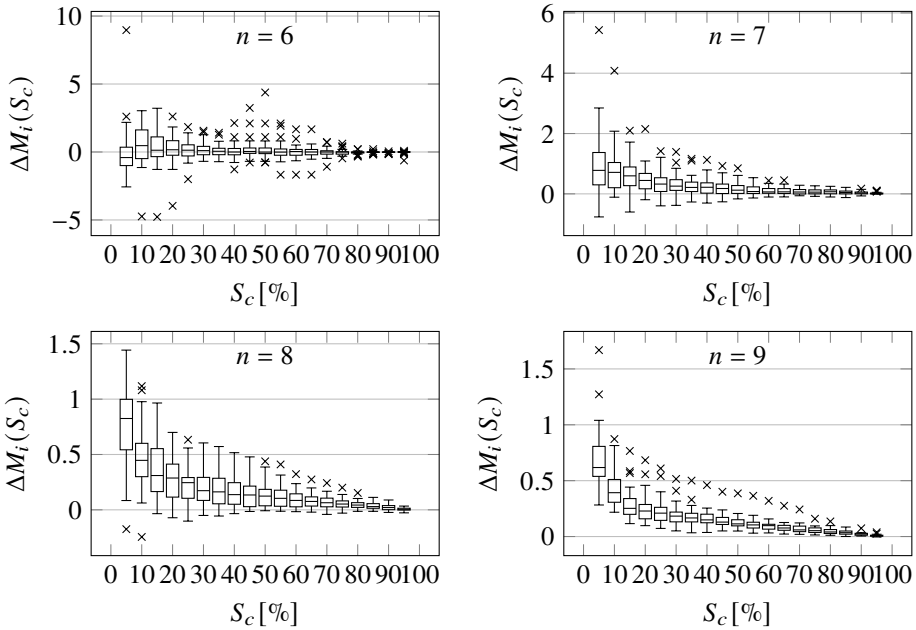
Rysunek 7.4: Wykresy pudełkowe miary *assortativity-of* dla różnych rozmiarów instancji problemów.



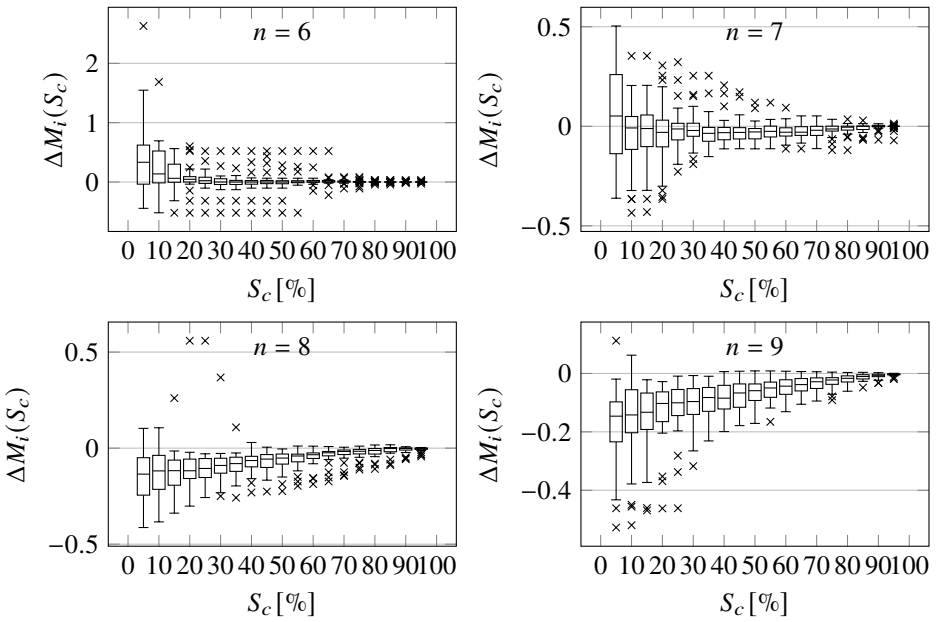
Rysunek 7.5: Wykresy pudełkowe miary *assortativity-in* dla różnych rozmiarów instancji problemów.



Rysunek 7.6: Wykresy pudełkowe miary *assortativity-out* dla różnych rozmiarów instancji problemów.



Rysunek 7.7: Wykresy pudełkowe miary *assortativity-bin* dla różnych rozmiarów instancji problemów.



Rysunek 7.8: Wykresy pudełkowe miary *average shortest path to optimum* dla różnych rozmiarów instancji problemów.

otrzymanych miar. Fakt ten wskazuje na to, że nakład obliczeniowy konieczny do obliczenia wartości miar z zadanym zakresem tolerancji błędu rośnie wolniej od rozmiaru przestrzeni rozwiązań. Jednak określenie czy dla problemu z przestrzenią rozwiązań o wykładniczym rozmiarze jest to wzrost o charakterze wielomianowym (jak to pośrednio założono w [123]), wymaga dalszych badań.

Oprócz wyraźnej zmienności odchyleń miar, ich wartości średnie są zwykle niezerowe (dla większości miar, są one niedoszacowane). Oznacza to, że na wartość oczekiwaną miar wpływają nie tylko cechy instancji (co jest pożądane) ale dodatkowo nakład obliczeniowy włożony w próbkowanie przestrzeni rozwiązań (co jest cechą niekorzystną). Zjawisko jest uzależnione od miary oraz rozmiaru instancji. Scharakteryzowanie tej zależności (w istocie błędu systematycznego) mogłoby umożliwić podjęcie środków zaradczych w postaci wprowadzenia korekty lub modyfikacji metody próbkowania.

7.4 Wnioski i uwagi

Zaprezentowane badania opisują wpływ próbkowania przestrzeni rozwiązań na wartość wybranych miar wykorzystywanych w ramach analizy prze-

strzeni rozwiązań. Wszystkie z badanych miar okazały się podatne na niewystarczający stopień próbkowania przestrzeni. Wraz ze wzrostem liczby próbek zmieniała się zarówno wariancja jak i wartość oczekiwana miar. Stąd, w kontekście testowanych instancji i ruchów, eksperymenty nie wskazują na zasadność stosowania LON *w opisany sposób* jako komponentu bardziej zaawansowanego systemu. Tym samym wykazana została jedna z tez pracy: *sposób przeprowadzenia próbkowania przestrzeni rozwiązań może mieć istotny wpływ na wartości estymacji miar tej przestrzeni.*

Niemniej, uzyskane rezultaty są reprezentatywne jedynie dla stosunkowo małych instancji. Stąd potencjalnym polem dalszych badań mogłyby być eksperymenty na instancjach o większych rozmiarach, charakterystycznych dla popularnych testowych zbiorów danych (na przykład zbiór Taillarda [270]). Interesującym wyzwaniem może być próba określenia skali zmienności miar dla większych instancji, zarówno w kontekście zadanej dostępnej mocy obliczeniowej, jak i konieczności korekcji estymowanych miar.

W ramach szerszego kontekstu dysertacji, metody LON i FLA stanowią jedynie jeden z wielu możliwych sposobów opisu instancji problemów optymalizacji. Choć ich wadą jest niewątpliwie duże zapotrzebowanie na moc obliczeniową, to stanowią sposób na „kompresję" informacji o instancji do postaci kilku liczb (wartości miar). Cecha ta stanowi oczywistą zaletę w kontekście wykorzystania tak uzyskanych miar jako danych wejściowych dla metod sztucznej inteligencji, takich jak sieci neuronowe. Niestety wyniki badań własnych opisanych w tym rozdziale sugerują zbyt dużą zmienność miar LON w zależności od stopnia próbkowania przestrzeni rozwiązań aby uzasadnić opłacalność ich kalkulacji „online" w ramach większego systemu. Wnioski te wydają się tym bardziej zasadne w kontekście braku możliwości próbkowania większej części przestrzeni rozwiązań dla dużych instancji problemów optymalizacji. A to właśnie rozwiązywanie dużych instancji mogłoby stanowić wyzwanie uzasadniające stosowanie złożonego systemu sztucznej inteligencji nadzorującego przeszukiwanie przestrzeni rozwiązań. Tymczasem niewielkie instancje zwykle można rozwiązywać nawet za pomocą metod dokładnych.

Rozdział 8

Przeszukiwanie z zabronieniami wykorzystujące neuronowy mechanizm pamięci

Algorytm przeszukiwania z zabronieniami jest jedną z najbardziej rozpoznawalnych metaheurystyk stosowanych w rozwiązywaniu problemów szeregowania zadań. Jak opisano w podrozdziale 3.2.2, oryginalny schemat zaproponowany przez Glovera [90] został wielokrotnie modyfikowany niezależnie przez wielu autorów, dostosowując go do specyfiki zastosowań. Elementem wyróżniającym wspomniany algorytm jest lista zabronień (tabu). W pierwotnym algorytmie lista tabu jest parametryzowana przez jedną liczbę, określającą jej długość, którą należy dostosować do rozwiązywanego problemu. Dobór tego parametru jest niezwykle istotny, ponieważ zbyt krótka lista tabu nie zapobiega wpadaniu algorytmu w cykle, natomiast zbyt długa lista tabu znacząco ogranicza możliwości eksplorowania przestrzeni rozwiązań, co może doprowadzić do utknięcia algorytmu (brak dozwolonego ruchu). Niestety, nie są znane metody które pozwalałyby na ustalenie optymalnej długości listy tabu dla arbitralnej instancji w dowolnym problemie. W badaniach można spotkać podejścia takie jak:

- ustalona długość listy tabu – np. zalecany rozmiar 7 dla problemów szeregowania [89];
- lista tabu z określonego zakresu zależnego od rozmiaru problemu – np. pomiędzy $3m/2$ do $3m$ w zależności od liczby miast m dla problemu TSP [148] ale zakresy te różnią się w zależności od prac, źródło [284]

zaleca $m/4$ dla sąsiedztwa wyznawczego przez 2-Opt lub w zakresie $m/6$ do $m/8$ dla sąsiedztwa opartego o 3-Opt;
- dynamiczne dostosowanie długości listy zabronień w czasie działania algorytmu [238].

Alternatywnym podejściem jest taka modyfikacja mechanizmu zabronień, która pozwala choć częściowo wyeliminować wspomniane wady, jednocześnie wzbogacając algorytm o elementy inspirowane sieciami neuronowymi.

Niniejszy rozdział opisuje modyfikacje klasycznego algorytmu przeszukiwania z zabronieniami o neuronowy mechanizm zapominania. Zastosowano alternatywne rozwiązanie do klasycznej listy zabronień. Pojedynczy neuron pełni funkcję „pamięci" o rozwiązaniach dzielących pewne wspólne cechy, a dowolne rozwiązanie nie jest jednoznacznie dozwolone lub zabronione ale może być bardziej lub mniej „preferowane". Tym samym wyeliminowana została możliwość „utknięcia" algorytmu (dla klasycznej metody może to nastąpić jeśli wszystkie rozwiązania z otoczenia są zabronione, czyli są na liście tabu). Testy zaproponowanej modyfikacji algorytmu przeprowadzona dla NP-trudnego, cyklicznego problemu gniazdowego – opisanego w wcześniej w ramach podrozdziału 2.6.

W szerszym kontekście dysertacji, przytoczone badania pozwalają wyciągnąć wnioski co do:

1. możliwości ulepszenie klasycznych metod optymalizacji dyskretnej o mechanizmy neuronowe,
2. skuteczności wykorzystania nawet prostych metod neuronowych – co daje podstawy do weryfikacji zasadności wykorzystania bardziej zaawansowanych metod (np. sieci neuronowych z kosztownym obliczeniowo treningiem),
3. odpowiedzi na pytanie: „czy głęboka modyfikacja nawet najbardziej fundamentalnych elementów metaheurystyk może przynosić korzyści, na przykładzie algorytmu TS i listy tabu?"

Pierwsze z pytań bezpośrednio odnosi się do jednej z tez doktoratu: poprawienie efektywności działania konwencjonalnych algorytmów metaheurystycznych, rozumiane jako uzyskiwanie w tym samym koszcie (np. czasowym) rozwiązań o wyższej jakości. Opisane w rozdziale badania własne zostały zaprezentowane na konferencji o zasięgu międzynarodowym, oraz opublikowane w materiałach pokonferencyjnych [29].

8.1 Powiązane badania

Współczesny rynek charakteryzuje się ogromnym zapotrzebowaniem na tanie ale jednocześnie zróżnicowane produkty. Tym samym bardzo ważna

jest cykliczna, masowa produkcja, gdzie produkty są wytwarzane elastycznie z zachowaniem stałego, powtarzającego się interwału czasowego (czas cyklu). Aby zmaksymalizować liczbę wyprodukowanych dóbr w jednostce czasu, należy zminimalizować czas cyklu. W tym celu koniecznie jest najpierw zamodelowanie problemu, a następnie skorzystanie z algorytmów optymalizacyjnych. W zależności od specyfikacji procesu produkcji oraz oczekiwanej dokładności odwzorowania rzeczywistych procesów, można zastosować różne modele problemu, w tym: ogólny problem linii produkcyjnej (ang. *General Assembly Line Problem*) [13] lub cykliczne wersje klasycznych problemów szeregowania zadań. W ramach przedstawionych badań zdecydowano się na podejście drugie, wykorzystując cykliczny problem gniazdowy, należący do klasy problemów silnie NP-trudnych. Ze względu na silną NP-trudność, konieczne jest wykorzystywanie heurystyk oraz metaheurystyk, ponieważ algorytmy dokładne mogą być wykorzystane jedynie do rozwiązywania stosunkowo małych instancji takich problemów. Przy planowaniu badań wykorzystano analizę stanu wiedzy na temat wariantów cyklicznego problemu gniazdowego, opartą na pracy [166].

Aby rozwiązać rozważany problem, wybrano przeszukiwanie z zabronieniami (TS), bazując na jego częstym wykorzystaniu do rozwiązywania kombinatorycznych problemów optymalizacji [45, 12], w tym cyklicznych problemów wielomaszynowych [39, 30]. Dodatkowo pożądaną cechą w kontekście eksperymentów numerycznych jest deterministyczny charakter algorytmu. Pozwala on na rozważanie pojedynczego wykonania algorytmu dla określonej kombinacji hiperparametrów oraz instancji, zamiast konieczności wielokrotnego jego uruchamiania (jak np. dla SA i GA).

W ramach badań własnych, zdecydowano się na wykorzystanie mechanizmu bazującego na zachowaniu pojedynczego neuronu, koncepcji pochodzącej z badań nad wykorzystaniem sztucznych sieci neuronowych. Mechanizm ten pozwala na unikanie ponownego wykorzystania rozwiązań już odwiedzonych, jednocześnie zapewniając lepszą dywersyfikację w procesie przeszukiwania przestrzeni rozwiązań. Rozwiązanie to wpisuje się w obiecujący trend wykorzystania zaawansowanych modeli matematycznych w rozwiązywaniu trudnych problemów optymalizacji: analizy teoretycznej wykorzystania wiele ruchów typu zamień [36], dostosowania TS do środowiska klastrowego złożonego z wielu jednostek GPU [31], tworzenia nowego rodzaju otoczenia dedykowanego do rozwiązania problemu [34], czy też wykorzystania własności blokowych dla kolejnych problemów optymalizacyjnych [39]. W szczególności podejście neuronowe zostało wykorzystanie między innymi w rozwiązywaniu problemu komiwojażera [103], oraz innych problemów szeregowania zadań [33, 35].

8.2 Opis algorytmu NTS

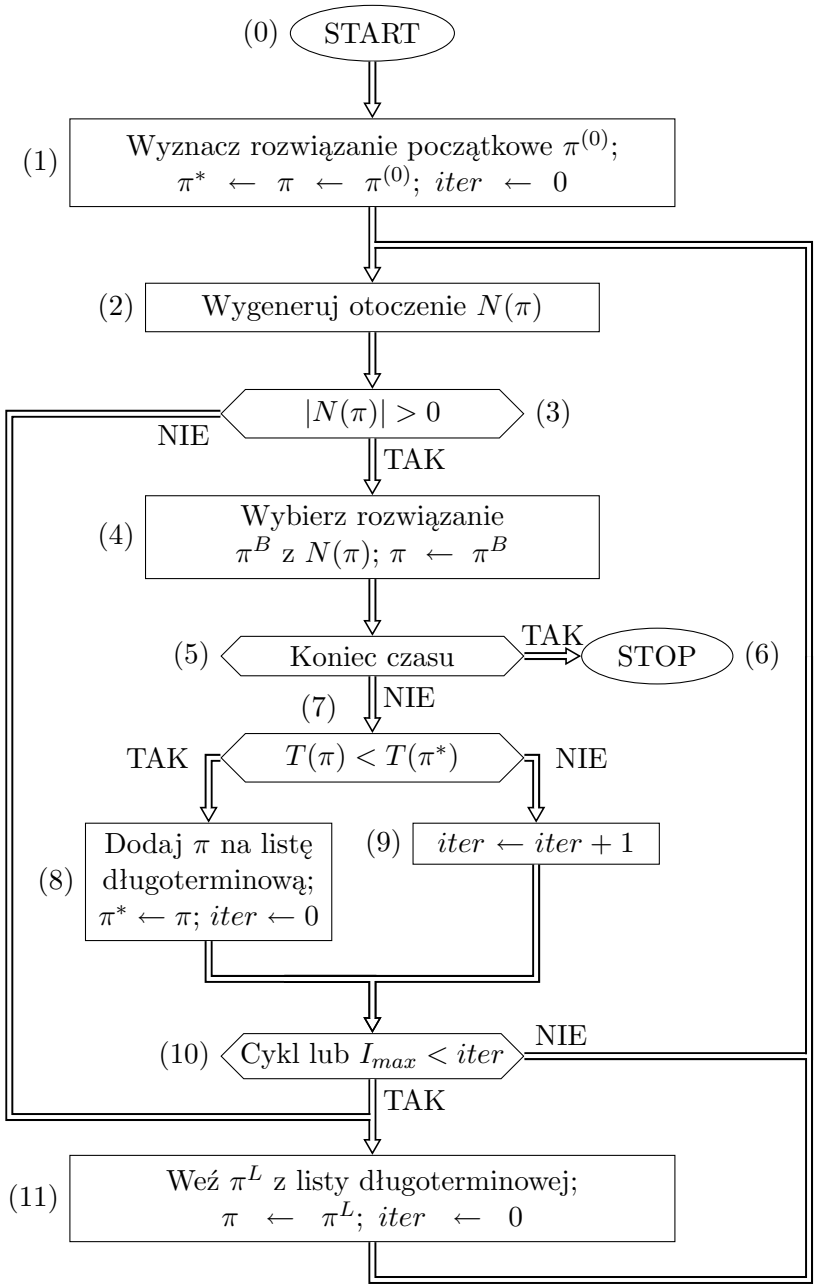
W zaproponowanym podejściu wykorzystano modyfikację klasycznej metaheurystyki, przeszukiwania z zabronieniami, opisanej wcześniej w podrozdziale 3.2.2. W literaturze można spotkać wiele przykładów adaptacji oryginalnej metody [17, 269]. W przeprowadzonych badaniach, modyfikacji poddano sposób wyboru rozwiązania z otoczenia, zastępujący klasyczną listę zabronień.

Schemat przedstawiający działanie zmodyfikowanej wersji algorytmu przedstawia rysunek 8.1. Rysunek ten zostanie wykorzystany do szczegółowego wyjaśnienia każdego z kroków zaproponowanego algorytmu – przeszukiwania z zabronieniami o neuronowym mechanizmie pamięci, NTS (z ang. *Neuro Tabu Search*). Numery w nawiasach odpowiadają numerom bloków na wcześniej wspomnianym schemacie.

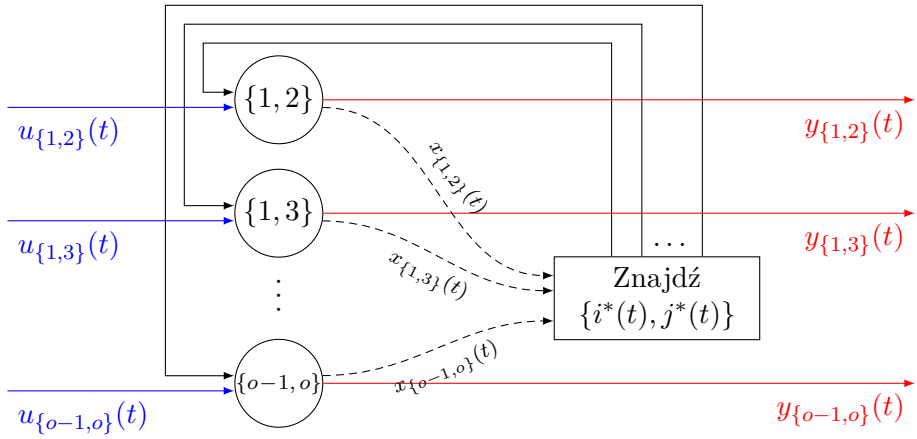
Algorytm rozpoczyna działanie (1) od dozwolonego rozwiązania oznaczonego jako π^0. Procedura startowa jest prostą heurystyką, która wyznacza kolejność operacji na maszynach w taki sposób, że każda z operacji z zadania i jest wykonywana przed operacją z zadania j, o ile $i < j$. Najlepsza dotychczas znaleziona kolejność oznaczona jest jako π^*, podczas gdy obecne rozważane rozwiązanie jest oznaczone jako π, a *iter* jest numerem iteracji algorytmu od ostatniej poprawy kolejności π^*.

Blok (2) przedstawia sposób generowania otoczenia (sąsiedztwa). Sąsiedztwo $N(\pi)$ reprezentuje zbiór wszystkich rozwiązań wygenerowanych z kolejności π, poprzez zastosowanie określonego operatora ruchu (operatory zostały wcześniej opisane w ramach sekcji 3.2.2, wraz z przykładami na rysunku 3.1). W ramach badań algorytmu NTS wykorzystano otoczenie blokowe N_1, które zostało zaimplementowane zgodnie z modelem opisanym w pracy Nowickiego i Smutnickiego [198]. Opis tego otoczenia w kontekście problemów cyklicznych został przedstawione w ramach pracy [45]. Wybór otoczenia N_1 pozwala ograniczyć rozmiar generowanego sąsiedztwa, tym samym przyspieszając średni czas wykonywania pojedynczej iteracji algorytmu. Elementy otoczenia są tworzone poprzez zamianę kolejności wykonywania pierwszej operacji z każdego bloku, z drugą operacją z bloku; oraz ostatniej operacji z bloku, z operacją przedostatnią z bloku.

Jeśli w ten sposób stworzone otoczenie jest puste – warunek ten jest sprawdzany w ramach bloku (3) na schemacie – następuje nawrót zgodnie z blokiem (11). Mechanizm nawrotu ma zapewnić dywersyfikację przeszukiwania poprzez opuszczenie mało obiecujących obszarów przestrzeni rozwiązań. Aby była możliwość stosowania takiego mechanizmu, konieczne jest utworzenie i uaktualnianie listy długoterminowej obiecujących rozwiązań. Umieszczane na niej są rozwiązania o wartości funkcji celu lepszej



Rysunek 8.1: Schemat blokowy zaimplementowanego wariantu przeszukiwania z zabronieniami, wykorzystanego w ramach badań nad neuronowym mechanizmem pamięci.



Rysunek 8.2: Schemat neuronowego mechanizmu wyboru rozwiązania z otoczenia dla zmodyfikowanej wersji metaheurystyki przeszukiwania z zabronieniami z neuronowym mechanizmem pamięci.

niż dotychczas najlepsze znalezione rozwiązanie (czyli lepsze niż obecne π^*). Oprócz samego rozwiązania przechowywane jest również otoczenie wygenerowane z tego rozwiązania, z pominięciem najlepszego rozwiązania z tego otoczenia. Pozwala to zapobiec odtworzenia tej samej trajektorii przeszukiwania. W wypadku nawrotu do obiecującego rozwiązania, wybierane jest inne rozwiązanie niż we wcześniejszym przebiegu algorytmu, ponieważ wtedy wybrane rozwiązanie nie jest obecne w otoczeniu. Nawrót wykonywany jest wyłącznie jeśli:

1. otoczenie wygenerowane na podstawie obecnego rozwiązania jest puste – blok (3) na schemacie;
2. algorytm cyklicznie powraca do tych samych rozwiązań – warunek sprawdzany w ramach bloku (10);
3. liczba iteracji bez poprawy najlepszego rozwiązania (*iter*) przekracza określony z góry parametr I_{max} – warunek z bloku (10).

Podczas nawrotu wykorzystywane jest rozwiązanie „najstarsze" na liście długoterminowej (co odpowiada działaniu kolejki LIFO – *Last In, First Out*), przy czym rozwiązane to jest usuwane z tej listy. Warunek wykonania nawrotu opisany w ramach bloku (10) został zastosowany ze względu na wcześniejsze, wstępne badania obliczeniowe. Algorytm pozbawiony tego warunku ma tendencje do wpadania w cykle, ponieważ zaproponowany mechanizm neuronowy (opisany w ramach bloku (4)) nie posiada „wbudowanego", rygorystycznego mechanizmu zapobiegania cyklom.

Następnym krokiem metody jest neuronowy sposób wyboru najlepszego rozwiązania z otoczenia (blok (4)), czemu algorytm zawdzięcza swoją nazwę. Mechanizm ten został przedstawiony na rysunku 8.1, jest to rozwiązanie podobne do zaproponowanego w pracy [259]. W kontraście do rozwiązania opartego o klasyczną listę tabu, nie definiuje ono zbioru ruchów jako ściśle zabronionych. Zamiast tego, każdy z możliwych $o(o-1)/2$ ruchów typu „zamień" jest modelowany z wykorzystaniem odpowiadającego mu neuronu. Neuron $\{i,j\}$ odpowiada ruchowi zamiany kolejności wykonywania operacji i i j, $i \neq j$. Na wejście neuronu $\{i,j\}$ w iteracji t, oznaczone jako $u_{\{i,j\}}(t)$, podawana jest wartość 0, o ile ruch $\{i,j\}$ nie został wykorzystany do stworzenia otoczenia w iteracji t, lub wartość 1 w przypadku przeciwnym (czyli, gdy neuron reprezentuje rozwiązanie z którego powstało otoczenie w iteracji t). Wartość wyjścia neuronu – zapisywana jako $y_{\{i,j\}}(t)$ – służy jako informacja, na ile „pożądany" jest odpowiadający mu ruch $\{i,j\}$. Wybór kolejności dla kolejnej iteracji algorytmu $\pi^B \in N(\pi)$ bazuje na wygenerowaniu wartości dla każdego z neuronów, a następnie wyboru ruchu odpowiadającego neuronowi o największej wartości na wyjściu. Stan neuronu opisany jest przez zmienne

- *Efekt tabu*, oznaczony przez $\gamma_{\{i,j\}}(t)$, odzwierciedla historię aktywacji neuronu, przy czym wartość ta zależy od stanu wyjścia neuronu $y_{\{i,j\}}(t)$ w poprzednich iteracjach. W pracy [259] zaproponowano efekt tabu o funkcji malejącej wykładniczo:

$$\gamma_{\{i,j\}}(t+1) = \begin{cases} k\gamma_{\{i,j\}}(t) + y_{\{i,j\}}(t) & \text{dla} \quad t > 0, \\ 0 & \text{dla} \quad t = 0, \end{cases} \tag{8.1}$$

  gdzie stała $k \in (0,1)$ jest współczynnikiem pozwalający na modyfikację tempa spadku.
- *Efekt wzmocnienia*, wykorzystywany jako wyznacznik jakości odpowiadającego neuronowi ruchu, zapisywany za pomocą równania:

$$\eta_{\{i,j\}}(t+1) = \begin{cases} \alpha \frac{T\left(\pi(t)_{\{i,j\}}\right)}{T(\pi^*(t))} & \text{dla} \quad u_{\{i,j\}}(t+1) = 1, \\ \infty & \text{dla} \quad u_{\{i,j\}}(t+1) = 0, \end{cases} \tag{8.2}$$

  gdzie $\pi(t)_{\{i,j\}}$ jest permutacją powstałą w wyniku zamiany w kolejności operacji i oraz j w rozwiązaniu π z iteracji t, α jest współczynnikiem skalującym, $\pi^*(t)$ to najlepsza znana w iteracji t kolejność wykonywania operacji.

Wartość na wyjściu neuronu jest wyliczana za pomocą wzoru:

$$y_{\{i,j\}}(t+1) = \frac{1}{2}\Big|\{i,j\} \cap \{i^*(t), j^*(t)\}\Big|, \tag{8.3}$$

gdzie $i^*(t)$ i $j^*(t)$ są ustalonymi operacjami spełniającymi warunek

$$x_{\{i^*(t),j^*(t)\}}(t) = \min\left\{x_{\{k,l\}}(t) \;\middle|\; k,l \in \mathcal{O} \wedge k \neq l\right\}, \tag{8.4}$$

$$x_{\{i,j\}}(t) = \eta_{\{i,j\}}(t) + \gamma_{\{i,j\}}(t). \tag{8.5}$$

Co za tym idzie, wartość $y_{\{i,j\}}(t) \in \{0,0{,}5,1\}$ odpowiada za stopień podobieństwa ruchu $\{i,j\}$ do najlepszego ruchu $\{i^*,j^*\}$. Bloki (5) oraz (7) odpowiadają za warunek przerwania wykonywania algorytmu po upływie zadanego czasu. W blokach (7–9) uaktualniana jest dotychczas najlepsza znaleziona kolejność wykonywania zadań π^*, następuje aktualizacja listy pamięci długoterminowej oraz licznika iteracji. Bloki (10–11) były już omówione wcześniej.

8.3 Opis algorytmu referencyjnego

W celu oceny efektywności zastosowanej metody NTS i możliwości miarodajnego porównania wyników, wykorzystano wyniki implementacji algorytmu przeszukiwania z zabronieniami (TS) jako bazy porównawczej. W tym celu zaimplementowano algorytm TS możliwe podobnie do proponowanej metody NTS, z jedyną różnicą w działaniu ograniczoną do sposobu wyboru najlepszego rozwiązania z otoczenia. Tym samym, porównywane metody NTS i TS różniły się jedynie metodą wykonywania bloku (4).

W implementacji TS, mechanizm zabronienia został zaimplementowany jako standardowa lista zabronień, odpowiadająca za krótkoterminową historię przeszukiwania. Rozwiązania są reprezentowane poprzez nieuporządkowaną parę operacji $\{i,j\}$, $v_i = v_j = m$, których zamiana kolejności wykonywania na maszynie m doprowadziła do powstania rozwiązania π (ruch typu zamień). Jeżeli długość listy tabu osiąga maksymalną, zadaną wartość $L_{\max}$, przed dodaniem cech kolejnego rozwiązania najstarszy element znajdujący się na liście tabu jest kasowany, tak aby zachować ograniczoną pojemność listy. W przypadku wygenerowania otoczenia, którego wszystkie odpowiadające rozwiązania znajdują się na liście tabu, lista ta jest sukcesywnie skracana poprzez usuwanie najstarszych elementów, aż co najmniej jedno z rozwiązań w otoczeniu nie przestanie być zabronione. Dodatkowo, zastosowano kryterium aspiracji – każde rozwiązanie z sąsiedztwa które stałoby się nowym najlepszym rozwiązaniem jest „dozwolone", nawet jeśli stan listy tabu nakazuje je zabronić. Strategią wyboru rozwiązania z otoczenia był wybór rozwiązania o najmniejszej wartości funkcji celu.

8.4 Eksperymenty obliczeniowe

Zarówno algorytm NTS jak i TS zostały zaimplementowane w języku programowania *C++*, wykorzystano przy tym środowisko programistyczne *Visual Studio 2015 Professional*. Badania wykonano na pojedynczym stanowisku komputerowym klasy PC, wyposażonym w procesor CPU *Intel i7-4930K*, taktowanym częstotliwością 4,31 GHz, 32 GB pamięci RAM oraz systemem operacyjnym *Windows 8.1* w wersji 64 bitowej. Z uwagi na brak dedykowanych instancji testowych dla cyklicznego problemu gniazdowego, zdecydowano się zaadaptować instancje pokrewnego problemu. Skorzystano z popularnego zbioru danych *OR-Library* [14], z którego wykorzystano instancje: `ft6`, `ft10`, `ft20` oraz `la01-la40`, przeznaczone dla problemu gniazdowego z kryterium $C_{\max}$. Rozwiązywano je jednak jako instancje cyklicznego problemu gniazdowego – zakładając, że operacje wykonywane są cyklicznie. Badania wstępne pozwoliły na dobór hiperparametrów obu algorytmów: dla algorytmu TS zdecydowano się przyjąć wartości: $L_{\max} = 7$, $I_{max} = 5000 + t/10$, gdzie t jest numerem iteracji. Dla algorytmu NTS przyjęto: $\alpha = 12$, $k = 0{,}98$ oraz $I_{max} = 5000 + t/10$. W celu oceny jakości rozwiązania instancji problemu, wykorzystano odchylenie względne najmniejszego wyznaczonego minimalnego czasu cyklu $T(\pi)$ od jego dolnego oszacowania T^{LB}

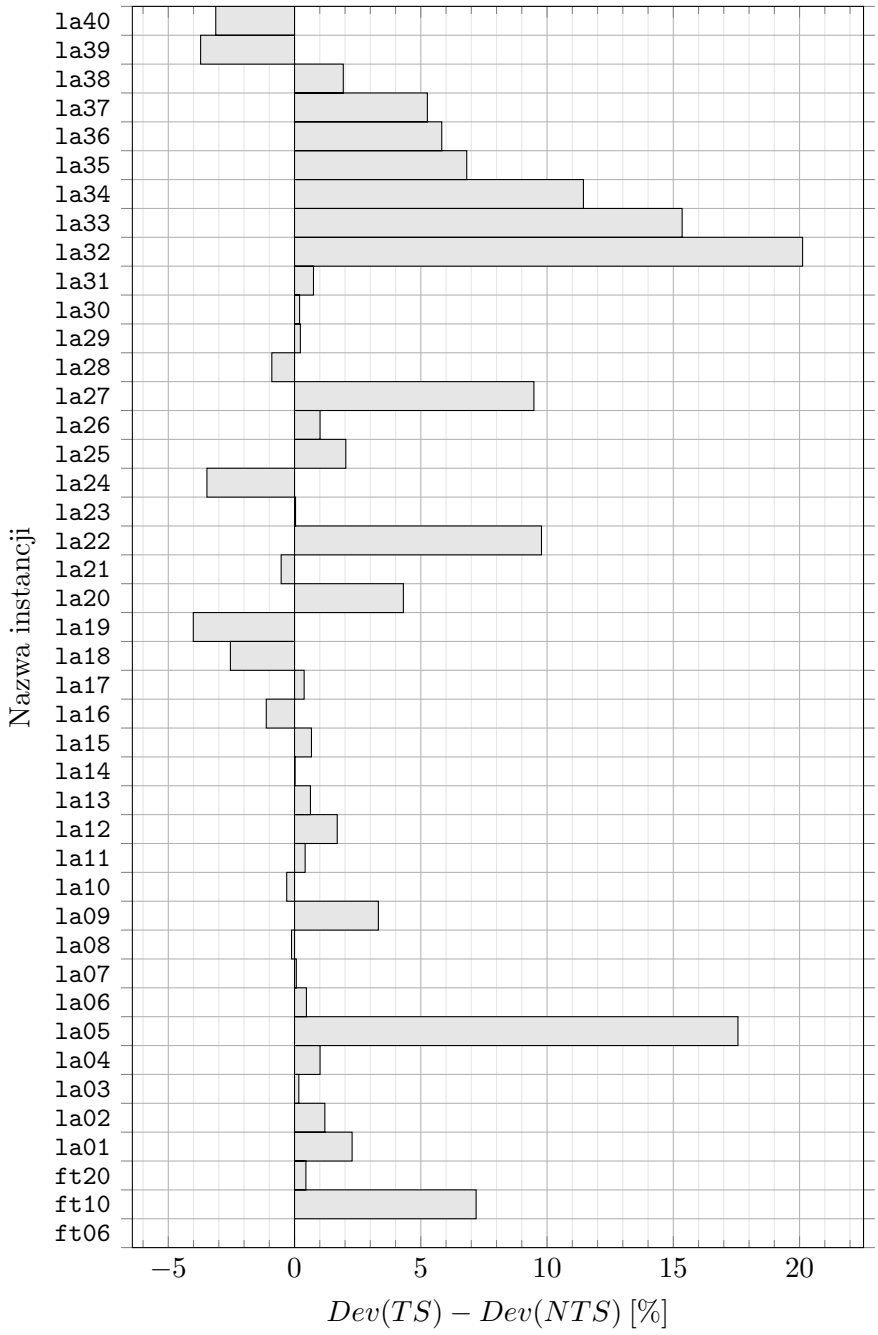
$$Dev = \frac{T(\pi) - T^{\mathrm{LB}}}{T^{\mathrm{LB}}} \cdot 100\,\%, \quad T^{LB} = \max_{1 \leqslant i \leqslant m} \left\{ \sum_{j \in O_i} p_i \right\}. \tag{8.6}$$

Obliczenia dla każdego z algorytmów ograniczono do czasu 100 [s]. Rezultaty przeprowadzonych eksperymentów obliczeniowych znajdują się w tabeli 8.1, wraz z wizualizacją porównania wyników na rysunku 8.3. Algorytm NTS uzyskał lepsze rezultaty dla instancji problemu o większym rozmiarze. Choć średnia wartość Dev dla NTS jest niższa $13,34\% - 10,67\% = 2,67\%$, to dla wypadku instancji: `la08`, `la10`, `la16`, `la18`, `la19`, `la21`, `la24`, `la28`, `la39`, `la40` nie odnotowano poprawy.

W związku z brakiem jednoznacznej poprawy dla wszystkich instancji, zdecydowano się wykorzystać test statystyczny Wilcoxona-Manna-Whitney'a dla dwóch prób. Założono przy tym, że badane przykłady testowe są reprezentatywną próbą populacji problemu. Zdefiniowano przy tym D_{TS} jako populację względnych odchyleń najmniejszego wyznaczonego przez algorytm TS minimalnego czasu cyklu $T(\pi)$, od jego oszacowania dolnego $T(\pi^{LB})$, dla wszystkich instancji problemu. Populacja D_{NTS} została zdefiniowana w sposób analogiczny. Przyjęto hipotezę zerową H_0: prawdopodobieństwo, że losowo wybrana obserwacja z populacji D_{TS} będzie miała więk-

Tabela 8.1: Wartości odchylenia względnego najmniejszego czasu cyklu od dolnego oszacowania dla algorytmów TS i NTS. Pogrubiono mniejsze wartości odchylenia Dev, świadczące o przewadze jednego z algorytmów.

Instancja		Dev [%]		Instancja		Dev [%]	
Nazwa	$n \times m$	TS	NTS	Nazwa	$n \times m$	TS	NTS
la01	10×5	2,79	**0,51**	la23	15×20	1,21	**1,17**
la02	10×5	2,65	**1,45**	la24	15×20	**8,77**	12,24
la03	10×5	2,62	**2,45**	la25	15×20	12,19	**10,16**
la04	10×5	8,01	**7,00**	la26	20×10	5,52	**4,51**
la05	10×5	29,02	**11,47**	la27	20×10	15,09	**5,61**
la06	15×5	14,31	**13,84**	la28	20×10	**0,89**	1,78
la07	15×5	11,07	**11,00**	la29	20×10	3,41	**3,19**
la08	15×5	**0,44**	0,56	la30	20×10	17,21	**17,01**
la09	15×5	3,74	**0,42**	la31	30×10	9,13	**8,38**
la10	15×5	**0,40**	0,71	la32	30×10	21,03	**0,91**
la11	20×5	12,39	**11,97**	la33	30×10	15,76	**0,41**
la12	20×5	1,96	**0,27**	la34	30×10	18,93	**7,49**
la13	20×5	4,23	**3,61**	la35	30×10	7,80	**0,98**
la14	20×5	0,02	**0,00**	la36	15×15	27,36	**21,53**
la15	20×5	8,32	**7,65**	la37	15×15	47,37	**42,11**
la16	10×10	**34,88**	36,00	la38	15×15	43,16	**41,23**
la17	10×10	8,54	**8,16**	la39	15×15	**19,76**	23,48
la18	10×10	**28,84**	31,37	la40	15×15	**15,92**	19,05
la19	10×10	**16,96**	20,98	ft06	6×6	11,24	**11,24**
la20	10×10	16,67	**12,35**	ft10	10×10	17,35	**10,16**
la21	15×20	**17,23**	17,76	ft20	20×5	6,36	**5,92**
la22	15×20	21,05	**11,27**	**średnia**		13,29	**10,68**



Rysunek 8.3: Różnice w jakości uzyskanych wyników dla instancji testowych, wyrażona jako procentowa różnica odchyleń. Wartości pozytywne wskazują na przewagę zmodyfikowanej wersji NTS.

szą wartość niż losowo wybrana obserwacja z populacji D_{NTS} jest równa prawdopodobieństwu, że losowo wybrana obserwacja z populacji D_{NTS} będzie miała większą wartość niż losowo wybrana obserwacja z populacji D_{TS} (tj. $P(D_{TS} > D_{NTS}) = P(D_{TS} < D_{NTS})$). Jako hipotezę alternatywną H_1 przyjęto: $P(D_{TS} > D_{NTS}) \neq P(D_{TS} < D_{NTS})$. Test nie rozstrzyga przypadku $P(D_{TS} = D_{NTS})$. Do przeprowadzenia testu statystycznego wykorzystano pakiet obliczeniowy R, uzyskując wartości $V = 736$, $p = 0{,}000763$. Tym samym dla założonego poziomu istotności statystycznej $p = 0{,}05$, odrzucono hipotezę H_0 na rzecz hipotezy H_1. Oznacza to, że na mocy przeprowadzonego testu statystycznego można powiedzieć, że rezultaty uzyskane przy pomocy zmodyfikowanego algorytmu NTS są istotnie różne od rezultatów uzyskanych z wykorzystaniem klasycznego algorytmu TS na zbiorze danych *OR-Library* [14].

8.5 Wnioski i uwagi

Przedstawione badania prezentują rezultaty wykorzystania zmodyfikowanego algorytmu przeszukiwania z zabronieniami o neuronowym mechanizmie pamięci. Rozwiązanie inspirowane jest biologicznym mechanizmem zapominania, przy czym żadna operacja ruchu nie jest jednoznacznie zakazana lub dozwolona. Na mocy przeprowadzonego testu statystycznego Wilcoxona-Manna-Whitney'a, rozwiązania uzyskane na instancjach z *OR-Library* [14] z wykorzystaniem zmodyfikowanej metody NTS można uznać za statystycznie różne od tych uzyskanych na tych samych danych za pomocą klasycznej metaheurystyki przeszukiwania z zabronieniami. Z kolei porównanie średniej jakości otrzymywanych rozwiązań, wskazuje na przewagę NTS nad TS. Przedstawione rezultaty pozwoliły potwierdzić jedną z tez pracy: *korzystając ze sztucznych sieci neuronowych lub analizy krajobrazu przestrzeni rozwiązań możliwe jest poprawienie efektywności działania konwencjonalnych algorytmów metaheurystycznych, rozumiane jako uzyskiwanie w tym samym koszcie (np. czasowym) rozwiązań o wyższej jakości*, w zakresie korzyści stosowania mechanizmów neuronowych. Dodatkowo okazało się, że korzystnej modyfikacji można poddawać nawet najbardziej kluczowe i fundamentalne elementy metaheurystyk – w tym przypadku listę tabu w algorytmie TS.

Kontynuacja opisanych badań mogłaby obejmować między innymi następujące kroki: zmianę modelu neuronu w mechanizmie zabronień, stworzenie dedykowanego rozwiązania zapobiegającego powstawaniu cykli, zrównolegleniem zaproponowanego rozwiązania lub wykorzystaniem wytrenowanej sieci neuronowej zamiast mechanizmu neuronowego. Uzyskane w ramach

dysertacji rezultaty, można wykorzystać jako swojego rodzaju uzasadnienie wykorzystania metod sztucznej inteligencji w rozwiązywaniu problemów optymalizacji dyskretnej. Szczególnie jako przesłankę za zasadnością prób stosowania metod bardziej wymagających – w tym tych o większym nakładzie mocy obliczeniowej (np. wymagających procedury treningu). Nakłady takie jednak muszą zostać zmierzone bądź chociaż oszacowane aby następnie możliwe było wzięcie ich pod uwagę w kontekście zasadności i praktyczności wykorzystania innowacyjnych, neuronowych rozwiązań.

Rozdział 9

Uczenie głębokie ze wzmocnieniem w optymalizacji dyskretnej

Do rozwiązywania dużych instancji trudnych problemów optymalizacji dyskretnej stosuje się głównie metody przybliżone, takie jak metaheurystyki, co zostało opisane wcześniej w rozdziale 2. Nie dają one gwarancji znalezienia rozwiązania optymalnego ale pozwalają znaleźć rozwiązanie które jest „wystarczająco dobre" w „akceptowalnym" czasie. Choć istnieją bardzo „ogólne" metaheurystyki, możliwe do zastosowania w wielu problemach optymalizacji, to w sporej liczbie przypadków konieczne jest ich dostosowanie i modyfikacja. Opiera się ona często o wykorzystanie istniejącej wiedzy o problemie (np. poprzez eksploatowanie własności problemu) tak aby proces przeszukiwania przestrzeni rozwiązań był bardziej efektywny, szczególnie w porównaniu do wersji nie wykorzystujących tej wiedzy. Dla przykładu, choć algorytm przeszukiwania z zabronieniami może posłużyć do rozwiązywania właściwie dowolnego problemu optymalizacji dyskretnej, to zastosowanie sąsiedztwa blokowego [198] dla problemu gniazdowego pozwala „lepiej" prowadzić proces przeszukiwania. Otoczenie blokowe odrzuca wiele rozwiązań, które nie dają szansy na poprawę funkcji celu, tym samym zwiększając efektywność metody poprzez ignorowanie rozwiązań nieobiecujących na rzecz eksploracji większej liczby potencjalnie dobrych rozwiązań. Aby opracować tego typu modyfikacje algorytmów wymagana jest jednak wcześniejsza żmudna praca zdobycia wiedzy o problemie. Niestety, algorytm dostosowany w taki sposób nie musi osiągać równie dobrych rezultatów po zmianie problemu, na przykład w wyniku dodania dodatkowych ograniczeń. W takiej sytuacji pojawia się konieczność ponownego pozyskania wiedzy.

Pewną nadzieją na ułatwienie tak wymagającego i pracochłonnego procesu jest potencjalna możliwość wykorzystania uczenia maszynowego – co pozwalałoby zautomatyzować zarówno etap pozyskiwania wiedzy, jak i jej wykorzystania. Perspektywa ta jest tym bardziej kusząca, że istnieje dostęp do danych w postaci historii przebiegu pracy wielu zróżnicowanych algorytmów dla problemów optymalizacji. Zwykle jednak informacje te nie są ani zbierane, ani wykorzystywane. Tymczasem mogłyby służyć za dane treningowe, pozwalające np. dobrać odpowiednią metaheurystykę do rozwiązywanego problemu w sposób automatyczny, umożliwić stworzenie nadrzędnych metod sterowania procesem przeszukiwania, umożliwić automatyczny dobór hiper-parametrów na podstawie danych o instancjach problemu, czy też stanowić dane konieczne do stworzenia metod wspierających klasyczne metody rozwiązywania problemów optymalizacji poprzez usprawnienie ich elementów. Kandydatem do takich ulepszeń jest np. mechanizm pamięci dla przeszukiwania z zabronieniami, jak przedstawiono w badaniach własnych opisanych w rozdziale 8. Choć badania opisane w przywołanym rozdziale były stosunkowo proste, można je uznać za wystarczające do uzasadnienia możliwości poprawy klasycznych metod optymalizacji. Jednak w przeciwieństwie do przytoczonych badań, te opisane w ramach niniejszego rozdziału wykorzystują metody znacznie bardziej zaawansowane, co wynika z zastosowania uczenia ze wzmocnieniem (RL, od ang. *Reinforcement Learning*). Niemniej należy podkreślić, że wykorzystanie bardziej zaawansowanych metod wymaga poniesienia pewnych nakładów, jak choćby przeprowadzenie skomplikowanej implementacji czy też czasochłonnego procesu zbierania danych uczących. Te koszty nie zawsze są akceptowalne w praktyce, a ich poniesienie wymaga zgody na kompromis.

W szerszym kontekście dysertacji celem badań zaprezentowanych w tym rozdziale jest odpowiedź na następujące pytania:

1. czy można wykorzystać zaawansowane metody oparte o sieci neuronowe w ramach uczenia ze wzmocnieniem do zastąpienia elementów klasycznych algorytmów metaheurystycznych?
2. czy można wykorzystać elementy analizy przestrzeni rozwiązań (np. metryki) aby poprawić wyniki podejścia z poprzedniego pytania?
3. czy podejście takie może być potencjalnie konkurencyjne do sprawdzonych metod (np. symulowane wyżarzanie)?

Pierwsze z pytań bezpośrednio odnosi się do jednej z tez niniejszej rozprawy: *Możliwe jest zastosowanie zaawansowanych metod opartych o sieci neuronowe w ramach uczenia ze wzmocnieniem do zastąpienia elementów klasycznych algorytmów metaheurystycznych.*

Rozdział ten zawiera opis przebiegu niepublikowanych badań własnych nad wykorzystaniem uczenia ze wzmocnieniem w ramach optymalizacji dyskretnej. Ze względu na ich ograniczony zakres, badania te należy uznać za swojego rodzaju podwaliny do dalszych eksperymentów. Mimo takiego stanu rzeczy, zdecydowano się zachować strukturę podobną do pozostałych rozdziałów opisujących badania własne: 6, 7, 8 – jednak ze znacznie rozbudowanym wstępem teoretycznym. Podjęto również świadomą decyzję o wykorzystaniu głównie angielskich, stosunkowo nowych nazw modeli oraz sieci neuronowych, które nie zawsze mają ugruntowane polskie tłumaczenie.

9.1 Powiązane badania

Pomimo zawartego w poprzednich rozdziałach opisu klasycznych metod rozwiązywania problemów optymalizacji dyskretnej oraz opisu sieci neuronowych, w dysertacji nie przedstawiono jeszcze stanu wiedzy łączącego oba te obszary. Niniejszy podrozdział służy dokładnie temu – przy czym zastosowano podział na dwie części: w pierwszej opisano tradycyjne podejścia wykorzystujące klasyczne sieci neuronowe, podczas gdy w drugiej skupiono się na metodach współczesnych, opartych zarówno o uczenie głębokie, jak i o uczenie ze wzmocnieniem.

9.1.1 Klasyczne sieci neuronowe w optymalizacji

Klasyczne modele oraz metody wykorzystujące sieci neuronowe w optymalizacji dyskretnej nie zyskały powszechnego uznania, pozostając jedynie interesującą alternatywą wobec bardziej sprawdzonych dedykowanych metod dokładnych oraz heurystyk. Taki stan rzeczy wynika zarówno ze względu na kryterium zapotrzebowania na moc obliczeniową, jak i, co ważniejsze, z powodu jakości uzyskiwanych przez nie rozwiązań. W związku z tym uznano za stosowne ograniczenie zakresu tej sekcji do przedstawienia jedynie kilku najbardziej znanych modeli z okresu klasycznego uczenia maszynowego do optymalizacji, w tym na podstawie książki [258]. Podejścia klasyczne można podzielić na kilka sposobów z perspektywy genezy zastosowanego modelu: a) klasyczne uczenie sieci neuronowych, wykorzystujące proces treningu na podstawie dostarczonych danych; b) metody inspirowane biologią sieci neuronowych, czyli metody wykorzystujące pewne elementy biologicznego procesu zachodzącego w mózgu, jednak bez „treningu". Alternatywnym podziałem metod i modeli może być również sposób ich wykorzystania: a) bezpośrednia próba otrzymania rozwiązania problemu bazowego za pomocą sieci; b) wykorzystanie sieci neuronowej jako metody pomocniczej, wpierającej inną, sprawdzoną metodę rozwiązywania problemu; c)

wykorzystanie sieci w celu rozwiązania problemów powiązaniach w ramach większego systemu ale nie bezpośrednio do rozwiązywania problemów optymalizacji.

Za przykład modelu sieci neuronowych inspirowanego biologią, ale nie podlegającego klasycznemu procesowi treningu, można przedstawić sieci Hopfielda [114]. Model powstały jako próba odwzorowania procesów związanych z pamięcią oraz przechowywaniem wspomnień. Sieci te mają interesującą, mocno rekurencyjną strukturę. Tym niemniej nie mają typowych wad związanych z jednoczesną rekurencją, połączoną z nieliniową funkcją aktywacji. Stan sieci nie jest niestabilny oraz sieci te nie wpadają w oscylacje czy też zachowania chaotycznie. Jest to spowodowane narzuceniem pewnych warunków:

- choć wszystkie neurony są ze sobą powiązane, to nie ma rekurencji na poziomie pojedynczego neuronu (stan poprzedni neuronu nie wpływa bezpośrednio na stan w następnym kroku sieci – nie ma bezpośredniego połączenia neuronu samego z sobą, choć pewna informacja może być przekazana poprzez połączenia z pozostałych neuronów);
- połączenia pomiędzy każdą parą neuronów są symetryczne;
- każdy z neuronów ma binarną funkcję aktywacji;
- stan całej sieci jest określany za pomocą określonej metody, składającej się z kolejnych kroków, gdzie zmieniany jest stan wyłącznie pojedynczego neuronu (zamiast treningu mamy tutaj do czynienia z procesem zmierzającym do ustalenia finalnego stanu całej sieci).

Narzucenie tych ograniczeń sprawia, że stan aktywacji neuronów w sieci Hopfielda można opisać za pomocą globalnej funkcji energii, przy czym sieć w ramach przyjętej metody aktualizacji zmierza do stanu o najmniejszej energii. Sprawia to, że sieć poprzez wartości aktywacji neuronów jest w stanie „zapamiętać" konfigurację stanów o lokalnie minimalnej wartości funkcji energii, do których „wraca" ze stanów pośrednich. Stąd analogia do pamięci oraz ważna zaleta modelu w kontekście modelowania procesu biologicznego: odporność na uszkodzenia, czy to związane z niepoprawną informacją (stanem początkowym), czy też uszkodzeniem samej sieci. Jak pokazano w pracy [115], sieci Hopfielda można wykorzystać do optymalizacji dyskretnej. W przytoczonej, klasycznej pracy, wykorzystano je do rozwiązania problemu komiwojażera za pomocą m^2 neuronów, których stan odpowiadał wykorzystaniu (jeśli neuron jest aktywny) trasy pomiędzy parą miast. Co ważne, wzór na globalny stan energii musiał być dostosowany do problemu, tak aby zwiększać funkcję energii w wypadku konfiguracji sieci reprezentujących rozwiązania niedopuszczalne.

Pomimo „kar" związanych z rozwiązaniem niedopuszczalnym, sieć może zakończyć swoje działanie właśnie w takim stanie. Aby częściowo złagodzić tą wyraźną wadę, stosuje się wielokrotne uruchamianie modelu z różnych stanów początkowych. Kolejnymi wadami modelu są: brak gwarancji znalezienia globalnego optimum oraz problemy ze skalowaniem wraz z próbą rozwiązywania coraz większych instancji problemu. Pomimo braku spektakularnego sukcesu w ramach optymalizacji dyskretnej, pierwotna koncepcja sieci Hopfielda wraz modyfikacjami nie została zapomniana oraz powraca we współczesnych badaniach w różnych dziedzinach [132, 213, 218].

Kolejnym ważnym modelem są sieci Kohonena, inaczej nazywane sieciami samoorganizującymi się (od ang. *Self-Organizing Maps* [150], SOM). Sieci te należą do kategorii uczenia maszynowego nienadzorowanego. Modele tego typu znalazły swoje powszechne zastosowania w ramach klasteryzacji, redukcji wymiarowości oraz jako metoda pozwalająca wizualizować dane wielowymiarowe, przy jednoczesnym zachowaniu części z relacji pomiędzy nimi. Często wykorzystywaną topologią sieci SOM jest ustawienie neuronów w dwuwymiarową kratownicę. Każdy z neuronów połączony jest z wybranymi sąsiednimi neuronami (w wersji często stosowanej neuron ma sąsiadów: po lewej, po prawej, z góry i z dołu) oraz w ramach połączenia z danymi wejściowymi, połączeniem typu każdy neuron z każdą wartością wektora wejściowego. Wagi początkowe neuronów inicjalizowane są w sposób losowy. Proces uczenia polega na aktywacji sieci z wykorzystaniem danych, po czym obliczeniu który z neuronów jest najbardziej „pobudzony". W tym celu wykorzystuje się korelacje pomiędzy wektorem danych wejściowych z wagami każdego z neuronów. W wersji uczenia WTA (od ang. *Winner Takes All*) jedynie wagi zwycięskiego neuronu są zmieniane, przy czym zmiana upodabnia wagi zwycięzcy do danych wejściowych dla których „wygrał" zadany neuron. Proces ten jest modyfikowany poprzez współczynnik uczenia, od którego zależy stopień zmiany. W wersji uczenia WTM (od ang. *Winner Takes Most*), wagi zwycięskiego neuronu zostają zmodyfikowane wraz z neuronami sąsiadującymi z tym neuronem. Proces jest powtarzany wielokrotnie w ramach iteracji do osiągnięcia warunku stopu. Ustawienie neuronów w ramach płaszczyzny dwuwymiarowej jest zasadne przy redukcji wymiarowości oraz w ramach wizualizacji. Strukturę sieci należy zawsze dostosować do wybranego zastosowania. W ten sposób w ramach badań nad wykorzystaniem modelu SOM do rozwiązywania problemu TSP [42], wykorzystano topologię jednowymiarową neuronów, w ramach której sąsiedztwo składało się wyłącznie z dwóch neuronów. To, w połączeniu z ustaleniem liczby neuronów na równą liczbie miast, wprowadziło analogię pomiędzy strukturą sieci, a trasą w TSP.

Jedną z wad SOM jest wrażliwość na początkową strukturę i liczbę neuronów. Niektóre dane, szczególnie w wypadku mniejszej liczby neuronów w sieci, nie są dobrze reprezentowane. Możliwe jest też występowanie konfliktów w ramach procesu samoorganizacji. Część z przedstawionych wad można jednak wyeliminować poprzez zastosowanie rozwiązań hybrydowych. Stąd, metody hybrydowe są często wykorzystywane [2, 42, 128], w przeciwieństwie do podejścia wcześniejszego, gdzie próbowano wykorzystać jedynie sieci SOM [5]. W ramach przytoczonych wcześniej badań nad rozwiązaniem problemu TSP [42], wykorzystano procedurę naprawy oraz metodę 2-Opt. Służą one rozwiązywaniu konfliktów spowodowanych niedokładnym odwzorowaniem relacji pomiędzy neuronem, a miastem oraz poprawie otrzymanej trasy. Mimo to, w badaniach [42], podejście hybrydowe z sieciami SOM doprowadziło do otrzymania gorszych wyników niż uzyskanych z zastosowaniem algorytmów ewolucyjnych i dostosowanej do problemu heurystyki Lin–Kernighan. Choć klasyczny model sieci SOM jest stosunkowo prosty, to istnieją współczesne modyfikacje tego rodzaju sieci [213], które pozwalają na uzyskanie bardzo dobrych rezultatów, jednak poza dziedziną optymalizacji dyskretnej. Przytoczone obiecujące wyniki uzyskano w ramach badań nad zachowaniami immunologicznymi [303].

Za przykład alternatywnego wykorzystania sieci w ramach większego systemu może posłużyć książka R. Knosala [147], gdzie do optymalizacji procesów inżynierii produkcji wykorzystano sieci neuronowe oraz metody sztucznej inteligencji. Metody te zastosowano w zadaniach, dla których są one wyjątkowo skuteczne, czyli do rozpoznawania obrazów oraz analizy językowej – a nie bezpośrednio do rozwiązywania problemów optymalizacji dyskretnej. Dla przykładu: ze względu na ogromną różnorodność wytwarzanych elementów o różnym stopniu skomplikowania w inżynierii produkcji, konieczne jest wsparcie konstruktorów w pozyskaniu informacji o już zaprojektowanych elementach, aby umożliwić wykorzystanie istniejącej już wiedzy. Do badań nad kojarzeniem podobieństw elementów produkowanych (technologii grupowej) wykorzystano modele i metody takie jak sieci Kohonena [207], sieci typu ART (od ang. *Adaptive Resonance Theory*, [272]) i ich odmiany, takie jak np. Fuzzy ART [120] czy sieci ARTMAP [86]. Podejście takie pozwoliło:

- wyeliminować sytuacje gdzie nowy element jest identyczny z już produkowanym, przez co ponoszenie ponownych kosztów projektowych jest niezasadne;
- nowy element jest podobny do już produkowanego, przez co należy wykorzystać wcześniejszą dokumentację aby dostosować proces produkcji na podstawie zgromadzonych doświadczeń;

- jeśli nowy element nie jest podobny do żadnego z wcześniej produkowanych, należy wykonań pełną dokumentację konstrukcyjną.

Tym samym wykazano możliwość usprawnienia większego systemu produkcji, poprzez rozwiązanie problemów kojarzonych z metodami sztucznej inteligencji: klasyfikacji, rozpoznawania obrazów czy grupowania. Pozwoliło to, za [147], na zwiększenie wydajności systemu produkcji m.in. poprzez: a) obniżenie liczby produkowanych elementów, b) zmniejszenie nakładów na konstruowanie, planowanie i wytwarzanie, c) wielokrotne wykorzystanie zgromadzonej wiedzy, d) obniżenie czasów przygotowawczo-zakończeniowych.

Za podsumowanie podejścia klasycznego może służyć cytat z książki C. Smutnickiego [258] (szczególnie aktualne na czas wydania, czyli rok 2012): *Chociaż sieci neuronowe* [...] *są dziedziną intensywnie rozwijaną, ich zastosowanie do rozwiązywania problemów dyskretnych wydaje się wciąż dalekie od oczekiwanego*. Dodatkowo w źródle tym zwrócono uwagę na brak konkurencyjności metod opartych o sieci neuronowe w stosunku do metod opartych o przeszukiwanie lokalne. Jako ilustracyjny przykład przytoczono jakość wyników otrzymanych dzięki przeszukiwaniu z zabronieniami (wykorzystującym właściwości blokowe) do rozwiązywania problemów przepływowych. Z uwagi na wspomniany brak konkurencyjności, ograniczono część przeglądu związaną z klasycznym podejściem wykorzystującym sieci neuronowe na rzecz metod współczesnych.

9.1.2 Nowoczesne metody: głębokie uczenie oraz uczenie ze wzmocnieniem w optymalizacji

Sukcesy w innych dziedzinach (głównie przetwarzanie języka naturalnego oraz obrazu) doprowadziły do prawdziwej „eksplozji" prac poświęconych zastosowaniu uczenia głębokiego. Ciągły postęp, wręcz „zalew" nowych modeli sieci neuronowych prowadzi do innowacyjnych prób przeniesienia skutecznych rozwiązań – z nadzieją na powtórzenie ich sukcesów – do kolejnych dziedzin nauki oraz inżynierii[1].

Za interesujący przykład mogą posłużyć modele rekurencyjne sieci neuronowych (ang. *Recurrent Neural Network*, RNN [229]), dedykowane se-

[1]Doprowadził też do „eksplozji" publikacji naukowych w ramach uczenia maszynowego [211]. Bardzo szybko powstają kolejne, chwilowo „najlepsze" modele poznaczone do pokonywania określonych zbiorów testowych. Następnie modele te można przenieść do kolejnych dziedzin. Na przykład model YOLO5[129], przeznaczony do detekcji obiektów ze zdjęć jak w konkursie ILSVRC, z powodzeniem może zostać dostawany zarówno do detekcji znacznie mniejszych obiektów jak budynki na zdjęciach satelitarnych, jak i stanowić istotny elementy systemu śledzenia ruchu pieszych w czasie rzeczywistym na transmitowanym obrazie z kamery... by z czasem być zastąpiony kolejnym „chwilowo najnowszym" modelem detekcji.

kwencyjnemu podejściu do przetwarzania danych, są one szczególnie przydatne w analizie szeregów czasowych. Początkowe modele rekurencyjne cechowały się takimi problemami jak: zanikający ale również wybuchający gradient czy brak możliwości „utrzymania" informacji długoterminowej. Aby przeciwdziałać części z tych wad stworzony został model *Long Short-Term Memory*, LSTM [112], którego nazwa podkreśla zachowanie możliwość zachowania zarówno informacji długo jak i krótkoterminowej. Model ten z powodzeniem został wykorzystany w problemach takich jak: rozpoznawanie pisma [97], mowy [236], przetwarzanie tekstu [295] czy tłumaczenie maszynowe [7, 265]. W ramach prób poprawy wyników dla tych zadań, wprowadzono mechanizm atencji [7, 98, 301]. Ma on za zadanie zachować informację o małych – ale istotnych – częściach informacji, co w ramach przetwarzania języka naturalnego pozwala na zachowanie informacji o kontekście przetwarzanego tekstu.

Powyżej przywołane badania obrazują stopniowy rozwój sieci neuronowych. Obiecujące zastosowania napędzają kolejne badania skierowane na poprawę jakości otrzymanych wyników poprzez eliminację słabych punktów wcześniejszych rozwiązań. Co ważne, te modele i mechanizmy – RNN, LSTM, atencja – nie zostały przywołane jedynie aby zilustrować postęp badań nad sieciami neuronowymi. Ich połączanie zostało wykorzystane z powodzeniem w ramach pracy [294] w modelu *Pointer Network*, PN, do rozwiązania wybranych problemów optymalizacji. W tym do rozwiązywania: problemu otoczki wypukłej (ang. *Convex Hull*), obliczenia triangulacji Delaunay (ang. *Delaunay Traingulations*) oraz symetrycznego problemu komiwojażera. Co interesujące, praca ta nie miała na celu stworzenia najlepszej możliwej metody rozwiązywania wymienionych problemów optymalizacji. Autorzy zamiast tego przedstawili własny sposób rozwiązania istotnej wady sieci neuronowych – stałego rozmiaru „słownika" danych wyjściowych[2]. Problemy optymalizacji zostały więc wybrane ze względu na zmienny rozmiar oczekiwanych rozwiązań. W wypadku otoczki wypukłej, liczba punktów ją stanowiąca może być różna pomiędzy rozwiązaniami w ramach tej samej instancji problemu. Z kolei dla problemu komiwojażera długość rozwiązania (trasy) zależy od liczby miast w instancji. Sam model

[2]Co można zilustrować na przykładzie klasycznych, prostych i jednokierunkowych sieci neuronowych, gdzie wada ta jest jeszcze bardziej dotkliwa. Sieć jest trenowana z wcześniej określonym, stałym rozmiarem wejścia oraz wyjścia. Dla problemu TSP odpowiadałoby to np. sieci neuronowej która jako wejście przyjmuje wyłącznie macierz odległości o zadanym rozmiarze aby na wyjściu zwrócić kolejność odwiedzonych miast. Sieć klasyczna wytrenowana w ten sposób byłaby przydatna wyłącznie dla instancji o zadanej liczbie miast i nie mogłaby zostać wykorzystana dla innej liczby miast bez zmiany struktury oraz ponownej procedury treningu.

PT, jako sieć rekurencyjna, konstruuje rozwiązane na przestrzeni kolejnych iteracji – przetwarzające dane w sposób sekwencyjny, charakterystyczny dla modelowania danych jako serii czasowej. Model składa się z dwóch sieci: części kodującej oraz dekodującej (*encoder* oraz *decoder*), wykorzystując od 256 do 512 jednostek LSTM, trenowanych z wykorzystaniem SDG na 1 milionie przykładów treningowych o rozmiarze instancji od 5 do 50 miast. Wyniki modelu okazały się lepsze od implementacji prostej heurystyki[3] ale nie pokonały 2 pozostałych heurystyk[4], co nie jest dobrym wynikiem przy tak małej liczbie miast. Należy przy tym zaznaczyć, że metody do których porównano wyniki modelu nie stanowiły najlepszych metod rozwiązywania TSP (w znaczeniu *state-of-the-art*), prawdopodobnie zostały wybrane jedynie ze względu na ich łatwą dostępność (w tym otwarty kod źródłowy).

Pomimo tego, podejście wykorzystujące model *Pointer Network*, PT, okazało się na tyle obiecujące i nowatorskie, że doczekało się licznych prac próbujących ulepszyć oryginalny model. Tym samym badania nad modelami PT można zaliczyć do stanowiących początek rozwoju „nowoczesnego" podejścia wykorzystującego sieci neuronowe w optymalizacji, szczególnie w ramach podejścia opartego o próby bezpośredniego rozwiązywania problemów optymalizacji dyskretnej. W pracy [16] zdecydowano się połączyć model pointer network z uczeniem ze wzmocnieniem. Wykorzystano do tego jeden z podstawowy algorytm RL – *REINFORCE* [304]. W badaniach tych oprócz problemu TSP pokazano możliwość wykorzystania modelu sieci również do rozwiązywania problemu plecakowego. W badaniach [194] zwrócono uwagę, że w pierwotnym modelu PN kolejność wprowadzania danych wpływa na otrzymane rozwiązanie. Jest to zjawisko pożądane w wypadku modeli do przetwarzania języka naturalnego ale na pewno nie jest korzystne i nie powinno mieć znaczenia w wypadku problemów optymalizacji takich jak TSP. Autorzy pracy swoje podejście wykorzystali do rozwiązania problemu VRP. Praca [141] wykorzystała inny rodzaj kodowania informacji wejściowej dla sieci neuronowej. W tym celu wykorzystano graf jako reprezentację informacji o instancji. Na wejście podawano również rozwiązanie niekompletne w postaci części trasy (dla problemu TSP, rozważano również inne problemy). Sam proces uczenia sieci wykorzystywał uczenie ze wzmocnieniem. Podkreślono przy tym wykorzystanie zebranych informacji w ramach bufora pamięci (ang. *memory replay* [187]) – czyli jednego z dość

[3] Suboptimal travelling salesman problem (tsp) solver – https://github.com/dmishin/tsp-solver – implementacja dostępna 01.08.2022.

[4] *Traveling salesman problem C++ implementation* – https://github.com/samlbest/traveling-salesman oraz *C++ implementation of traveling salesman problem using christofides and 2-Opt* – https://github.com/beckysag/traveling-salesman; obie implementacje dostępne 01.08.2022.

powszechnie wykorzystywanych ulepszeń procesu uczenia w RL. W badaniach [197] wykorzystano możliwość transformacji instancji problemu TSP do problemu QAP. Użyto przy tym model sieci neuronowej wykorzystujący graf (ang. *Graph Neural Network* [239, 313], GNN). Sieć była uczona w sposób nadzorowany, rozwiązanie było uzyskane z pomocą przeszukiwania snopowego (ang. *Beam Search* [174]).

W pracy [133] rozważano wariant TSP z wieloma komiwojażerami. Sieć uczono w sposób nadzorowany, oczekując na wyjściu rozwiązań częściowych, które następnie były przekształcanie w rozwiązanie końcowe za pomocą przeszukiwania snopowego. Autorzy podkreślają, że rozwiązywanie wersji problemów dyskretnych z dodatkowymi ograniczeniami może być właściwą niszą dla podejścia neuronowego.

W pracy [75] wykorzystano nowy rodzaj sieci neuronowej *transformer* [290], w przeciwieństwie do wcześniej wykorzystywanego modelu LSTM. Decyzja miała za zadanie poprawić proces kodowania informacji podawanych jako dane wejściowe do sieci o położeniu miast (poprawa zarówno *encodera* oraz *decodera*). Model *transformer* rozwija w znaczący sposób wspomniany model *atencji* i w ramach badań nad przetwarzaniem języka naturalnego wypiera wcześniej powszechnie stosowany model LSTM. Uczenie odbywało się z wykorzystaniem uczenia ze wzmocnieniem, a dodatkowo w ramach poprawy rozwiązania wykorzystano standardową heurystykę 2-Opt. Autorzy pracy podkreślili, że ich zdaniem wyniki zachęcają do połączenia podejścia opartego o wykorzystanie sieci neuronowych wraz z wykorzystywanymi w optymalizacji heurystykami aby osiągać lepsze wyniki niż te możliwe w wypadku zastosowania jedynie jednej z tych metod.

Również praca [152] bazuje zarówno na modelu *transformer*, jak i mechanizmach analogicznych do oryginalnej pracy *Pointer Network*. Otrzymane wyniki były bliskie globalnemu optimum (ale go nie osiągnęły) dla instancji do 100 miast. Oprócz problemu TSP zbadano możliwość rozwiązywania dwóch wariantów problemu VRP (SDVRP oraz CVRP) oraz problemów *Orienteering Problem* i *Stochastic Prize Collecting TSP*. W *Orienteering Problem*, każdy węzeł posiada nagrodę, a celem jest wyznaczenie trasy która jest krótsza od narzuconego limitu przy jednoczesnej maksymalizacji sumarycznej nagrody z odwiedzonych węzłów. W *Stochastic Prize Collecting TSP* celem jest zebranie minimalnej sumarycznej nagrody przy jednoczesnej minimalizacji pokonanego dystansu oraz uwzględnieniem zapłaconych kar. Każdy z węzłów ma zarówno nagrodę za odwiedzenie ale również karę za nie odwiedzenie. W wersji stochastycznej, przewidywana wartość nagrody jest znana przed jego odwiedzeniem ale realna wartość jest poznana dopiero po odwiedzeniu. Autorzy podkreślają, że dodatkowe

ograniczenia w rozważanych problemach sprawiają iż nie zawsze istnieje dla nich dedykowana heurystyka. Ten fakt zwiększa praktyczną możliwość zastosowania podejścia neuronowego, które może nauczyć się strategi rozwiązywania zmodyfikowanej wersji problemu.

W pracy [131] ponownie wykorzystano kodowanie informacji wejściowej sieci neuronowej wykorzystujące graf oraz uczenie nadzorowane. Zbiór uczący zawierał aż 1 milion rozwiązań dla przykładowych instancji składających się z 20, 50 lub 100 miast, natomiast zbiór walidacyjny posiadał po 10 tysięcy instancji. Model zwracał prawdopodobieństwo wykorzystania określonych połączeń pomiędzy miastami, przez co aby uzyskać finalne rozwiązanie konieczne było wybranie połączeń. Wykorzystano do tego celu przeszukiwanie zachłanne oraz przeszukiwanie snopowe. Uzyskano rozwiązania gorsze od optimum o 0,01% dla 50 miast oraz 1,39% dla 100 miast w wypadku zastosowania przeszukiwania snopowego oraz 3,10% oraz 8,38% w wypadku przeszukiwania zachłannego.

Ciekawe podejście zostało zaprezentowane w pracy [305], gdzie wykorzystano proces przeszukiwania lokalnego inspirowanego algorytmem 2-Opt. Sieć neuronowa, bazująca na architekturze zbliżonej do modelu *transformer*, służy w nim do wyboru które miasta w ramach transformacji obecnego rozwiązania zostaną wykorzystane do zastosowania ruchu typu „odwróć". Perturbacja ta jest zawsze akceptowana, co oznacza, że proces przeszukiwania nie kończy się w lokalnych minimach. Dla danych losowych udało się uzyskać rozwiązania gorsze od optimum o 1,42% dla problemu TSP oraz 2,47% dla instancji problemu CVRP – przy rozwiązywaniu instancji składających się ze 100 miast (rozwiązano również instancje składające się z 20 oraz 50 miast).

W pracy [307] wykorzystano połączenie sieci neuronowych oraz metody *Monte Carlo Tree Search*, MCTS [44]. MCTS jest heurystyką łączącą zalety przeszukiwania struktury drzewa z zaletami przeszukiwania losowego. Polega ona na podejmowaniu decyzji za pomocą losowego próbkowania przestrzeni, jednocześnie na tej podstawie budując drzewo stanów. Metoda ta z powodzeniem została wykorzystana do problemów które można przedstawić w postaci sekwencji decyzji, np. do rozwiązywania gier planszowych, takich jak: szachy, warcaby czy Go. Połączenie MCTS z sieciami neuronowymi, w postaci modelu AlphaGO [249], pozwoliło osiągnąć rewelacyjny wynik – pokonać ludzkiego przeciwnika o randze arcymistrza w grze Go[5].

[5]To historyczne wydarzenie można zobaczyć w serwisie YouTube: *Google DeepMind Challenge Match: Lee Sedol vs AlphaGo* https://youtu.be/vFr3K2DORc8 lub przeczytać o nim w źródle https://www.bbc.com/news/technology-35785875; materiały dostępne 01.09.2022. Interesujący jest też zupełnie inny odbiór tego wydarzenia pomiędzy kulturami Zachodu, a Wschodu Azji – szerokie omówienie „chińskiego efektu Sputnika" można

W pracy [307] rozwiązywano problem TSP, z wykorzystaniem modelu sieci neuronowej wykorzystujący graf oraz MCTS. Sieć służyła jako źródło informacji o połączeniach pomiędzy miastami i zapewniała oszacowanie prawdopodobieństwa wybrania konkretnego połączenia w ramach procesu decyzji o wyborze następnie odwiedzanego miasta. W badaniach eksperymentalnych otrzymano rozwiązania gorsze od optymalnych o 0,01%, 0,20% i 1,04% odpowiednio dla losowych instancji o rozmiarach 20, 50 i 100. Dodatkowo przedstawiono wyniki na instancjach losowych, wygenerowanych w sposób tworzący skupiska (z nieco gorszymi rezultatami). Przedstawiono również granice praktyczności wykorzystania tego podejścia do 100 miast. Na losowych instancjach o rozmiarach 200, 300 oraz 500 uzyskano znacząco względnie gorsze wyniki 1,91%, 2,99% i 4,37%. Dodatkowo model uczony na instancjach o rozmiarze 500 pozwolił na uzyskanie wyników gorszych od optimum o 3,33% i 4,48% dla rozmiarów 500 i 1000. Autorzy przedstawili średni czas uzyskania wyników w ramach wykorzystanej metody: 3,58 sekund dla $m = 50$, 27,62 sekund dla $m = 100$, 3,9 minut dla $m = 200$, 21 minut dla $m = 300$, 42 minut dla $m = 500$ oraz 292 minut dla $m = 1000$ na sprzęcie przeznaczonym do zastosowań profesjonalnych, w ramach klastra obliczeniowego (m.in. 2 × Xeon(R) E5-2620 oraz 8 × Nvidia2080Ti).

Współczesne podejścia wykorzystujące sieci neuronowe do rozwiązywania problemów optymalizacji nie opierają się wyłącznie na modyfikacji modelu PT. Istnieje szerokie pole możliwości zastosowania wielu metod uczenia maszynowego [19, 101, 137]. Spośród wielu alternatyw postanowiono bliżej przedstawić algorytmy oparte o uczenie ze wzmocnieniem (RL). Wybór ten można argumentować zarówno poprzez liczne badania naukowe wskazujące pewien trend (przytoczone dalej), jak i przykłady sukcesów wykorzystania RL. Do tych sukcesów z pewnością należy pokonywanie „ludzkich mistrzów" coraz bardziej skomplikowanych gier (czy zgodnie z nomenklaturą RL – środowisk), w tym: szachów [251], wspomnianej już gry Go [249, 252] oraz wybranych współczesnych gier komputerowych [293]. Szczególnie ostatni przykład łączy się zarówno ze wzrostem przestrzeni stanów, jak i obejmuje konieczności podejmowania decyzji w środowisku o niepełnej informacji w czasie rzeczywistym. Tym samym powstaje swojego rodzaju oczekiwanie co do możliwości zastosowania podejścia opartego o RL do rozwiązywania problemów optymalizacji. Szczególnie do rozwiązywania dużych instancji, tam gdzie wykorzystanie algorytmów dokładnych nie jest możliwe. Co ważne, w wielu badaniach wykorzystuje się RL jako metodę pomocniczą, wspierającą wcześniej znany i sprawdzony algorytm, a nie jako narzędzie do bezpośredniej „konstrukcji" rozwiązywania problemu (co

znaleźć w źródle popularnonaukowym [162].

może stanowić pewien kontrast do prac wywodzących się z modelu PT[6]). Za przykład „trendu" wykorzystania RL w optymalizacji można przytoczyć badania i prace naukowe takie jak:

- Przeszukiwanie z zabronieniami, nadzorowane przez RL, dla problemu kolorowania, gdzie komponent RL kieruje przeszukiwanie w kierunku obiecujących regionów [264].
- Badania nad wykorzystaniem rekurencyjnej sieci neuronowej w połączeniu z metodami RL (2 podejścia oparte o trening wstępny oraz aktywne przeszukiwanie, oba bazujące na *Policy Gradients*) w celu rozwiązania problemu TSP (do 100 miast), podejście z powodzeniem zaadaptowane dla problemu plecakowego [16] (do 200 przedmiotów).
- Badania [246] nad wykorzystaniem podejścia hybrydowego, łączącego wykorzystanie metody RL (Q-Learning) oraz kilku, zmodyfikowanych metaheurystyk inspirowanych naturą (dokładnie zachowaniem szarych wilków – *The Grey Wolf Optimizer* oraz zachowaniem wielorybów – *The Whale Optimization Algorithm*) w celu optymalizacji funkcji ze zbiorów CEC2014 i CEC2015 [169] (*Special Session and Competition on Single Objective Bound Constrained Numerical Optimization*) – przykładowa funkcja F11 Griewank: $\frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1$.
- Wykorzystanie teorii gier (równowagi Nasha) do modyfikacji Q-Learningu (jednej z bazowych metod RL), w celu rozwiązania rozproszonego permutacyjnego problemu przepływowego [223].
- W pracy [40] przedstawiono nowe podejście do rozwiązywania TSP za pomocą RL i sztucznych sieci neuronowych. Metoda wykorzystuje model transformer oraz trening z wykorzystaniem RL, a więc nie wymaga wcześniejszej wiedzy o optimum instancji, w celu stworzenia zbioru treningowego. Na uwagę zasługuje szczególnie wysoka jakość uzyskiwanych rezultatów. Autorzy raportują względny błąd rzędu 0,004% dla TSP50 oraz 0,39% dla TSP100.

[6]Część z wcześniej przytoczonych prac-następców modelu PT wykorzystuje RL. Nie są to podejścia rozłączne. Co więcej, ich połączenie wydaje się praktycznym podejściem, ponieważ tak uzyskany algorytm uczy się poprawnego sposobu przeszukiwania przestrzeni rozwiązań bazując na eksploracji. Może to następnie doprowadzić do uzyskania ciekawych sposobów wykorzystania zebranej wiedzy, np. w postaci nietypowego sposobu przeszukiwania przestrzeni rozwiązań. Dodatkowo, w przeciwieństwie do podejścia opartego o uczenie nadzorowane, nie jest konieczne posiadanie zbioru danych w postaci par wejście-oczekiwane wyjście. Konstrukcja takiego zbioru sugeruje, że posiadamy już algorytm który te oczekiwane wyjście zapewni, np. Concorde dla problemu TSP, co w wielu przypadkach podważa zasadność opracowywania nowego algorytmu. Co więcej, uczenie nadzorowane nie pozwoli zwykle uzyskać wyników lepszych niż dostarczone w ramach wykorzystanego zbioru treningowego.

Pogłębiony przegląd literatury o zastosowaniu uczenia ze wzmocnieniem do rozwiązywania problemów dyskretnych można znaleźć w [183]. Podsumowanie zawartego w nim stanu wiedzy na rok 2021 wydaje się optymistyczne. W źródle tym zwrócono uwagę na możliwość generalizacji pomiędzy problemami – raz stworzona metoda z powodzeniem może zostać wykorzystana w ramach różnych problemów optymalizacji. Otrzymana jakość rozwiązań zbliża się do tych z wyznaczonych przez popularne heurystyki czy generalne metody dokładne (np. oprogramowanie do rozwiązywania MILP). Jednak zaznaczyć trzeba, że bardziej skomplikowane rodzaje problemów oraz bardzo duże instancje pozostają wyzwaniem dla podejścia opartego o sieci neuronowe.

Podsumowując część przeglądową, podejście wykorzystujące nowoczesne sieci neuronowe (w tym uczenie głębokie oraz uczenie ze wzmocnieniem) – pomimo wyraźnie większego progu wejścia – wydaje się obiecujące i przyszłościowe. Wymaga jednak szerokiej wiedzy teoretycznej z obu dziedzin – optymalizacji dyskretnej i uczenia maszynowego, połączonej z umiejętnościami programistycznymi do poprawnej implementacji zaawansowanych metod. Wraz z nieuniknionymi postępami wykorzystania sieci neuronowych w innych dziedzinach, nowe modele i metody neuronowe z dość dużym prawdopodobieństwem będą testowane w nowych obszarach, w tym do rozwiązywania problemów optymalizacji dyskretnej. Choć zapewne wiele z problemów optymalizacji dyskretnej, szczególnie tych dobrze zbadanych, pozostanie zdominowane przez klasyczne metody rozwiązywania; to podejście oparte o sieci neuronowe przestaje oferować „wyniki dalekie od oczekiwanych" – co w roku 2012, za [258], było jeszcze trafnym posumowaniem.

9.2 Eksperymenty obliczeniowe

W celu odpowiedzenia na pytania badawcze zawarte w wstępie tego rozdziału konieczne stało się przeprowadzenie eksperymentów obliczeniowych w oparciu o przytoczoną literaturę oraz wnioski z badań własnych opisanych w poprzednich rozdziałach. Choć uczenie ze wzmocnieniem zostało wybrane jako potencjalnie obiecująca metoda wykorzystania sieci neuronowych, to oczekiwano pewnych trudności w ramach implementacji metod RL do rozwiązywania problemów optymalizacji. Spowodowane było to niedogodnościami RL opisanymi zarówno we wcześniej przytoczonej literaturze, jak i nierecenzowanych raportach [125]. Tym samym spodziewano się wyzwań takich jak:

- Uczenie ze wzmocnieniem wykorzystujące sieci neuronowe wymaga ogromnej liczby próbek danych (co jest związane z niską istotnością

pojedynczej próbki, ang. *Sample Inefficient*) – np. zbiór treningowy złożony z ponad 18 milionów próbek nie jest niczym nietypowym w ramach badań RL, przykład z pracy [109].

- Choć RL może działać w wielu, zróżnicowanych środowiskach to metody „szyte" na miarę, wykorzystujące właściwości problemu bazowego najczęściej działają szybciej i osiągają lepsze rezultaty.
- Badania RL wymagają określenia funkcji nagrody – choć w ramach badań porównawczych, często przyjmuje się funkcję nagrody jako „odgórnie" narzuconą (np. dla zestawów środowisk takich jak `Atari`[7] czy `MuJoCo`[8]) – jednak dobór odpowiedniej nagrody w ramach tworzenia własnego, nowego środowiska „od zera" jest zadaniem trudnym. Złe dobranie funkcji nagrody może prowadzić do niewłaściwego zachowania agenta, który może nie być w stanie nawet „rozwiązać" środowiska czy też po prostu prowadzić do szybszego przeuczenia sieci neuronowej. Dodatkowo, w definicji nagrody przekazywana jest informacja o środowisku, co być może niepożądane jeśli celem badań jest stworzenie metody o bardziej ogólnym przeznaczeniu – w wypadku przeciwnym (celem badań jest osiągniecie określonego wyniku) może istnieć pokusa dostarczenia tak dużej ilości informacji poprzez funkcję nagrody aż wybrana metoda RL w końcu „zacznie działać".
- Agent który uzyskuje systematycznie dobry wynik w ramach określonego zachowania, może uzyskiwać ten wynik poprzez bardzo nietypowe (lub nieporządane) zachowanie w obrębie zdefiniowanego środowiska. Takie, które badacz posiadający dostęp do rozbudowanej wiedzy a priori lub rzeczywistego problemu, nie wziąłby pod uwagę albo co najmniej nie uznał za poprawne. Obrazowym przykładem może być „znalezienie" i wykorzystanie błędu w symulatorze fizyki środowiska w ramach którego agent podejmuje decyzje w pracy [9][9].
- Istnieją problemy z generalizacją pomiędzy środowiskami – choć za standard w uczeniu nadzorowanym uważa się wykorzystanie zbiorów: treningowego, walidacyjnego oraz testowego – w ramach niektórych badań nad RL nie zawsze przyjmuje się tak ostry podział danych.
- Proces treningu w ramach RL jest znacznie mniej stabilny od swojego odpowiednika w ramach uczenia nadzorowanego – zarówno problem z przeuczeniem sieci neuronowej jest bardziej dotkliwy jak i dobór hi-

[7] W ramach biblioteki `Gym`: https://www.gymlibrary.dev/environments/atari/ dostęp 13.09.2022

[8] W ramach biblioteki `Gym`: https://www.gymlibrary.dev/environments/mujoco/ dostęp 13.09.2022

[9] Zachowanie takie można zobaczyć w ramach filmiku https://youtu.be/Lu56xVlZ40M?t=153 dostęp 13.09.2022.

perparametrów ma większy wpływ na „efekt końcowy". Co, w połączeniu z zapotrzebowaniem metod RL wykorzystujących sieci neuronowe na większą liczbę próbek w ramach treningu, utrudnia i wydłuża proces prowadzenia badań.

Świadomość wymienionych trudności doprowadziła do decyzji o stopniowym tworzeniu rozwiązania. Podejście takie miało zapewnić możliwość bezpośredniego wskazania na problematyczne obszary poprzez uzyskane częściowych rezultatów oraz zapewnić elastyczność na przestrzeni kolejnych faz powstawania coraz bardziej skomplikowanego systemu. Należy jednak podkreślić, że istnienie wymienionych trudności nie jest równoznaczne z budowaniem narracji negatywnej, stanowiącej argument za rezygnacją z zastosowania metod RL. Istnieją badania jednoznacznie wskazujące na możliwość uzyskania bardzo dobrych wyników z zastosowaniem RL [21, 109, 293].

Za kluczowe przyjęto dwa założenia: wysoką jakość kodu i wykorzystanie otwartych bibliotek uczenia maszynowego. Jakość implementacji musiała gwarantować efektywność wykorzystania zasobów sprzętowych – potrzeba ta wynikała z konieczności pozyskania możliwe dużego zbioru danych w skończonym czasie, które to dane następnie miał posłużyć do trenowania agenta w ramach metod RL. Oczekiwano przy tym typowego zachowania w ramach uczenia głębokiego: im więcej danych, tym lepsze wyniki trenowanego modelu. Ważną rolą była też możliwość prototypownia różnych rozwiązań niezależnie od siebie oraz wyeliminowanie sytuacji, gdzie podejście jest odrzucane bez testów z powodu braku wystarczających zasobów obliczeniowych. Postanowiono wykorzystywać otwarte biblioteki do uczenia maszynowego zamiast implementować rozwiązanie samodzielnie od podstaw. Decyzja ta wynika z cechy charakterystycznej sieci neuronowych – jeśli istnieje błąd „niekrytyczny" w implementacji sieci neuronowej, to możliwe jest uzyskanie wystarczająco dobrych rezultatów aby autor kodu nie zauważył pomyłki, choć jakość wyników jest gorsza od możliwej do uzyskania w wypadku implementacji poprawnej. Pomniejsze błędy tego typu mogą się oczywiście kumulować, aż w końcu znacząco wpływają na osiągnięte rezultaty modelu. Wykorzystanie otwartego oprogramowania – poprzez kod poddany ocenie i poprawie społeczności – miał ograniczyć obszar popełnianych błędów do kodu związanego z elementami kluczowymi dla badań własnych. Były nimi m.in. wybór i dostosowanie bazowego algorytmu optymalizacji, dobór algorytmu RL, modelowanie funkcji nagrody, ustalaniem przestrzeni stanów w ramach obserwacji, wybór przestrzeni akcji jakim dysponuje agent, czy też dobór hiperparametrów.

Za środowisko przyjęto połączanie tradycyjnego algorytmu optymalizacji dyskretnej oraz instancji rozwiązywanego problemu. Tym samym po-

jedynczy epizod reprezentuje określoną liczbę iteracji algorytmu optymalizacji, wykorzystanego do rozwiązania pojedynczej instancji bazowego problemu optymalizacji. Obserwacja składa się z informacji, które wybrano jako istotne dla działania algorytmu optymalizacji. Akcje, które podejmuje agent wpływają na sposób działania algorytmu optymalizacji. Sygnał nagrody otrzymywany przez agenta połączony jest z funkcją celu w ramach rozwiązywanego problemu optymalizacji. Agent jest metodą RL wykorzystującą sieć neuronową.

9.2.1 Faza 1 – Test narzędzi w języku `Julia`

Ze względu na przyjęte założenia, eksperymenty w tej fazie skupiały się na próbie implementacji wstępnego rozwiązania w ramach języka programowania `Julia`. Co do zalet takiego podejścia, wynikających z mocnych stron tego języka programowania, oczekiwano:

a) bardzo szybkiego wykonywania kodu kompilowanego do kodu maszynowego (niewiele wolniejsze w stosunku do „C++"),
b) znacznie lepszej możliwości zrównoleglenia kodu, dzięki metodom stanowiącym bezpośrednio część składową standardowych bibliotek języka (`Julia` dysponuje odpowiednikiem interfejsu `OpenMP`),
c) istnienia otwartych bibliotek, odpowiedników z języka `Python` do uczenia maszynowego oraz przetwarzania danych, w tym: biblioteka odpowiedzialna za uczenie maszynowe `Flux`[10] oraz biblioteka dedykowana uczeniu ze wzmocnieniem `ReinforcementLearning.jl`[11].

Celem pierwszej fazy budowania systemu była walidacja wstępnie wybranych metod i narzędzi. Stąd, zamiast od razu konstruować docelowy system rozwiązujący problemy optymalizacji dyskretnej, zdecydowano się na stworzenie skrajnie uproszczonego środowiska.

W tym celu utworzono kod własnego środowiska symulującego zmianę temperatury w ramach metaheurystyki symulowanego wyżarzania. Agent podejmuje decyzje o zmianie temperatury w następnej iteracji (przykładowe wartości zmian w ramach akcji agenta: 0,5, 0,2, 0,1, 0,05, 0,02, 0,01, 0,005, 0,002, 0,001, 0). Nagroda jest akumulowana, w sposób niejawny i jest otrzymywana przez agenta na koniec epizodu (po wykonaniu zadanej liczby iteracji). Pożądane zachowanie agenta to stopniowe, liniowe obniżanie temperatury zgodnie ze schematem schładzania

$$T_i^d = 1 - \frac{i+1}{max\_i}, \tag{9.1}$$

[10] https://fluxml.ai/Flux.jl/stable/ dostęp 13.08.2022
[11] https://juliareinforcementlearning.org/ dostęp 13.08.2022

gdzie i jest numerem iteracji, a T_i^d oczekiwaną temperaturą w iteracji i. Akumulowana nagroda jest obliczana na podstawie wzoru

$$hr_i = -\sum_{j=1}^{i}(|T_i^d - T_i|), \tag{9.2}$$

gdzie hr_i oznacza akumulowaną wartość nagrody w iteracji i, a T_i oznacza temperaturę w tej iteracji. Wartości początkowe środowiska to $T_i = 1$, $hr = 0$, $i = 1$, $max\_i = 50$. Stan środowiska obejmował znormalizowaną informację o numerze iteracji oraz o obecnej temperaturze. Przeprowadzone eksperymenty wykorzystywały jedną z prostszych metod RL – tabelę Q (ang. *Q-Table*). Jest to tabela pozwalająca na przypisanie spodziewanej nagrody do akcji w ramach określonego stanu środowisk. Pomimo licznych eksperymentów oraz zastosowania między innymi różnych sposobów kategoryzacji wartości opisanych w ramach stanów obserwacji, nie udało się uzyskać odwzorowania oczekiwanej krzywej schładzania temperatury. Otrzymane krzywe nie odzwierciedlały żądnego znanego z literatury schematu schładzania, tym samym nie osiągnięto odtworzenia docelowego, liniowego schematu. Dodatkowo, nawet w wypadku występowania wzorców w zachowaniu agenta na przestrzeni kilku iteracji, to w wypadku ponownego treningu uzyskiwano kompletnie inny, również niepoprawny schemat. Obserwowane zachowanie agenta można określić więc jako losowe, a fazę pierwszą określić mianem porażki.

Niestety uzyskane rezultaty wskazały albo na błąd w implementacji albo na brak pełnego zrozumienia wykorzystanych bibliotek. W wypadku tej drugiej możliwości, po części winna może być dokumentacja wykorzystanych bibliotek, która zawiera nieścisłości[12] – jest to typowa wada dla niszowych projektów otwartego oprogramowania, które utrzymywane są bezpłatnie przez społeczność. Oczywiście wszelkie wątpliwości tego typu można rozwiązać poprzez bezpośrednią analizę kodu biblioteki ale wymaga to czasu oraz bardzo dobrego zrozumienia języka `Julia`. Choć zaproponowane rozwiązanie działało „wystarczająco" szybko, to wyniki zdecydowanie nie odpowiadają oczekiwaniom zbudowanym podczas przeglądu literatury. Tym samym należy uznać, że wybrane narzędzia w ramach podejścia opartego o język `Julia` zostały zweryfikowane negatywnie. Podjęte działania naprawcze zostały opisane w ramach fazy drugiej.

[12]Przykładowa pusta strona dokumentacji https://juliareinforcementlearning.org/docs/Which_algorithm_should_I_use/ dostęp 23.10.2022

9.2.2 Faza 2 – Zmiana podejścia, nacisk na testy poprawności implementacji

W związku z uzyskaniem stosunkowo słabych wyników nawet dla bardzo prostych środowisk w ramach fazy pierwszej, zdecydowano się przeprowadzić podobne badania wykorzystując lepiej ugruntowane rozwiązania w ramach języka `Python`. Ponownie wykorzystano biblioteki do uczenia maszynowego. Podstawą była otwarta biblioteka `PyTorch`[13], stworzona przez firmę Facebook (obecnie Meta), z przeznaczeniem do wykorzystania zarówno w projektach naukowych, jak i komercyjnych (w tym wewnątrz firmy). Dodatkowo wykorzystano bibliotekę `Stable-Baselines3`[14], czyli trzecią iterację biblioteki, której celem jest zapewnienie godnych zaufania i solidnych implementacji popularnych metod RL.

Eksperyment 1 W ramach początkowego eksperymentu fazy drugiej zdecydowano się na wykorzystanie własnej implementacji modelu *Rainbow* [109], jednak opartej o zmodyfikowany kod z repozytorium `Rainbow is all you need!`[15]. Decyzja ta została podjęta ponieważ zamierzano uniknąć sytuacji związanej z brakiem pełnego zrozumienia biblioteki, tak jak w wypadku `ReinforcementLearning.jl` z fazy pierwszej. Jedną z podstawowym metod RL jest *Q-Learning* – metoda oparta o stopniową aktualizację tabeli Q (ang. *Q-Table*), czyli tabelę pozwalającą na przypisanie spodziewanej nagrody do akcji w ramach określonego stanu środowiska. Swojego rodzaju rozszerzeniem tej metody jest *Deep Q-Network*[188], DQN, czyli metoda w której zastępuje się tabele Q poprzez aproksymację wykorzystującą głęboką sieć neuronową. Jest to szczególnie uzasadnione w wypadku dużej przestrzeni zarówno stanów jak i akcji w ramach zadanego środowiska. Model DQN mógł być następnie ulepszany poprzez liczne, początkowo niezwiązane ze sobą poprawki w ramach metod RL:

1. *Double DQN*[104] – wykorzystanie dwóch par wag w sieci neuronowej, synchronizowanych co określoną liczbę iteracji, co ma na celu zmniejszenie przeszacowania poprzez choć częściowe oddzielenie wyboru akcji od oceny podjętej akcji. Wykorzystanie tej poprawki sprawia, że proces może być bardziej stabilny.
2. *Prioritized Experience Replay* [240] – metoda ulepszania bufora danych (w ramach którego przechowywana jest historia wcześniej odwiedzonych stanów i podjętych akcji). Poprawa polega na przyporządkowaniu próbkom danych w ramach bufora pamięci priorytetu, tak aby

[13] https://pytorch.org/ dostęp 13.08.2022
[14] https://stable-baselines3.readthedocs.io/en/master/ dostęp 13.08.2022
[15] https://github.com/Curt-Park/rainbow-is-all-you-need dostęp 29.07.2022

następnie częściej wykorzystać próbki „ważniejsze" – prowadzi to do bardziej efektywnego treningu sieci neuronowej.
3. *DuelingNet* [297] – zmiana w architekturze sieci neuronowej – dostosowanie ostatnich warstw sieci w celu oddzielenie oszacowania wartości stanów od szacowania wartości podjęcia określonych akcji.
4. *NoisyNet* [84] – dodanie określonego szumu do wartości wag sieci neuronowej ma na celu wymuszenie dodatkowej eksploracji przez agenta (intuicyjnie – zmiana wag może prowadzić do stałej, skomplikowanej zmiany zachowania agenta na przestrzeni wielu iteracji).
5. *CategoricalDQN* [15] – zmiana podejścia polegająca na określeniu rozkładu z jakiego pochodzi oczekiwana nagroda, zamiast szacowania wyłącznie wartości oczekiwanej nagrody.
6. *N-step Learning* [267] – zmiana w ramach procedury uczenia, zastąpienie kumulacji pojedynczych nagród poprzez uwzględnienie okna składającego się z kolejnych n-kroków. Zmiana prowadzi do przyspieszania treningu połączonego z poprawą jakość uzyskanych wyników.

Połączenie tych poprawek w ramach pojedynczego modelu to właśnie model *Rainbow*[16] Model ten, choć bazuje na „prostym" DQN, pozwolił na uzyskanie bardzo dobrych wyników w ramach pracy [109]. Natomiast repozytorium `Rainbow is all you need!` stanowi świetny materiał edukacyjny, zawierający implementacje i obszerne wyjaśnienie każdego z etapów przejścia od metody bazowej DQN, przez każdą z poprawek, aż do uzyskania „pełnego" modelu *Raibow*. Wykorzystanie wspomnianego repozytorium, w połączeniu ze źródłową literaturą, pozwoliło wyeliminować wątpliwości dotyczące braku zrozumienia wykorzystywanych metod (co było konsekwencją rezultatów z fazy 1).

W odróżnienie od badań z fazy pierwszej zamiast „środowisk zabawkowych", zdecydowano się na stworzenie bardzo prostego środowiska opartego o niewielką instancje problemu komiwojażera (wykorzystano w tym celu instancję `GR17` ze zbioru `TSPLib`). Częścią środowiska była również własna implementacja algorytmu symulowanego wyżarzania (SA), opartego o losowe ruchy typu odwróć, zmodyfikowanego w celu zastosowania RL. Agent w ramach obserwacji dostawał informację o:

a) obecnej temperaturze SA,
b) obecnej iteracji (znormalizowanej jako stosunek numeru iteracji do maksymalnej, docelowej liczby iteracji),
c) różnicy pomiędzy wartością funkcji celu najlepszego dotychczas znalezionego rozwiązania, a dolnym oszacowaniem,

[16]Stad też pochodzi nazwa modelu – od modelu bazowego DQN połączonego z poprawkami, co przypomina według autorów oryginalnej publikacji tęczę.

d) różnicy pomiędzy wartością funkcji celu rozwiązania z obecnej iteracji w ramach SA, a dolnym oszacowaniem.

W celu obliczeń różnic określonych w punktach c) i d) wykorzystano wzór:

$$\frac{f(\pi) - \mathrm{LB}}{\mathrm{UB} - \mathrm{LB}}, \tag{9.3}$$

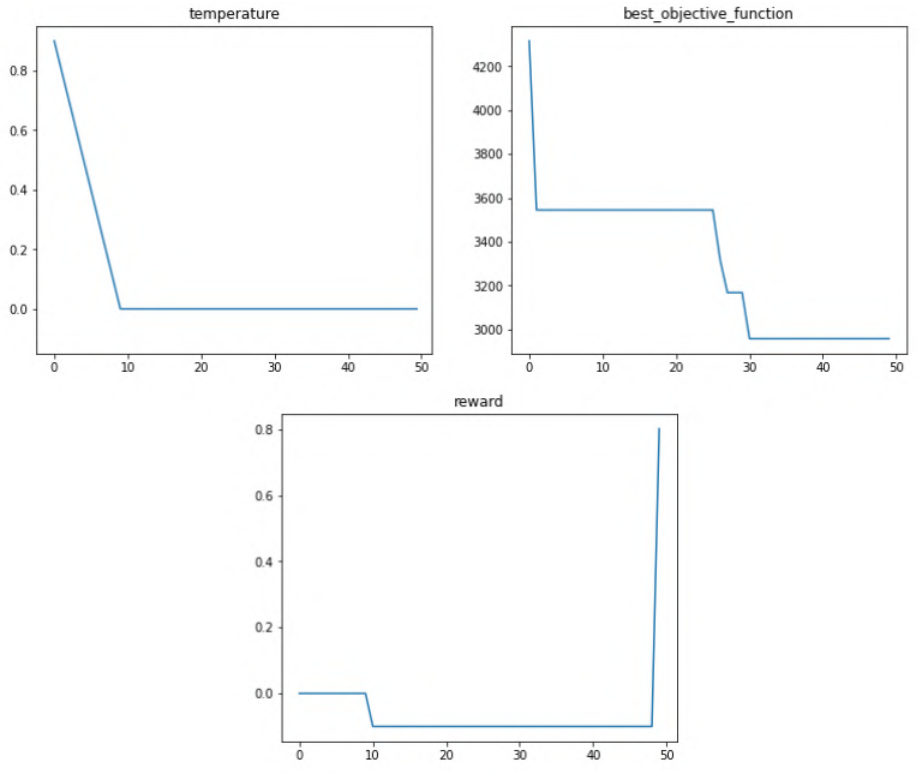
gdzie: LB to dolne oszacowanie, UB to górne oszacowanie, a $f(\pi)$ to długość trasy w ramach kolejności π. Wykorzystano przy tym bardzo proste oszacowania LB i UB – zsumowano dla każdego z miast odpowiednio najkrótsze i najdłuższe połączenie – bez gwarancji dopuszczalności trasy. Agent RL, na podstawie dostarczonych w ramach obserwacji informacji, podejmował decyzje o zmianie temperatury co 20 iteracji SA (badania wstępne wyeliminowały możliwość podejmowania decyzji co iterację SA) – w postaci wyboru dokładnie jednej z akcji:

a) zwiększeniu temperatury w następnych iteracjach SA o 0,1,
b) zwiększeniu temperatury w następnych iteracjach SA o 0,02,
c) zmniejszeniu temperatury w następnych iteracjach SA o 0,02,
d) zmniejszeniu temperatury w następnych iteracjach SA o 0,1.

Sygnał nagrody zadano po ostatniej założonej iteracji algorytmu i wyniósł

$$\frac{\mathrm{UB} - f(\pi_{\mathrm{best\_found}})}{\mathrm{UB} - \mathrm{LB}}, \tag{9.4}$$

gdzie $f(\pi_{\mathrm{best\_found}})$ oznacza długość trasy najlepszego znalezionego rozwiązania w ramach epizodu. Ze względu na stosunkowo mały rozmiar bazowej instancji TSP, przyjęto limit 1000 iteracji SA, co przekładało się na 50 „decyzji" agenta RL. Przekroczenie tego limitu oznaczało koniec epizodu, po którym następował restart zmiennych zarówno środowiska jak i SA. Odpowiadało to rozpoczęciu nowego procesu przeszukiwania, rozpoczynając od nowego, losowego rozwiązania. Nie zdecydowano się na rozpoczynanie z ustalonego rozwiązania ze względu na spodziewane, wynikające z tego szybsze przeuczenie sieci neuronowej. Dodatkowo agent był karany za próbę podwyższenia temperatury ponad limit 1 lub obniżenia jej poniżej 0. Nie przeprowadzono znaczącej optymalizacji licznych hiperparametrów (miedzy innymi: trening obejmował $8 \cdot 10^5$ iteracji SA, przyjęto rozmiar bufora pamięci 10^4 próbek, wsad przy aktualizacji wag wynosił 128, parametr gamma 0,99, alfa 0,2, beta 0,6, wykorzystano funkcję aktywacji RELU, nie zmieniono rozmiaru ani głębokości domyślnej dla biblioteki sieci neuronowej).



Rysunek 9.1: Przykładowy przebieg epizodu dla agenta w ramach eksperymentu 1 z fazy 2. Po lewej, u góry, wartość temperatury SA; po prawej, u góry, wartość najlepszej znalezionej trasy; pod nimi nagroda otrzymywana przez agenta. Zachowanie niepożądane – utrzymywanie temp. 0.

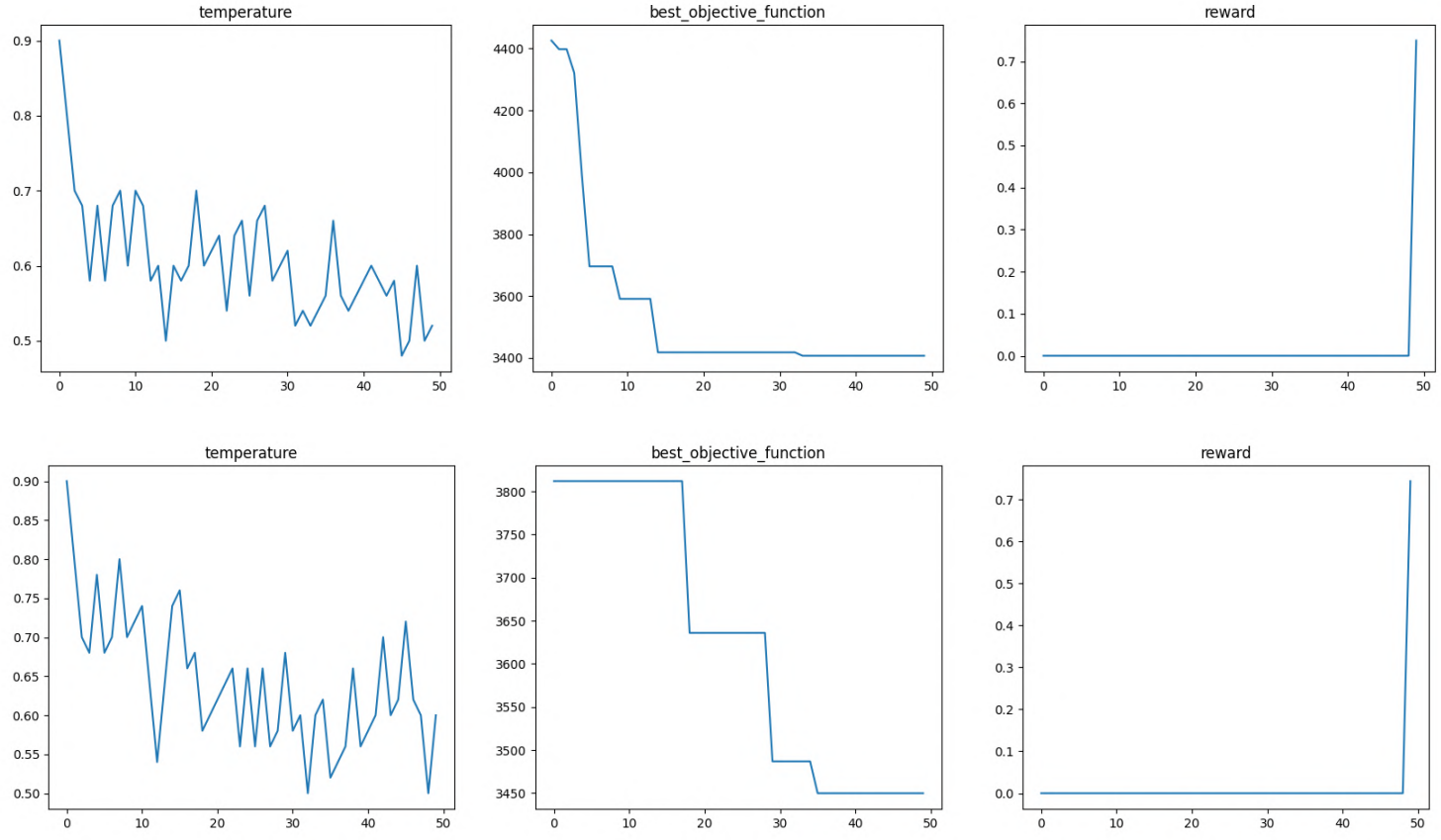
Przykładowy przebieg działania agenta w ramach tak utworzonego środowiska został przedstawiony na rysunku 9.1. Agent nie nauczył się wykorzystywać pełnego potencjału SA. Z uwagi na szybkie obniżenie temperatury do 0, algorytm działa podobnie do przeszukiwania lokalnego. Niepożądane zachowanie agenta przełożyło się na znalezienie trasy o długości zaledwie poniżej 3000, podczas gdy optimum wynosi 2085.

Eksperyment 2 W ramach kolejnego eksperymentu wykorzystano to samo, własne środowisko, jednak zmodyfikowano wykorzystany algorytm RL. Wybrano w tym celu model `PPO` [241] z biblioteki `Stable-Baselines3`. Podejście takie miało zapewnić: a) całkowitą pewność co do poprawności implementacji środowiska; b) choć implementacja *Rainbow* bazująca na repozytorium `Rainbow is all you need!` miała walory edukacyjne, to do-

celowo planowano przetestować wpływ wyboru metody RL na osiągane rezultaty. W tym celu oczywistą przewagą miało wykorzystanie biblioteki `Stable-Baselines3`, która zawiera sprawdzone implementacje metod RL – takich jak między innymi: *Asynchronous Advantage Actor Critic (A3C)*, *Deep Deterministic Policy Gradient (DDPG)*, *DQN*, *Hindsight Experience Replay (HER)*, *Proximal Policy Optimization (PPO)*, *Soft Actor Critic (SAC)* czy *Twin Delayed DDPG (TD3)*. To, w ramach kolejnych faz budowy systemu, umożliwiłoby ocenę wybranych metod RL dla własnego środowiska. Ponownie nie przeprowadzono systematycznej optymalizacji hiperparametrów (trening obejmował 5 milionów iteracji SA, przyjęto rozmiar bufora pamięci 2048 próbek, wsad przy aktualizacji wag wynosił 64, parametr gamma 0,99, lambda 0,95, nie zmieniono rozmiaru ani głębokości domyślnej dla biblioteki sieci neuronowej).

Przykładowe trzy przebiegi wytrenowanego agenta w ramach eksperymentu drugiego z fazy drugiej przedstawia rysunek 9.2. Należy przy tym podkreślić, że sama zmiana modelu nie doprowadziła do poprawy uzyskanych wyników. Wręcz przeciwnie, bardziej skomplikowany model doprowadził do uzyskania wyników w okolicach wartości 3400, co jest jeszcze gorszym wynikiem niż trasa z poprzedniego eksperymentu. Zachowanie wytrenowanego agenta ponownie nie było pożądane. Choć agent początkowo poprawnie zmniejszył temperaturę, to widać brak obniżenia jej poniżej poziomu 0,5. Prowadzi to do zbyt częstego przechodzenia do rozwiązań o gorszej wartości funkcji celu w ramach SA. Pomimo niepożądanego zachowania agenta – eksperyment pozwolił zbudować przypuszczanie o poprawnej implementacji bazując na uzyskaniu zbliżonych rezultatów środowiska dla różnych modeli. W kolejnym eksperymencie postanowiono zmienić sposób dostarczanej nagrody, aby w ten sposób potencjalnie wyeliminować niepożądane zachowanie agenta.

Eksperyment 3 Otrzymane schematy zmiany temperatury w ramach poprzednich dwóch eksperymentów fazy drugiej wyraźnie nie mogły zostać uznane za pożądane. Aby ułatwić proces treningu metody RL, postanowiono wprowadzić 2 zmiany: a) wydłużono liczbę iteracji aby zwiększyć szanse rozwiązania instancji problemu – co niestety trzeba było skompensować poprzez wydłużenie samej procedury treningu; b) zmieniono sposób nagradzania agenta – poprzednie eksperymenty, mogły sugerować, że nagroda jest dostarczana zbyt rzadko (w znaczeniu ang. *Sparse Reward*), przez co zachowania pożądane agenta nie były wystarczająco nagradzane. Zdecydowano się sprawdzić te przypuszczenie poprzez dodanie oprócz nagrody na koniec epizodu (z równania (9.4)), dodatkowej kary w trakcie pozostałych



Rysunek 9.2: Przykładowy przebieg epizodu dla agenta w ramach eksperymentu 2 z fazy 2. Po lewej wartość temperatury SA, w środku wartość najlepszej znalezionej trasy, po prawej nagroda otrzymywana przez agenta. Zachowanie niepożądane – brak dostatecznego obniżenia temperatury.

iteracji w postaci

$$-0{,}1 \cdot \frac{f(\pi_{\text{best\_found}}) - \text{LB}}{\text{UB} - \text{LB}}. \tag{9.5}$$

Kara ta odpowiada oczekiwaniom co do poprawnego zachowania algorytmu przeszukiwania przestrzeni rozwiązań:

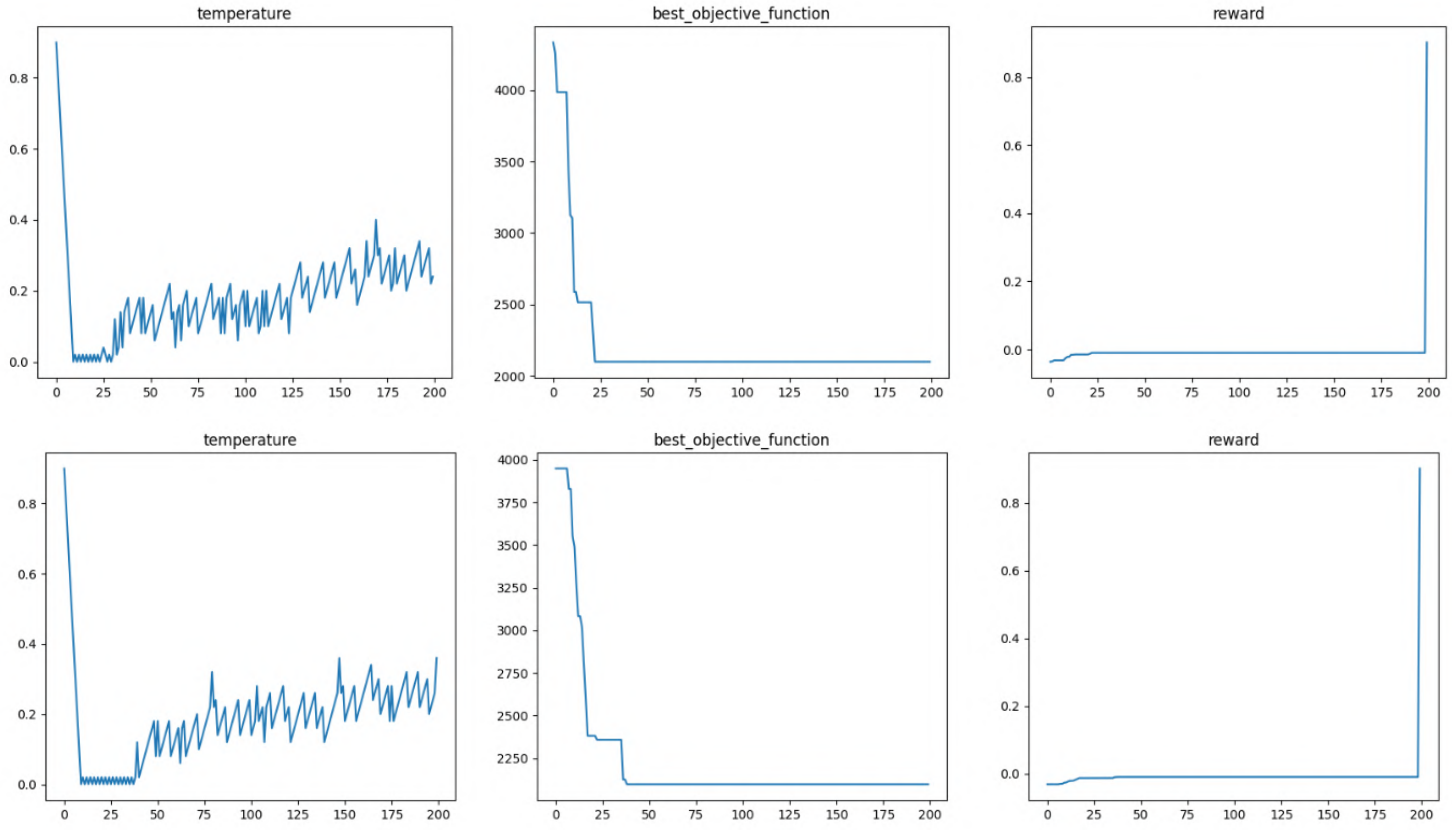
- oczekiwane jest, że algorytm znajdzie rozwiązanie optymalnie jak najszybciej, co zostało odzwierciedlone w postaci ograniczonej kary na przestrzeni iteracji – podejście ma również tą zaletę, że bezcelowe błądzenie jest karane;
- ostatecznie najważniejszą wartością oceny procesu przeszukiwania jest wartość funkcji celu najlepszego znalezionego rozwiązania – wartość obecnego rozwiązania nie ma wpływu na nagrodę, agent nie jest karany za celową eksplorację.

Przykładowe trzy przebiegi wytrenowanego agenta w ramach eksperymentu trzeciego z fazy drugiej przedstawia rysunek 9.3. Zachowanie agenta w ramach środowiska uległo znaczącej poprawie – każdy z przebiegów, pomimo rozpoczęcia z innych rozwiązań, doprowadza do znalezienia rozwiązania o długości trasy w okolicach 2200. Choć nie jest to wartość optymalna, to jest to znaczny postęp w stosunku do poprzednich eksperymentów tej fazy. Dodatkowo otrzymany schemat temperatury wydaje się interesujący – agent gwałtownie obniża temperaturę dla początkowych iteracji co odpowiada zbliżeniu zachowania algorytmu do przeszukiwania lokalnego, by następnie stopniowo ją zwiększać wykorzystując tym samym mechanizm SA do wychodzenia z lokalnego minimum.

Podsumowanie przeprowadzonych badań Przeprowadzone badania wykazały istnienie potencjału wykorzystania zaawansowanych metod opartych o sieci neuronowe oraz uczenie ze wzmocnieniem do modyfikacji działania klasycznego algorytmu metaheurystycznego – symulowanego wyżarzania. Z uwagi na przyjęte ograniczenia zakresu badań, zakończono je na fazie drugiej. Uzyskane wyniki dają natomiast podstawy do naszkicowania planu dalszych prac, opisanego w kolejnym podrozdziale.

9.3 Perspektywy rozwojowe

W kontekście obiecujących wyników przeprowadzonych badań, opracowano strategię dotyczącą kolejnych faz, mającą na celu stworzenie „w pełni rozbudowanego" systemu optymalizacji. Plan ten mógłby ulec zmianie lub modyfikacji, w zależności od uzyskiwanych wyników pośrednich. Daje jednak obraz nakładu pracy niezbędnej do głębszej integracji omawianych metod.



Rysunek 9.3: Przykładowy przebieg epizodu dla agenta w ramach eksperymentu 3 z fazy 2. Po lewej wartość temperatury SA, w środku wartość najlepszej znalezionej trasy, po prawej nagroda otrzymywana przez agenta. Zachowanie agenta wydaje się zbliżone do oczekiwanego.

Faza 3 – Konieczne zmiany w implementacji, umożliwiające porównanie otrzymanych wyników. W ramach fazy 2 udało się osiągnąć działanie agenta RL, które można uznać za zbliżone do poprawnego – jednak wyników nie poddano wnikliwej analizie, która wydaje się zbyteczna przez zbyt wąski kontekst przeprowadzonych badań. Osiągnięte rezultaty otrzymano bazując jedynie na małej instancji problemu TSP. Dodatkowo testy wykonano na zbiorze danych na którym odbywał się trening – co jest złamaniem podziału na zbiór treningowy, walidacyjny i testowy – miało to swoje uzasadnienie z powodu fatalnych rezultatów fazy 1 i spełniło swoją role jako test implementacji. Wady fazy 2 należy wyeliminować aby przejść do badań właściwych, w tym umożliwić istotną analizę porównawczą.

Autor dysertacji w ramach fazy trzeciej wskazuje konieczność naprawy obszarów takich jak:

1. poprawa implementacji środowiska – obecne rozwiązanie nie sprawdzi się w wypadku większych instancji problemów optymalizacji, ograniczenia związane w językiem `Python` spowolnią wykonywanie prac w sposób utrudniający testowanie zróżnicowanych podejść (np. pomiędzy wybranymi metodami RL). Oczekiwanym sposobem naprawy tej wady jest przeniesienie części kodu środowiska do języka `C++` z zastosowaniem *warpperów* łączących kod z różnych języków.
2. rozszerzenie środowiska o wszystkie instancje ze zbioru `TSPLib` – jednak z podziałem wykorzystanych instancji na zbiór treningowy, walidacyjny i testowy. W wypadku braku wystarczającej liczby danych: rozszerzenie zbioru treningowego i walidacyjnego o przykłady generowane w sposób losowy.
3. wykorzystanie dodatkowego problemu optymalizacji dyskretnej – autor dysertacji nie spodziewa się spektakularnych sukcesów w rozwiązywaniu jednego z najlepiej zbadanych problemów optymalizacji dyskretnej (TSP), bez wątpienia podejście oparte o solver `Concorde` okaże się „lepsze". Jednak bazując na przytoczonej literaturze z przeglądu – obiecującym obszarem zastosowania przedstawionego systemu wydają się problemy o rozbudowanych dodatkowych ograniczeniach dla których nie są znane ich własności. Autor dysertacji za szczególnie obiecującą uważa możliwość przeniesienia wytrenowanego modelu z jednego problemu optymalizacji dyskretnej do innego w ramach wykorzystania podejścia opartego o tzw. *Transfer Learning*. Nie oznacza to jednak odejścia od wykorzystania problemu TSP – porównanie jakości uzyskanych wyników pomiędzy wybranymi problemami pozwoliłoby na pełniejszą ocenę zaproponowanego systemu (zarówno w obszarach potencjalnie obiecujących jak i bez unikania trudnych

obszarów zastosowania).

Tym samym za cel fazy 3 przyjmuje się modyfikację istniejącego systemu umożliwiając otrzymanie wyników, które dopiero następnie będzie można poddać dalszej analizie porównawczej – np. poprzez wykorzystanie testów statystycznych. Krok ten skupia się na rozszerzeniu wykorzystanych danych tak aby umożliwić „uczciwe" porównanie metod klasycznych do zaproponowanego rozwiązania własnego.

Faza 4 – Zmiana algorytmu optymalizacji. W ramach fazy 2 wykorzystano algorytm symulowanego wyżarzania, gdyż było to uzasadnione poprzez fatalne wyniki fazy 1 Należy jednak podkreślić, że wybrano algorytm SA ze względu na jego prostotę oraz łatwy sposób wpływania na przebieg przeszukiwania przestrzeni rozwiązań poprzez zmianę wartości temperatury. Algorytm SA nie jest uznawany jako „najlepsza" metoda rozwiązywania większości z problemów optymalizacji dyskretnej (nie w rozumieniu ang. *State of the Art*) – szczególnie nie dla problemu TSP. Dodatkowo działanie SA opiera się o losowe transformacje pojedynczego rozwiązania, przez co sam proces przeszukiwania ma charakter losowy. Utrudnia to „trening" metod RL – autor dysertacji oczekuje lepszej synergii z algorytmem wykorzystującym przegląd otoczenia rozwiązania w ramach sąsiedztwa. Naturalnym kandydatem przy zmianie algorytmu staje się więc przeszukiwanie z zabronieniami – jednak element pamięci może okazać się problematyczny z perspektywy metod RL opartych o decyzyjny proces Markowa.

Alternatywnym, ciekawym krokiem w ramach tej fazy może być stworzenie własnej hybrydowej metody optymalizacji dyskretnej poprzez połączenie elementów innych, znanych algorytmów przeszukiwania. Pewną przesłanką uzasadniającą takie działanie jest często rozważany w kontekście działania metod RL dylemat pomiędzy eksploracją a wykorzystaniem zebranej wiedzy (ang. *exploration and exploitation dilemma*). Dylemat, ten w ramach przeszukiwania przestrzeni rozwiązań jest reprezentowany poprzez odpowiednio: mechanizmy pozwalające wydostać się z lokalnego optimum oraz mechanizmy przeszukiwania sąsiedztwa, które prowadzą do lokalnego optimum. Tym samym podejście hybrydowe polegałoby na wykorzystaniu agenta który, na przestrzeni iteracji algorytmu przeszukiwania, wybiera akcję odpowiadającą wykorzystaniu pewnego mechanizmu przeszukiwania spośród kilku alternatyw. Za cel fazy 4 przyjmuje się więc wybór właściwego algorytmu przeszukiwania, który będzie wykorzystany jako część środowiska w dalszych badaniach – w ramach metody hybrydowej razem z agentem opartym o RL.

Faza 5 – Zmiany w ramach metod RL oraz zastosowanych sieci neuronowych. W ramach metod RL w fazie 2 (*Rainbow* oraz *PPO*) wykorzystano bardzo proste modele sieci neuronowych o niewielkiej liczbie warstw. Było to w pełni zrozumiałe w ramach eksperymentów opierających się o stosunkowo niską liczbę próbek danych pochodzących z pojedynczej, małej instancji problemu TSP. Wykorzystanie modelu głębokiego na ograniczonej próbce danych wiąże się zwykle z trudnością w treningu oraz przyspieszeniem niekorzystnego procesu przeuczenia sieci neuronowej. Na podstawie przeglądu literatury, przytoczonego w tym rozdziale, jako warte przetestowania w tej fazie uznaje się podejście wykorzystujące model *LSTM* oraz *transformer*.

Kolejnym krokiem w ramach tej fazy są zamiany w ramach metod RL. Jak opisano w ramach przeprowadzonych eksperymentów obliczeniowych, można z powodzeniem wykorzystać bibliotekę `Stable-Baselines3` w celu wyboru ostatecznie wykorzystanej metody RL. Dodatkowo, co zostało już zaobserwowane w ramach przedstawionych badań – pomiędzy eksperymentem 2 i 3 z fazy drugiej – sygnał nagrody jest wyjątkowo ważny w kontekście uzyskania prawidłowego zachowania agenta. Tym samym badania dotyczące modyfikacji funkcji nagrody również należy włączyć w ramy fazy piątej. Przykładową modyfikacją, wartą rozważania, może być dodanie dodatkowego komponentu kary za zbyt częste odwiedzanie tych samych rozwiązań przez algorytm przeszukiwania (na zasadzie analogii do mechanizmów uniemożliwiających wpadanie w cykle czy też mechanizmów pamięci). Na podobnej zasadzie systematyczny dobór informacji przekazywanej w ramach obserwacji jest również uzasadniony.

Za cel fazy 5 przyjmuje się wybranie modelu oraz wstępnej struktury sieci neuronowej wykorzystywanej w ramach metod RL, jak również samej metody RL.

Faza 6 – Włączenie elementów FLA jako części obserwacji Na przestrzeni dysertacji wykorzystane zostały zarówno metody neuronowe, jak i analiza krajobrazu przestrzeni rozwiązań. Nie była jednak poruszana kwestia połączenia tych podjeść. Autor dysertacji uważa, że przedstawiony system z powodzeniem mógłby zostać uzupełniony o elementy FLA – jako źródło dodatkowej informacja w ramach obserwacji.

Naturalnymi kandydatami do zastosowania analizy krajobrazu przestrzeni rozwiązań w ramach zaproponowanego systemu są metrykami FLA, takie jak: szorstkość, neutralność, modalność, rozkład przystosowania czy gęstość stanów. Należy jednak podkreślić, że w ramach tej fazy oczekiwane jest podejście inne niż to przedstawione w rozdziałach dotyczących badań

wyłącznie nad metodą LON – niepożądane jest obliczenie globalnych wartości przytoczonych metryk. Zamiast tego oczekuje się szacowania wartości metryk na bazie dotychczas odwiedzonych rozwiązań w ramach przeszukiwania za pomocą hybrydowego systemu, wykorzystującego RL. A nawet ograniczenie zastosowania metryk FLA do szacowania małych wycinków przestrzeni, takich jak np. rozwiązań obejmujących wyłącznie wybrane sąsiedztwo. Tym samym, za obiecujące oczekuje się wykorzystanie metody FLA przedmiotowo – nie jako globalnego opisu instancji, do czego zostały stworzone, a jako kolejne, dodatkowego źródła informacji, z konkretnym celem, jakim jest poprawa procesu przeszukiwania przestrzeni rozwiązań.

Za cel fazy 6 uznano badania dotyczące wpływu zastosowania wybranych metryk FLA jako źródła dodatkowej informacji w ramach zaproponowanego hybrydowego systemu. Ocenę taką można przeprowadzić poprzez testy statystyczne, korzystając z wyników uzyskanych w ramach procedury zbliżonej do testów A/B – tj. pomiędzy systemem z metrykami FLA dodanymi jako część stanu w ramach obserwacji oraz systemem bez tych metryk.

Faza 7 – Optymalizacja hiper-parametrów oraz uzyskanie finalnej wersji systemu Ostatnim krokiem zaplanowanych badań jest optymalizacja hiper-parametrów. Nie oznacza to, że pewna optymalizacja nie zostanie wykonana w ramach wcześniejszych faz. Jak podkreślono wcześniej w ramach tego rozdziału, metody RL są „czułe" na dobór hiper-parametrów, wiec dobór wartości parametrów może być konieczny w pewnym zakresie również na przestrzeni wcześniejszych etapów. W ramach tego etapu warto uwzględnić systematyczne podejście do 2 obszarów: a) sposobu doboru ostatecznych hiper-parametrów dla opisywanego, autorskiego systemu b) poświęcić czas równy treningowi metody hybrydowej opartej o zastosowanie RL, do dłuższego doboru hiper-parametrów dla klasycznych metod optymalizacji. Autor dysertacji przyznaje, że podejście takie jest wymagane, aby pokazać (i nie ukrywać, jak jest to często robione w literaturze) skalę kompromisu, który wiąże się z zastosowaniem sieci neuronowych do zadania optymalizacji dyskretnej. Za cel fazy 7 przyjmuje się dobór ostatecznie wykorzystanych hiper-parametrów, zarówno dla systemu hybrydowego, jak i metod klasycznych, z którymi system ten zostanie porównany.

9.4 Wnioski i uwagi

Przedstawiony rozdział opisuje tematykę wykorzystania sztucznych sieci neuronowych do rozwiązywania problemów optymalizacji dyskretnej. Pogłębiono w nim zarówno przegląd literatury, jak i pokazano własne, niepu-

blikowane badania w tej dziedzinie. Poszerzono przegląd literatury, będący bazą dla przeprowadzonych badań, o metody rozwiązywania problemów optymalizacji wykorzystujące: a) klasyczne podejście do sieci neuronowych, b) uczenie głębokie oraz c) uczenie ze wzmocnieniem. Część przywołanej literatury wykorzystywała oba, ostanie z wymienionych podejść.

W dalszej części tekstu zaprezentowano wyniki autorskich eksperymentów, w tym dwie ukończone fazy budowania prototypu systemu optymalizacji dyskretnej opartego o RL, wykorzystującego modele *Rainbow* oraz *PPO*. Zakres przytoczonych badań własnych, choć ograniczony,okazał się wystarczający do wyciągnięcia pewnych wniosków. W tym najważniejszego z nich, który świadczy pozytywnie o *możliwości zastępowania elementów klasycznych algorytmów metaheurystycznych przez zaawansowane metody oparte o sieci neuronowe w ramach uczenia ze wzmocnieniem* – co potwierdza jedną z głównych tez doktoratu.

W ostatniej części rozdziału przedstawiono plan dalszych badań, mających na celu szersze wykorzystanie potencjału uczenia głębokiego oraz uczenia ze wzmocnieniem w rozwiązywaniu problemów optymalizacji dyskretnej. Plan wskazuje, w oparciu o stan wiedzy i przeprowadzone eksperymenty, kolejne kroki w kierunku głębszej integracji metod optymalizacji dyskretnej, uczenia maszynowego oraz analizy przestrzeni rozwiązań.

Rozdział 10

Zakończenie

Na podstawie tez rozprawy i w oparciu o przedstawione w poprzednich rozdziałach wyniki badań własnych, można przedstawić następujące wnioski. Wyniki rozprawy stanowią empiryczny dowód, że *korzystając ze sztucznych sieci neuronowych lub analizy krajobrazu przestrzeni rozwiązań możliwe jest poprawienie efektywności działania konwencjonalnych algorytmów metaheurystycznych, rozumiane jako uzyskiwanie w tym samym koszcie (np. czasowym) rozwiązań o wyższej jakości.* Uzasadnieniem tego wniosku są badania przedstawione w rozdziale 8, które obrazują możliwość modyfikacji nawet najbardziej kluczowych i fundamentalnych elementów klasycznej metaheurystyki. W badaniach tych z powodzeniem zamieniono klasyczną listę zabronień w algorytmie TS na neuronowy mechanizm zapominania; przy czym modyfikacja ta pozwoliła na uzyskanie rozwiązań o lepszej wartości funkcji celu dla cyklicznego problemu gniazdowego.

W rozdziale 6 przedstawiono badania potwierdzające tezę postulującą, że *korzystając ze sztucznych sieci neuronowych oraz analizy krajobrazu przestrzeni rozwiązań możliwe jest rozwiązywanie problemu wyboru algorytmu* – pozwalając na wyłonienie „zwycięzcy" z ograniczonego portfolio komplementarnych algorytmów. Z powodzeniem wykorzystano jedną z metod FLA: analizę sieci lokalnych optimów, jako sposób na ekstrakcję cech z wybranych instancji bazowego problemu optymalizacji – problemu komiwojażera. Cechy te zostały użyte między innymi przez klasyczne sieci neuronowe w ostatnim kroku rozwiązania problemu wyboru algorytmu, czyli klasyfikacji wygrywającego algorytmu.

Znaczący koszt obliczeniowy konieczny do stworzenia grafu LON – stanowiący istotną wadę tej metody – był motywacją do próby ustalenia związku pomiędzy metodą próbkowania przestrzeni, a wartościami uzyskanych miar. Znalezienie miar, których wartości nie byłyby uzależnione

od stopnia kompletności próbkowania przestrzeni rozwiązań, a jednocześnie „oddawałyby" cechy tej przestrzeni, byłoby znacznym sukcesem. Tematyka ta została poruszona w badaniach opisanych w rozdziale 7, które niestety wskazały, że *sposób przeprowadzenia próbkowania przestrzeni rozwiązań może mieć istotny wpływ na wartości estymacji miar tej przestrzeni.* Badania te przeprowadzono na realistycznym problemie – cyklicznym problemie przepływowym z gniazdami produkcyjnymi z pojedynczymi operatorami. Wraz ze wzrostem liczby próbek przestrzeni rozwiązań, wskazano zmianę zarówno wariancji, jak i wartości oczekiwanej opisujących ją miar. Tym samym wyniki eksperymentu nie mogą stanowić uzasadnienia stosowania metody LON – wykorzystanej w sposób przedstawiony w badaniach własnych – jako komponentu zaawansowanego systemu optymalizacji.

Choć stan badań przedstawiony w rozdziale 9 zasługuje jedynie na określenie ich jako badania wstępne – to jest to zupełnie wystarczające, aby pozytywnie zweryfikować tezę o *możliwości zastępowania elementów klasycznych algorytmów metaheurystycznych przez zaawansowane metody oparte o sieci neuronowe w ramach uczenia ze wzmocnieniem.* Podczas kilku iteracji eksperymentów z powodzeniem zmodyfikowano działanie heurystyki symulowanego wyżarzania. Wykorzystując narzędzia uczenia ze wzmocnieniem, wytrenowano agenta (opartego o sieć neuronową) do podejmowania akcji które wpływały na wartość temperatury w kolejnych iteracjach procesu przeszukiwania rozwiązań problemu TSP. Pomimo początkowych trudności w eksperymentach, udało się uzyskać interesujące schematy schładzania, czy raczej zmiany temperatury. Na podstawie obiecujących wyników stworzono plan proponowanych, kolejnych faz budowy systemu optymalizacji dyskretnej wykorzystującego uczenie ze wzmocnieniem.

Wyniki przywołanych badań (z wyjątkiem badań wstępnych przedstawionych w rozdziale 9), zostały opublikowane w materiałach o zasięgu międzynarodowym, w pracach dostępnych w języku angielskim: Bożejko, Gnatowski, **Niżyński**, Wodecki – 2016 [29]; Bożejko, Gnatowski, **Niżyński**, Affenzeller, Beham – 2018 [28], Gnatowski, **Niżyński** – 2020 [92].

Autor dysertacji opublikował również prace niezwiązane bezpośrednio z tematyką pracy doktorskiej: a) wykorzystanie sieci neuronowych poza kontekstem optymalizacji dyskretnej, do klasyfikacji poprawnej motoryki chodu – Kluwak, **Niżyński** – 2020 [146]; b) zrównoleglenie specjalistycznego algorytmu szeregowania dla dwu-maszynowego gniazda robotycznego w procesie spawania ram rowerowych – Gnatowski, **Niżyński** – 2021 [93].

Możliwe kierunki dalszych badań

W ocenie autora dysertacja może stanowić punkt wyjścia do dalszych badań dotyczących wykorzystania metod opartych o uczenie maszynowe i analizę krajobrazu przestrzeni rozwiązań w rozwiązywaniu problemów optymalizacji dyskretnej.

Za interesujące i warte rozważenia w przyszłych badaniach można uznać dalsze eksplorowanie wykorzystania sztucznych sieci neuronowych szczególnie w kontekście problemów szeregowania zadań oraz marszrutyzacji. Jednym z podejść, do tej pory nierozpatrywanym w badaniach własnych, jest bezpośrednie konstruowanie rozwiązań problemów za pomocą sieci neuronowych. W literaturze można znaleźć wiele takich prób, co zostało opisane w rozdziale 9. Niemniej, w znaczącej mierze metody tego typu mają na celu przezwyciężenie ograniczeń związanych z samą specyfiką sieci neuronowych — i choć z tej perspektywy są niezwykle ciekawe – to niestety niekoniecznie zapewniają wystarczającą poprawę jakości uzyskanych rozwiązań. Ich celem nie jest więc walka o najlepsze wyniki dla znanych zbiorów danych, co stanowi wyraźny kontrast do klasycznych metod optymalizacji dyskretnej.

Tym samym potencjalnie bardziej obiecującą alternatywą – rozważaną w badaniach własnych – jest ulepszenie wybranych elementów algorytmów metahuerystycznych z wykorzystaniem sztucznych sieci neuronowych. W ramach tego drugiego podejścia można wskazać przykład badań własnych opisanych w rozdziale 8, gdzie modyfikacji poddano mechanizm pamięci dla metaheurystki przeszukiwania z zabronieniami. Dalsza praca mogłaby obejmować próbę poprawy kolejnych mechanizmów i elementów algorytmu, np. zmianę modelu neuronu w mechanizmie zabronień czy stworzenie dedykowanego rozwiązania zapobiegającego powstawaniu cykli.

Jednak to kontynuacja badań rozpoczętych w rozdziale 9, a dotyczących wykorzystania uczenie ze wzmocnieniem jest według autora szczególnie obiecująca. Istnieje wiele rokujących kierunków badań: od zmiany wykorzystanego algorytmu metaheurystycznego, przez wybór metody RL (czy też architektury sieci neuronowej agenta), aż do doboru sposobu reprezentacji informacji o stanie przeszukiwania zawartych w obserwacji. Szczegółowy plan proponowanych, przyszłych badań został nakreślony w oddzielnej części poprzedniego rozdziału 9.3. Rozbudowany system optymalizacji, wykorzystujący sztuczne sieci neuronowe oraz klasyczne algorytmy, będący zwieńczeniem tego rodzaju eksperymentów, mógłby być w dalszej kolejności wzbogacony o dodatkowe elementy metod analizy krajobrazu rozwiązań.

Ze względu na wyniki badań przestawione w rozdziałach 6 oraz 7, autor dysertacji ocenia metodę analizy lokalnych optimów jako niezbyt obiecującą

w kontekście przyszłych badań związanych z próbą stworzenia zaawansowanego systemem optymalizacji dyskretnej. Wynika to zarówno z dużego nakładu obliczeniowego koniecznego do utworzenia grafu LON, jak i zmienności miar w zależności od stopnia próbkowania przestrzeni.

Nie oznacza to całkowitego przekreślenia użyteczności metod analizy przestrzeni rozwiązań. Po prostu za lepiej rokujące można wskazać wykorzystanie prostszych metod i metryk, takich jak: szorstkość, neutralność czy gęstość stanów – i próba szacowania ich podczas „pracy online" algorytmów przeszukiwania przestrzeni rozwiązań. Tym samym kierunek potencjalnych, dalszych badań charakteryzowałby się bardziej utylitarnym podejściem do miar przestrzeni rozwiązań – z koncentracją na ocenie ich przydatności, jako źródła unikalnej reprezentacji danych. Prawdopodobnie podejście to oznaczałoby konieczność wykorzystania miar w ograniczonym zakresie i akceptację ich zmienności w trakcie działania zmodyfikowanej metaheurystyki. Decyzja taka wiązałaby się również z porzuceniem prób szacowania wartości miar w sposób „globalny", tj. charakteryzujący instancję problemu jako całość, przed rozpoczęciem właściwego przeszukiwania przestrzeni rozwiązań za pomocą proponowanego zaawansowanego systemu.

Bibliografia

[1] J. Agnese, J. Herrera, H. Tao, oraz X. Zhu, *A survey and taxonomy of adversarial neural networks for text-to-image synthesis*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 10 (2020).

[2] R. Ahmad oraz D. Kim, *An extended self-organizing map based on 2-opt algorithm for solving symmetrical traveling salesperson problem*, Neural Computing and Applications, 26 (2015), str. 987–994.

[3] A. Allahverdi, *The third comprehensive survey on scheduling problems with setup times/costs*, European Journal of Operational Research, 246 (2015), str. 345–378.

[4] A. Allahverdi, J. N. Gupta, oraz T. Aldowaisan, *A review of scheduling research involving setup considerations*, Omega, 27 (1999), str. 219–239.

[5] B. Angeniol, G. D. L. C. Vaubois, oraz J.-Y. Le Texier, *Self-organizing feature maps and the travelling salesman problem*, Neural Networks, 1 (1988), str. 289–293.

[6] D. L. Applegate, R. E. Bixby, V. Chvatal, oraz W. J. Cook, *The Traveling Salesman Problem: A Computational Study*, Princeton Series in Applied Mathematics, Princeton University Press, Princeton, NJ, USA, 2006.

[7] D. Bahdanau, K. Cho, oraz Y. Bengio, *Neural machine translation by jointly learning to align and translate*, arXiv preprint arxiv:1409.0473, (2015).

[8] D. Bai, M. Tang, Z.-H. Zhang, oraz E. D. Santibanez-Gonzalez, *Flow shop learning effect scheduling problem with release dates*, Omega, 78 (2018), str. 21–38.

[9] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, oraz I. Mordatch, *Emergent tool use from multi-agent interaction*, Machine Learning, Cornell University, (2019).

[10] P. Baldi oraz P. J. Sadowski, *Understanding dropout*, Advances in neural information processing systems, 26 (2013).

[11] K. Banachewicz, L. Massaron, oraz A. Goldbloom, *The Kaggle Book: Data analysis and machine learning for competitive data science*, Packt Publishing, 2022.

[12] S. Basu, *Tabu search implementation on traveling salesman problem and its variations: a literature survey*, American Journal of Operations Research, (2012).

[13] O. Battaïa oraz A. Dolgui, *A taxonomy of line balancing problems and their solutionapproaches*, International Journal of Production Economics, 142 (2013), str. 259–277.

[14] J. E. Beasley, *Or-library: distributing test problems by electronic mail*, Journal of the operational research society, 41 (1990), str. 1069–1072.

[15] M. G. Bellemare, W. Dabney, oraz R. Munos, *A distributional perspective on reinforcement learning*, w International Conference on Machine Learning, PMLR, 2017, str. 449–458.

[16] I. Bello, H. Pham, Q. V. Le, M. Norouzi, oraz S. Bengio, *Neural combinatorial optimization with reinforcement learning*, arXiv preprint arXiv:1611.09940, (2016).

[17] M. Ben-Daya oraz M. Al-Fawzan, *A tabu search approach for the flow shop scheduling problem*, European journal of operational research, 109 (1998), str. 88–95.

[18] E. M. Bender, T. Gebru, A. McMillan-Major, oraz S. Shmitchell, *On the dangers of stochastic parrots: Can language models be too big?*, w Proceedings of the 2021 ACM conference on fairness, accountability, and transparency, 2021, str. 610–623.

[19] Y. Bengio, A. Lodi, oraz A. Prouvost, *Machine learning for combinatorial optimization: a methodological tour d'horizon*, European Journal of Operational Research, 290 (2021), str. 405–421.

[20] J. Bergstra oraz Y. Bengio, *Random search for hyper-parameter optimization.*, Journal of machine learning research, 13 (2012).

[21] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al., *Dota 2 with large scale deep reinforcement learning*, arXiv preprint arXiv:1912.06680, (2019).

[22] V. R. Bhargava oraz M. Velasquez, *Ethics of the attention economy: The problem of social media addiction*, Business Ethics Quarterly, 31 (2021), str. 321–359.

[23] C. Bishop, *Pattern Recognition and Machine Learning*, Information Science and Statistics, Springer New York, 2016.

[24] P. Bojanowski, E. Grave, A. Joulin, oraz T. Mikolov, *Enriching word vectors with subword information*, Transactions of the association for computational linguistics, 5 (2017), str. 135–146.

[25] L. Bottou, F. E. Curtis, oraz J. Nocedal, *Optimization methods for large-scale machine learning*, SIAM review, 60 (2018), str. 223–311.

[26] W. Bożejko, A. Gnatowski, R. Idzikowski, oraz M. Wodecki, *Cyclic flow shop scheduling problem with two-machine cells*, Archives of Control Sciences, 27 (2017), str. 151–167.

[27] W. Bożejko, A. Gnatowski, R. Klempous, M. Affenzeller, oraz A. Beham, *Cyclic scheduling of a robotic cell*, w 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), IEEE, 2016, str. 379–384.

[28] W. Bożejko, A. Gnatowski, T. Niżyński, M. Affenzeller, oraz A. Beham, *Local optima networks in solving algorithm selection problem for tsp*, w International Conference on Dependability and Complex Systems, Springer, 2018, str. 83–93.

[29] W. Bożejko, A. Gnatowski, T. Niżyński, oraz M. Wodecki, *Tabu Search Algorithm with Neural Tabu Mechanism for the Cyclic Job Shop Problem*, w Artificial Intelligence and Soft Computing. ICAISC 2016. Lecture Notes in Computer Science, L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, oraz J. M. Zurada, ed., vol. 9119 of Lecture Notes in Computer Science, Springer International Publishing, Cham, 2016, str. 409–418.

[30] W. Bożejko, A. Gnatowski, J. Pempera, oraz M. Wodecki, *Parallel tabu search for the cyclic job shop scheduling problem*, Computers & Industrial Engineering, 113 (2017), str. 512–524.

[31] W. Bożejko, Z. Hejducki, M. Uchroński, oraz M. Wodecki, *Solving the flexible job shop problem on multi-gpu*, Procedia Computer Science, 9 (2012), str. 2020–2023.

[32] W. Bożejko, J. Pempera, oraz M. Wodecki, *Minimal cycle time determination and golf neighborhood generation for the cyclic flexible job shop problem*, Bulletin of the Polish Academy of Sciences. Technical Sciences, 66 (2018).

[33] W. Bożejko oraz M. Uchroński, *A neuro-tabu search algorithm for the job shop problem*, w International Conference on Artificial Intelligence and Soft Computing, Springer, 2010, str. 387–394.

[34] W. Bożejko, M. Uchroński, oraz M. Wodecki, *The new golf neighborhood for the exible job shop problem*, Procedia Computer Science, 1 (2010), str. 289–296.

[35] ——, *Parallel neuro-tabu search algorithm for the job shop scheduling problem*, w International Conference on Artificial Intelligence and Soft Computing, Springer, 2013, str. 489–499.

[36] W. Bożejko oraz M. Wodecki, *On the theoretical properties of swap multimoves*, Operations Research Letters, 35 (2007), str. 227–231.

[37] W. Bożejko, A. Gnatowski, J. Pempera, oraz M. Wodecki, *Parallel tabu search metaheuristics for the cyclic job shop scheduling problem*, Submited to: Computers & Industrial Engineering, (2015).

[38] W. Bożejko, J. Pempera, oraz M. Wodecki, *Parallel Simulated Annealing Algorithm for Cyclic Flexible Job Shop Scheduling Problem*, w Lecture Notes in Artificial Intelligence, vol. 9120, Springer International Publishing, 2015, str. 603–612.

[39] W. Bożejko, M. Uchroński, oraz M. Wodecki, *Block approach to the cyclic flow shop scheduling*, Computers & Industrial Engineering, 81 (2015), str. 158–166.

[40] X. Bresson oraz T. Laurent, *The transformer network for the traveling salesman problem*, arXiv preprint arXiv:2103.03012, (2021).

[41] A. Brock, J. Donahue, oraz K. Simonyan, *Large scale gan training for high fidelity natural image synthesis*, arXiv preprint arXiv:1809.11096, (2018).

[42] Ł. Brocki oraz D. Koržinek, *Kohonen self-organizing map for the traveling salesperson problem*, w Recent Advances in Mechatronics, Springer, 2007, str. 116–119.

[43] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., *Language models are few-shot learners*, Advances in neural information processing systems, 33 (2020), str. 1877–1901.

[44] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, oraz S. Colton, *A survey of monte carlo tree search methods*, IEEE Transactions on Computational Intelligence and AI in games, 4 (2012), str. 1–43.

[45] P. Brucker, S. Heitmann, oraz J. Hurink, *Flow-shop problems with intermediate buffers*, OR Spectrum, 25 (2003), str. 549–574.

[46] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, et al., *Sparks of artificial general intelligence: Early experiments with gpt-4*, arXiv preprint arXiv:2303.12712, (2023).

[47] S. Capraz, H. Azyikmis, oraz A. Ozsoy, *An optimized gpu-accelerated route planning of multi-uav systems using simulated annealing*, International Journal of Machine Learning and Computing, 10 (2020).

[48] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, et al., *Universal sentence encoder*, arXiv preprint arXiv:1803.11175, (2018).

[49] V. Černỳ, *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*, Journal of optimization theory and applications, 45 (1985), str. 41–51.

[50] I. A. Chaudhry oraz A. A. Khan, *A research survey: Review of flexible job shop scheduling techniques*, International Transactions in Operational Research, 23 (2016), str. 551–591.

[51] B. Chen, C. N. Potts, oraz G. J. Woeginger, *A review of machine scheduling: Complexity, algorithms and approximability*, Handbook of combinatorial optimization, (1998), str. 1493–1641.

[52] T. Chen oraz C. Guestrin, *Xgboost: A scalable tree boosting system*, w Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, 2016, str. 785–794.

[53] F. Chicano, F. Daolio, G. Ochoa, S. Vérel, M. Tomassini, oraz E. Alba, *Local optima networks, landscape autocorrelation and heuristic search performance*, w International Conference on Parallel Problem Solving from Nature, Springer, 2012, str. 337–347.

[54] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, oraz Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, arXiv preprint arXiv:1406.1078, (2014).

[55] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, oraz G. E. Dahl, *On empirical comparisons of optimizers for deep learning*, arXiv preprint arXiv:1910.05446, (2019).

[56] F. Chollet, *Deep Learning with Python*, Manning Publications Company, 2017.

[57] D.-A. Clevert, T. Unterthiner, oraz S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (elus)*, arXiv preprint arXiv:1511.07289, (2015).

[58] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al., *Training verifiers to solve math word problems*, arXiv preprint arXiv:2110.14168, (2021).

[59] P. Collard, S. Verel, oraz M. Clergue, *Local search heuristics: Fitness cloud versus fitness landscape*, arXiv preprint arXiv:0709.4010, (2007).

[60] R. K. Congram, C. N. Potts, oraz S. L. van de Velde, *An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem*, INFORMS Journal on Computing, 14 (2002), str. 52–67.

[61] T. H. Cormen, C. E. Leiserson, R. L. Rivest, oraz C. Stein, *Introduction to algorithms*, MIT press, 2009.

[62] Y. Crama, V. Kats, J. Van de Klundert, oraz E. Levner, *Cyclic scheduling in robotic flowshops*, Annals of operations Research, 96 (2000), str. 97–124.

[63] G. A. Croes, *A method for solving traveling-salesman problems*, Operations Research, 6 (1958), str. 791–812.

[64] Z. J. Czech, *Statistical measures of a fitness landscape for the vehicle routing problem*, w 2008 IEEE International Symposium on Parallel and Distributed Processing, IEEE, 2008, str. 1–8.

[65] F. Daolio, M. Tomassini, S. Vérel, oraz G. Ochoa, *Communities of minima in local optima networks of combinatorial spaces*, Physica A: Statistical Mechanics and its Applications, 390 (2011), str. 1684–1694.

[66] F. Daolio, S. Verel, G. Ochoa, oraz M. Tomassini, *Local optima networks of the quadratic assignment problem*, w IEEE Congress on Evolutionary Computation, IEEE, 2010, str. 1–8.

[67] ——, *Local optima networks and the performance of iterated local search*, w Proceedings of the 14th annual conference on Genetic and evolutionary computation, 2012, str. 369–376.

[68] ——, *Local optima networks of the permutation flow-shop problem*, w International Conference on Artificial Evolution (Evolution Artificielle), Springer, 2013, str. 41–52.

[69] Y. Davidor, *Epistasis variance: A viewpoint on ga-hardness*, w Foundations of genetic algorithms, vol. 1, Elsevier, 1991, str. 23–35.

[70] L. Davis, *Adapting operator probabilities in genetic algorithms*, w Proceedings of the 3rd International Conference on Genetic Algorithms, 1989, str. 61–69.

[71] M. Dawande, H. N. Geismar, S. P. Sethi, oraz C. Sriskandarajah, *Sequencing and scheduling in robotic cells: Recent developments*, Journal of Scheduling, 8 (2005), str. 387–426.

[72] M. W. Dawande, H. N. Geismar, S. P. Sethi, oraz C. Sriskandarajah, *Throughput optimization in robotic cells*, vol. 101, Springer Science & Business Media, 2007.

[73] L. De Giovanni oraz F. Pezzella, *An improved genetic algorithm for the distributed and flexible job-shop scheduling problem*, European journal of operational research, 200 (2010), str. 395–408.

[74] K. Deb oraz D. E. Goldberg, *Sufficient conditions for deceptive and easy binary functions*, Annals of mathematics and Artificial Intelligence, 10 (1994), str. 385–408.

[75] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, oraz L.-M. Rousseau, *Learning heuristics for the tsp by policy gradient*, w International conference on the integration of constraint programming, artificial intelligence, and operations research, Springer, 2018, str. 170–181.

[76] J. Devlin, M.-W. Chang, K. Lee, oraz K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, arXiv preprint arXiv:1810.04805, (2018).

[77] B. Doerr oraz F. Neumann, *A survey on recent progress in the theory of evolutionary algorithms for discrete optimization*, ACM Transactions on Evolutionary Learning and Optimization, 1 (2021), str. 1–43.

[78] O. Dürr, B. Sick, oraz E. Murina, *Probabilistic Deep Learning: With Python, Keras and TensorFlow Probability*, Manning Publications, 2020.

[79] J. Earley, *An efficient context-free parsing algorithm*, Communications of the ACM, 13 (1970), str. 94–102.

[80] T. Fawcett, *An introduction to roc analysis*, Pattern recognition letters, 27 (2006), str. 861–874.

[81] R. W. Floyd, *Algorithm 97: shortest path*, Communications of the ACM, 5 (1962), str. 345.

[82] C. Fonlupt, D. Robilliard, oraz P. Preux, *A bit-wise epistasis measure for binary search spaces*, w International Conference on Parallel Problem Solving from Nature, Springer, 1998, str. 47–56.

[83] S. Forrest oraz J. Holland, *The royal road for genetic algorithms: Fitness landscapes and ga performance*, w Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, 1991, str. 245–254.

[84] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, et al., *Noisy networks for exploration*, arXiv preprint arXiv:1706.10295, (2017).

[85] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, oraz Q. Pan, *A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems*, IEEE/CAA Journal of Automatica Sinica, 6 (2019), str. 904–916.

[86] M. Georgiopoulos, J. Huang, oraz G. L. Heileman, *Properties of learning in artmap*, Neural Networks, 7 (1994), str. 495–506.

[87] A. Géron, K. Sawka, oraz G. W. Helion, *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow: pojęcia, techniki i narzędzia służące do tworzenia inteligentnych systemów*, Helion, 2018.

[88] X. Glorot, A. Bordes, oraz Y. Bengio, *Deep sparse rectifier neural networks*, w Proceedings of the fourteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2011, str. 315–323.

[89] F. Glover, *Future paths for integer programming and links to artificial intelligence*, Computers & operations research, 13 (1986), str. 533–549.

[90] ——, *Tabu search—part I*, ORSA Journal on computing, 1 (1989), str. 190–206.

[91] ——, *Tabu search—part II*, ORSA Journal on computing, 2 (1990), str. 4–32.

[92] A. Gnatowski oraz T. Nieżyński, *On estimating lon-based measures in cyclic assignment problem in non-permutational flow shop scheduling problem*, w Modelling and Performance Analysis of Cyclic Systems, Springer, 2020, str. 63–84.

[93] ——, *A parallel algorithm for scheduling a two-machine robotic cell in bicycle frame welding process*, Applied Sciences, 11 (2021).

[94] I. Goodfellow, Y. Bengio, oraz A. Courville, *Deep Learning*, MIT Press, 2016. http://www.deeplearningbook.org.

[95] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, oraz Y. Bengio, *Generative adversarial nets*, Advances in neural information processing systems, 27 (2014).

[96] R. L. Graham, E. L. Lawler, J. K. Lenstra, oraz A. R. Kan, *Optimization and approximation in deterministic sequencing*

and scheduling: a survey, w Annals of discrete mathematics, vol. 5, Elsevier, 1979, str. 287–326.

[97] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, oraz J. Schmidhuber, *A novel connectionist system for unconstrained handwriting recognition*, IEEE transactions on pattern analysis and machine intelligence, 31 (2008), str. 855–868.

[98] A. Graves, G. Wayne, oraz I. Danihelka, *Neural turing machines*, arXiv preprint arXiv:1410.5401, (2014).

[99] H. Gultekin, B. Coban, oraz V. E. Akhlaghi, *Cyclic scheduling of parts and robot moves in m-machine robotic cells*, Computers & Operations Research, 90 (2018), str. 161–172.

[100] P. Guo, W. Cheng, oraz Y. Wang, *Parallel machine scheduling with step-deteriorating jobs and setup times by a hybrid discrete cuckoo search algorithm*, Engineering Optimization, 47 (2015), str. 1564–1585.

[101] T. Guo, C. Han, S. Tang, oraz M. Ding, *Solving combinatorial problems with machine learning methods*, w Nonlinear Combinatorial Optimization, Springer, 2019, str. 207–229.

[102] N. G. Hall, H. Kamoun, oraz C. Sriskandarajah, *Scheduling in robotic cells: classification, two and three machine cells*, Operations Research, 45 (1997), str. 421–439.

[103] M. Hasegawa, T. Ikeguchi, oraz K. Aihara, *Combination of chaotic neurodynamics with the 2-opt algorithm to solve traveling salesman problems*, Physical Review Letters, 79 (1997), str. 2344.

[104] H. v. Hasselt, A. Guez, oraz D. Silver, *Deep reinforcement learning with double q-learning*, w Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, AAAI Press, 2016, str. 2094–2100.

[105] K. He, X. Zhang, S. Ren, oraz J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, w Proceedings of the IEEE international conference on computer vision, 2015, str. 1026–1034.

[106] ——, *Deep residual learning for image recognition*, w Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, str. 770–778.

[107] ——, *Identity mappings in deep residual networks*, w Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14, Springer, 2016, str. 630–645.

[108] L. Hernando, F. Daolio, N. Veerapen, oraz G. Ochoa, *Local optima networks of the permutation flowshop scheduling problem: Makespan vs. total flow time*, w 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, str. 1964–1971.

[109] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, oraz D. Silver, *Rainbow: Combining improvements in deep reinforcement learning*, w Thirty-second AAAI conference on artificial intelligence, 2018.

[110] G. E. Hinton, *A practical guide to training restricted boltzmann machines*, w Neural Networks: Tricks of the Trade: Second Edition, Springer, 2012, str. 599–619.

[111] G. E. Hinton, S. Osindero, oraz Y.-W. Teh, *A fast learning algorithm for deep belief nets*, Neural computation, 18 (2006), str. 1527–1554.

[112] S. Hochreiter oraz J. Schmidhuber, *Long short-term memory*, Neural computation, 9 (1997), str. 1735–1780.

[113] H. Holland John, *Adaptation in natural and artificial systems*, Ann Arbor: University of Michigan Press, (1975).

[114] J. J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities.*, Proceedings of the national academy of sciences, 79 (1982), str. 2554–2558.

[115] J. J. Hopfield oraz D. W. Tank, *"neural" computation of decisions in optimization problems*, Biological cybernetics, 52 (1985), str. 141–152.

[116] W. Hordijk, *A measure of landscapes*, Evolutionary computation, 4 (1996), str. 335–360.

[117] W. Hordijk oraz P. F. Stadler, *Amplitude spectra of fitness landscapes*, Advances in Complex Systems, 1 (1998), str. 39–66.

[118] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, oraz H. Adam, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, arXiv preprint arXiv:1704.04861, (2017).

[119] Y. Hua, J. Guo, oraz H. Zhao, *Deep belief networks and deep learning*, w Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things, IEEE, 2015, str. 1–4.

[120] J. Huang, M. Georgiopoulos, oraz G. L. Heileman, *Fuzzy art properties*, Neural Networks, 8 (1995), str. 203–213.

[121] G. Hulten, W. Fenrich, W. Sikorski, P. Kwiatkowski, W. S. Press, oraz W. N. PWN., *Budowanie systemów inteligentnych: przewodnik po inżynierii uczenia się maszyn*, PWN, 2020.

[122] J. Humeau, A. Liefooghe, E.-G. Talbi, oraz S. Verel, *Paradiseo-mo: From fitness landscape analysis to efficient local search algorithms*, Journal of Heuristics, 19 (2013), str. 881–915.

[123] D. Iclanzan, F. Daolio, oraz M. Tomassini, *Data-driven local optima network characterization of QAPLIB instances*, w Proceedings of the 2014 conference on Genetic and evolutionary computation—GECCO'14, New York, New York, USA, 2014, ACM Press, str. 453–460.

[124] E. Ipek, O. Mutlu, J. F. Martínez, oraz R. Caruana, *Self-optimizing memory controllers: A reinforcement learning approach*, ACM SIGARCH Computer Architecture News, 36 (2008), str. 39–50.

[125] A. Irpan, *Deep reinforcement learning doesn't work yet.* https://www.alexirpan.com/2018/02/14/rl-hard.html, 2018.

[126] E. Izhikevich, *Simple model of spiking neurons*, IEEE Transactions on neural networks, 14 (2003), str. 1569–1572.

[127] N. D. Jana, J. Sil, oraz S. Das, *Selection of appropriate metaheuristic algorithms for protein structure prediction in AB off-lattice model: a perspective from fitness landscape analysis*, Information Sciences, 391–392 (2017), str. 28–64.

[128] H.-D. Jin, K.-S. Leung, M.-L. Wong, oraz Z.-B. Xu, *An efficient self-organizing map designed by genetic algorithms for the traveling salesman problem*, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 33 (2003), str. 877–888.

[129] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon, TaoXie, K. Michael, J. Fang, imyhxy, Lorna, C. Wong, Z. Yifu, A. V, D. Montes, Z. Wang, C. Fati, J. Nadar, Laughing, UnglvKitDe, tkianai, yxNONG, P. Skalski, A. Hogan, M. Strobel, M. Jain, L. Mammana, oraz xylieong, *ultralytics/yolov5: v6.2 - YOLOv5 Classification Models, Apple M1, Reproducibility, ClearML and Deci.ai integrations*, sierpień 2022. https://github.com/ultralytics/yolov5.

[130] T. Jones, S. Forrest, et al., *Fitness distance correlation as a measure of problem difficulty for genetic algorithms.*, w ICGA, vol. 95, 1995, str. 184–192.

[131] C. K. Joshi, T. Laurent, oraz X. Bresson, *An efficient graph convolutional network technique for the travelling salesman problem*, arXiv preprint arXiv:1906.01227, (2019).

[132] G. Joya, M. Atencia, oraz F. Sandoval, *Hopfield neural networks for optimization: study of the different dynamics*, Neurocomputing, 43 (2002), str. 219–237.

[133] Y. Kaempfer oraz L. Wolf, *Learning the multiple traveling salesmen problem with permutation invariant pooling networks*, arXiv preprint arXiv:1803.09621, (2018).

[134] T. Kampmeyer, *Cyclic Scheduling Problems*, PhD thesis, Universitat Osnabrück, Osnabrück, Niemcy, lipiec 2006.

[135] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, oraz D. Amodei, *Scaling laws for neural language models*, arXiv preprint arXiv:2001.08361, (2020).

[136] D. Karaboga oraz B. Basturk, *On the performance of artificial bee colony (abc) algorithm*, Applied soft computing, 8 (2008), str. 687–697.

[137] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, oraz E.-G. Talbi, *Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art*, European Journal of Operational Research, 296 (2022), str. 393–422.

[138] T. Karras, S. Laine, oraz T. Aila, *A style-based generator architecture for generative adversarial networks*, w Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, str. 4401–4410.

[139] S. Kaufman, S. Rosset, C. Perlich, oraz O. Stitelman, *Leakage in data mining: Formulation, detection, and avoidance*, ACM Transactions on Knowledge Discovery from Data (TKDD), 6 (2012), str. 1–21.

[140] J. Kennedy oraz R. Eberhart, *Particle swarm optimization*, w Proceedings of ICNN'95-international conference on neural networks, vol. 4, IEEE, 1995, str. 1942–1948.

[141] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, oraz L. Song, *Learning combinatorial optimization algorithms over graphs*, Advances in neural information processing systems, 30 (2017).

[142] M. Kimura, *The neutral theory of molecular evolution*, Cambridge University Press, 1983.

[143] D. P. Kingma oraz J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

[144] S. Kirkpatrick, C. D. Gelatt, oraz M. P. Vecchi, *Optimization by simulated annealing*, science, 220 (1983), str. 671–680.

[145] G. Kizilateş oraz F. Nuriyeva, *On the nearest neighbor algorithms for the traveling salesman problem*, w Advances in Computational Science, Engineering and Information Technology, Springer, 2013, str. 111–118.

[146] K. Kluwak oraz T. Niżyński, *Gait classification using lstm networks for tagging system*, w 2020 IEEE 15th International Conference of System of Systems Engineering (SoSE), IEEE, 2020, str. 295–300.

[147] R. Knosala oraz G. Ćwikła, *Zastosowania metod sztucznej inteligencji w inżynierii produkcji*, Wydawnictwa Naukowo-Techniczne, 2002.

[148] J. Knox oraz F. Glover, *Comparative testing of the traveling salesman problem heuristics derived from tabu search, genetic algorithms and simulated annealing*, tech. restr., Technical Report, Center for Applied Artificial Intelligence, 1989.

[149] N. Kohl oraz P. Stone, *Policy gradient reinforcement learning for fast quadrupedal locomotion*, w IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004, vol. 3, IEEE, 2004, str. 2619–2624.

[150] T. Kohonen, *The self-organizing map*, Proceedings of the IEEE, 78 (1990), str. 1464–1480.

[151] F. Kolahan, A. Tavakoli, B. Tajdin, oraz M. Hosayni, *Analysis of neighborhood generation and move selection strategies on the performance of tabu search*, w 6th WSEAS International Conference on Applied Computer Science, 2006.

[152] W. Kool, H. Van Hoof, oraz M. Welling, *Attention, learn to solve routing problems!*, arXiv preprint arXiv:1803.08475, (2018).

[153] L. Kotthoff, *Algorithm selection for combinatorial search problems: A survey*, w Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach, C. Bessiere, L. De Raedt, L. Kotthoff, S. Nijssen, B. O'Sullivan, oraz D. Pedreschi, ed., Springer International Publishing, Cham, 2016, str. 149–190.

[154] J. R. Koza, *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*, vol. 34, Stanford University, Department of Computer Science Stanford, CA, 1990.

[155] J. R. Koza et al., *Genetic programming II*, vol. 17, MIT press Cambridge, 1994.

[156] A. Krizhevsky, I. Sutskever, oraz G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems, 25 (2012).

[157] M. Lai, *Giraffe: Using deep reinforcement learning to play chess*, arXiv preprint arXiv:1509.01549, (2015).

[158] X. Lai, J.-K. Hao, oraz D. Yue, *Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem*, European Journal of Operational Research, 274 (2019), str. 35–48.

[159] M. E. Lalami oraz D. El-Baz, *Gpu implementation of the branch and bound method for knapsack problems*, w 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IEEE, 2012, str. 1769–1777.

[160] E. L. Lawler oraz D. E. Wood, *Branch-and-bound methods: A survey*, Operations research, 14 (1966), str. 699–719.

[161] J. Lee, D. Crighton, oraz M. Ablowitz, *A First Course in Combinatorial Optimization*, Cambridge Texts in Applied Mathematics, Cambridge University Press, 2004.

[162] K. Lee, *Inteligencja sztuczna, rewolucja prawdziwa: Chiny, USA i przyszłość świata*, Media Rodzina, 2019.

[163] K. Lee, O. Firat, A. Agarwal, C. Fannjiang, oraz D. Sussillo, *Hallucinations in neural machine translation*, 2018. https://openreview.net/forum?id=SkxJ-309FQ.

[164] K. Lee oraz C. Qiufan, *Sztuczna inteligencja 2041*, Media Rodzina, 2022.

[165] P. Levine, *Eugenics: a very short introduction*, Oxford University Press, 2016.

[166] E. Levner, V. Kats, D. A. L. De Pablo, oraz T. E. Cheng, *Complexity of cyclic scheduling problems: A state-of-the-art survey*, Computers & Industrial Engineering, 59 (2010), str. 352–361.

[167] L. Li, W. Chu, J. Langford, oraz R. E. Schapire, *A contextual-bandit approach to personalized news article recommendation*, w Proceedings of the 19th international conference on World wide web, 2010, str. 661–670.

[168] Y. Li oraz T. Yang, *Word embedding for understanding natural language: a survey*, w Guide to big data applications, Springer, 2018, str. 83–104.

[169] J. Liang, B. Qu, P. Suganthan, oraz Q. Chen, *Problem definitions and evaluation criteria for the cec 2015 competition on learning-based real-parameter single objective optimization*, Technical

Report201411A, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, 29 (2014), str. 625–640.

[170] T. Lin, Y. Wang, X. Liu, oraz X. Qiu, *A survey of transformers*, AI Open, (2022).

[171] J. D. Little, K. G. Murty, D. W. Sweeney, oraz C. Karel, *An algorithm for the traveling salesman problem*, Operations research, 11 (1963), str. 972–989.

[172] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, oraz V. Stoyanov, *Roberta: A robustly optimized bert pretraining approach*, arXiv preprint arXiv:1907.11692, (2019).

[173] M. A. Lones, *Metaheuristics in nature-inspired algorithms*, w Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, 2014.

[174] B. T. Lowerre, *The harpy speech recognition system.*, Carnegie Mellon University, 1976.

[175] G. Lu, J. Li, oraz X. Yao, *Fitness-probability cloud and a measure of problem hardness for evolutionary algorithms*, w European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, 2011, str. 108–117.

[176] H. Lu, J. Shi, Z. Fei, Q. Zhou, oraz K. Mao, *Measures in the time and frequency domains for fitness landscape analysis of dynamic optimization problems*, Applied Soft Computing, 51 (2017), str. 192–208.

[177] M. Lunacek oraz D. Whitley, *The dispersion metric and the cma evolution strategy*, w Proceedings of the 8th annual conference on Genetic and evolutionary computation, 2006, str. 477–484.

[178] B. Manderick, *The genetic algorithm and the structure of fitness landscape*, w 4th ICGA, 1991, str. 143–150.

[179] M.-E. Marmion, A. Blot, L. Jourdan, oraz C. Dhaenens, *Neutrality in the graph coloring problem*, w International Conference on Learning and Intelligent Optimization, Springer, 2013, str. 125–130.

[180] P. Marrow, M. Heath, oraz I. I. Re, *Evolvability: Evolution, computation, biology*, w Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program (GECCO-99 Workshop on Evolvability), 1999, str. 30–33.

[181] J. F. Martinez oraz E. Ipek, *Dynamic multicore resource management: A machine learning approach*, IEEE micro, 29 (2009), str. 8–17.

[182] D. C. Mattfeld, C. Bierwirth, oraz H. Kopfer, *A search space analysis of the job shop scheduling problem*, Annals of Operations Research, 86 (1999), str. 441–453.

[183] N. Mazyavkina, S. Sviridov, S. Ivanov, oraz E. Burnaev, *Reinforcement learning for combinatorial optimization: A survey*, Computers & Operations Research, 134 (2021).

[184] P. Merz oraz B. Freisleben, *Fitness landscape analysis and memetic algorithms for the quadratic assignment problem*, IEEE transactions on evolutionary computation, 4 (2000), str. 337–352.

[185] Z. Michalewicz, *Algorytmy genetyczne + struktury danych = programy ewolucyjne*, Wydawnictwa Naukowo-Techniczne, 1996.

[186] T. Mikolov, K. Chen, G. Corrado, oraz J. Dean, *Efficient estimation of word representations in vector space*, arXiv preprint arXiv:1301.3781, (2013).

[187] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, oraz M. Riedmiller, *Playing atari with deep reinforcement learning*, arXiv preprint arXiv:1312.5602, (2013).

[188] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., *Human-level control through deep reinforcement learning*, Nature, 518 (2015), str. 529–533.

[189] M. Morales, *Grokking deep reinforcement learning*, Manning Publications, 2020.

[190] F. Movahedi, J. L. Coyle, oraz E. Sejdić, *Deep belief networks for electroencephalography: A review of recent contributions and future outlooks*, IEEE journal of biomedical and health informatics, 22 (2017), str. 642–652.

[191] B. Naudts oraz L. Kallel, *A comparison of predictive measures of problem difficulty in evolutionary algorithms*, IEEE Transactions on Evolutionary Computation, 4 (2000), str. 1–15.

[192] B. Naudts, D. Suys, oraz A. Verschoren, *Epistasis as a basic concept in formal landscape analysis*, w Proceedings of the 7th International Conference on Genetic Algorithms, East Lansing, MI, USA, July 19-23, 1997, Morgan Kaufmann, 1997, str. 65–72.

[193] M. Nawaz, E. E. Enscore Jr, oraz I. Ham, *A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem*, Omega, 11 (1983), str. 91–95.

[194] M. Nazari, A. Oroojlooy, L. Snyder, oraz M. Takác, *Reinforcement learning for solving the vehicle routing problem*, Advances in neural information processing systems, 31 (2018).

[195] M. E. Newman, *Mixing patterns in networks*, Physical review E, 67 (2003).

[196] ——, *The structure and function of complex networks*, SIAM review, 45 (2003), str. 167–256.

[197] A. Nowak, S. Villar, A. S. Bandeira, oraz J. Bruna, *Revised note on learning quadratic assignment with graph neural networks*, w 2018 IEEE Data Science Workshop (DSW), IEEE, 2018, str. 1–5.

[198] E. Nowicki oraz C. Smutnicki, *A fast taboo search algorithm for the job shop problem*, Management Science, 42 (1996), str. 797–813.

[199] E. Nowicki oraz C. Smutnicki, *An advanced tabu search algorithm for the job shop problem*, Journal of Scheduling, 8 (2005), str. 145–159.

[200] G. Ochoa oraz N. Veerapen, *Additional Dimensions to the Study of Funnels in Combinatorial Landscapes*, w Proceedings of the 2016 on Genetic and Evolutionary Computation Conference—GECCO'16, New York, New York, USA, 2016, ACM Press, str. 373–380.

[201] ——, *Mapping the global structure of TSP fitness landscapes*, Journal of Heuristics, (2017), str. 1–30.

[202] G. Ochoa, S. Verel, F. Daolio, oraz M. Tomassini, *Local Optima Networks: A New Model of Combinatorial Fitness Landscapes*, w Recent Advances in the Theory and Application of Fitness

Landscapes, H. Richter oraz A. Engelbrecht, ed., Springer Berlin Heidelberg, 2014, ch. 9, str. 233–262.

[203] G. Ochoa, S. Verel, oraz M. Tomassini, *First-improvement vs. best-improvement local optima networks of nk landscapes*, w International Conference on Parallel Problem Solving from Nature, Springer, 2010, str. 104–113.

[204] OpenAI, *Gpt-4 technical report*, arXiv preprint arXiv:2303.08774, (2023).

[205] O. Ozturk, M. A. Begen, oraz G. S. Zaric, *A branch and bound algorithm for scheduling unit size jobs on parallel batching machines to minimize makespan*, International Journal of Production Research, 55 (2017), str. 1815–1831.

[206] T. P. Peixoto, *The graph-tool python library*, figshare, (2014).

[207] T. Pilot oraz R. Knosala, *The application of neural networks in group technology*, Journal of Materials Processing Technology, 78 (1998), str. 150–155.

[208] E. Pitzer oraz M. Affenzeller, *A comprehensive survey on fitness landscape analysis*, Recent advances in intelligent engineering systems, (2012), str. 161–191.

[209] E. Pitzer, A. Beham, oraz M. Affenzeller, *Automatic Algorithm Selection for the Quadratic Assignment Problem Using Fitness Landscape Analysis*, w Proceedings of the 13th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP'13, Springer-Verlag, Berlin, Heidelberg, 2013, str. 109–120.

[210] D. M. Powers, *Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation*, arXiv preprint arXiv:2010.16061, (2020).

[211] R. Pugliese, S. Regondi, oraz R. Marini, *Machine learning-based approach: Global trends, research directions, and regulatory standpoints*, Data Science and Management, 4 (2021), str. 19–29.

[212] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, oraz I. Sutskever, *Zero-shot text-to-image generation*, 2021.

[213] H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, T. Adler, L. Gruber, M. Holzleitner, M. Pavlović, G. K. Sandve, et al., *Hopfield networks is all you need*, arXiv preprint arXiv:2008.02217, (2020).

[214] S. Ramzan, M. M. Iqbal, oraz T. Kalsum, *Text-to-image generation using deep learning*, Engineering Proceedings, 20 (2022), str. 16.

[215] A. Rand, *Atlas zbuntowany*, Zysk i S-ka Wydawnictwo, 2017.

[216] ——, *Źródło*, Zysk i S-ka Wydawnictwo, 2017.

[217] A. Rao oraz T. Jelvis, *Foundations of reinforcement learning with applications in finance*, CRC Press, 2022.

[218] P. Rebentrost, T. R. Bromley, C. Weedbrook, oraz S. Lloyd, *Quantum hopfield neural network*, Physical Review A, 98 (2018).

[219] J. Redmon, S. Divvala, R. Girshick, oraz A. Farhadi, *You only look once: Unified, real-time object detection*, w Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, str. 779–788.

[220] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, oraz H. Lee, *Generative adversarial text to image synthesis*, w International conference on machine learning, PMLR, 2016.

[221] C. M. Reidys oraz P. F. Stadler, *Neutrality in fitness landscapes*, Applied Mathematics and Computation, 117 (2001), str. 321–350.

[222] G. Reinelt, *Tsplib—a traveling salesman problem library*, ORSA journal on computing, 3 (1991), str. 376–384.

[223] J. Ren, C. Ye, oraz Y. Li, *A new solution to distributed permutation flow shop scheduling problem based on nash q-learning*, Management, 13 (2021), str. 136–146.

[224] I. Ribas, R. Leisten, oraz J. M. Framiñan, *Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective*, Computers & Operations Research, 37 (2010), str. 1439–1454.

[225] J. R. Rice, *The Algorithm Selection Problem*, Advances in Computers, 15 (1976), str. 65–118.

[226] O. Ronneberger, P. Fischer, oraz T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, w Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, Springer, 2015, str. 234–241.

[227] H. Rosé, W. Ebeling, oraz T. Asselmeyer, *The density of states—a measure of the difficulty of optimisation problems*, w International Conference on Parallel Problem Solving from Nature, Springer, 1996, str. 208–217.

[228] R. Ruiz oraz J. A. Vázquez-Rodríguez, *The hybrid flow shop scheduling problem*, European journal of operational research, 205 (2010), str. 1–18.

[229] D. E. Rumelhart, G. E. Hinton, oraz R. J. Williams, *Learning internal representations by error propagation*, tech. restr., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[230] ——, *Learning representations by back-propagating errors*, Nature, 323 (1986), str. 533–536.

[231] G. A. Rummery oraz M. Niranjan, *On-line Q-learning using connectionist systems*, vol. 37, University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[232] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., *Imagenet large scale visual recognition challenge*, International journal of computer vision, 115 (2015), str. 211–252.

[233] S. Russell oraz P. Norvig, *Artificial Intelligence: A Modern Approach*, CreateSpace Independent Publishing Platform, 2016.

[234] S. Russell oraz P. Norvig, *Sztuczna inteligencja: nowe spojrzenie*, vol. 2, Helion, 4 ed., 2023.

[235] O. Sagi oraz L. Rokach, *Ensemble learning: A survey*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 8 (2018).

[236] H. Sak, A. Senior, oraz F. Beaufays, *Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition*, arXiv preprint arxiv:1402.1128, (2015).

[237] R. Salakhutdinov oraz G. Hinton, *Semantic hashing*, International Journal of Approximate Reasoning, 50 (2009), str. 969–978.

[238] S. Salhi, *Defining tabu list size and aspiration criterion within tabu search methods*, Computers & Operations Research, 29 (2002), str. 67–86.

[239] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, oraz G. Monfardini, *The graph neural network model*, IEEE transactions on neural networks, 20 (2008), str. 61–80.

[240] T. Schaul, J. Quan, I. Antonoglou, oraz D. Silver, *Prioritized experience replay*, arXiv preprint arXiv:1511.05952, (2015).

[241] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, oraz O. Klimov, *Proximal policy optimization algorithms*, arXiv preprint arXiv:1707.06347, (2017).

[242] C. J. Schuster, *No-wait job shop scheduling: Tabu search and complexity of subproblems*, Mathematical Methods of Operations Research, 63 (2006), str. 473–491.

[243] H.-P. Schwefel, *Numerical optimization of computer models*, John Wiley & Sons, Inc., 1981.

[244] ——, *Evolution strategies: A family of non-linear optimization techniques based on imitating some principles of organic evolution*, Annals of Operations Research, 1 (1984), str. 165–167.

[245] S. P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, oraz W. Kubiak, *Sequencing of parts and robot moves in a robotic cell*, International Journal of Flexible Manufacturing Systems, 4 (1992), str. 331–358.

[246] A. Seyyedabbasi, R. Aliyev, F. Kiani, M. U. Gulle, H. Basyildiz, oraz M. A. Shah, *Hybrid algorithms based on combining reinforcement learning and metaheuristic methods to solve global optimization problems*, Knowledge-Based Systems, 223 (2021).

[247] S. Shirakawa oraz T. Nagao, *Bag of local landscape features for fitness landscape analysis*, Soft Computing, 20 (2016), str. 3787–3802.

[248] C. Shorten oraz T. M. Khoshgoftaar, *A survey on image data augmentation for deep learning*, Journal of big data, 6 (2019), str. 1–48.

[249] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., *Mastering the game of go with deep neural networks and tree search*, Nature, 529 (2016), str. 484–489.

[250] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*, arXiv preprint arXiv:1712.01815, (2017).

[251] ——, *A general reinforcement learning algorithm that masters chess, shogi, and go through self-play*, Science, 362 (2018), str. 1140–1144.

[252] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al., *Mastering the game of go without human knowledge*, Nature, 550 (2017), str. 354–359.

[253] D. Simon, *Evolutionary Optimization Algorithms*, Wiley, 2013.

[254] T. Smith, P. Husbands, oraz M. O'Shea, *Fitness landscapes and evolvability*, Evolutionary computation, 10 (2002), str. 1–34.

[255] K. A. Smith-Miles, *Neural networks for prediction and classification*, w Encyclopaedia of Data Warehousing and Mining, J. Wang, ed., Information Science Publishing, 2006, str. 865–869.

[256] ——, *Towards insightful algorithm selection for optimisation using meta-learning concepts*, w 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), IEEE, jun 2008, str. 4118–4124.

[257] C. Smutnicki, *Algorytmy szeregowania*, Akademicka Oficyna Wydawnicza EXIT, 2002.

[258] ——, *Algorytmy szeregowania zadań*, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, (2012).

[259] M. Solimanpur, P. Vrat, oraz R. Shankar, *A neuro-tabu search heuristic for the flow shop scheduling problem*, Computers & Operations Research, 31 (2004), str. 2151–2164.

[260] K. Sörensen, *Metaheuristics—the metaphor exposed*, International Transactions in Operational Research, 22 (2015), str. 3–18.

[261] P. F. Stadler, *Landscapes and their correlation functions*, Journal of Mathematical chemistry, 20 (1996), str. 1–45.

[262] P. F. Stadler, *Fitness landscapes*, w Biological Evolution and Statistical Physics, M. Lässig oraz A. Valleriani, ed., Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, str. 183–204.

[263] J. Strelau, A. Matczak, W. N. Scholar, B. Zawadzki, oraz S. W. P. Społecznej, *Różnice indywidualne: historia, determinanty, zastosowania*, Wydawnictwo Naukowe Scholar, 2015.

[264] Z. Sun, U. Benlic, M. Li, oraz Q. Wu, *Reinforcement learning based tabu search for the minimum load coloring problem*, Computers & Operations Research, 143 (2022).

[265] I. Sutskever, O. Vinyals, oraz Q. V. Le, *Sequence to sequence learning with neural networks*, Advances in neural information processing systems, 27 (2014).

[266] R. Sutton oraz A. Barto, *Reinforcement Learning, second edition: An Introduction*, Adaptive Computation and Machine Learning series, MIT Press, 2018.

[267] R. S. Sutton, *Learning to predict by the methods of temporal differences*, Machine learning, 3 (1988), str. 9–44.

[268] R. Tadeusiewicz, B. Leper, B. Borowik, oraz T. Gąciarz, *Odkrywanie właściwości sieci neuronowych: przy użyciu programów w języku C#*, Wydawnictwa PAU, 2007.

[269] E. Taillard, *Some efficient heuristic methods for the flow shop sequencing problem*, European journal of Operational research, 47 (1990), str. 65–74.

[270] ——, *Benchmarks for basic scheduling problems*, european journal of operational research, 64 (1993), str. 278–285.

[271] E.-G. Talbi, *Metaheuristics: from design to implementation*, vol. 74, John Wiley & Sons, 2009.

[272] A.-H. Tan, *Adaptive resonance associative map*, Neural Networks, 8 (1995), str. 437–446.

[273] J. Tavares, F. B. Pereira, oraz E. Costa, *Multidimensional knapsack problem: A fitness landscape analysis*, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 38 (2008), str. 604–616.

[274] Y. Tay, M. Dehghani, D. Bahri, oraz D. Metzler, *Efficient transformers: A survey*, arXiv preprint arXiv:2009.06732, (2022).

[275] A. Tayal, S. P. Singh, et al., *Analysis of simulated annealing cooling schemas for design of optimal flexible layout under uncertain dynamic product demand*, International Journal of Operation Research, (2016).

[276] G. Tesauro et al., *Temporal difference learning and td-gammon*, Communications of the ACM, 38 (1995), str. 58–68.

[277] A. Thakur, *Approaching (Almost) Any Machine Learning Problem*, Abhishek Thakur, 2020.

[278] G. Theocharous, P. S. Thomas, oraz M. Ghavamzadeh, *Ad recommendation systems for life-time value optimization*, w Proceedings of the 24th international conference on world wide web, 2015.

[279] S. L. Thomson, G. Ochoa, F. Daolio, oraz N. Veerapen, *The effect of landscape funnels in QAPLIB instances*, w Proceedings of the Genetic and Evolutionary Computation Conference Companion on—GECCO'17, New York, New York, USA, 2017, ACM Press.

[280] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, et al., *Lamda: Language models for dialog applications*, arXiv preprint arXiv:2201.08239, (2022).

[281] K. Toch, M. Buczek, oraz M. Labocha-Derkowska, *Epistaza: teoria, sposób badania, znaczenie*, Kosmos, 68 (2019).

[282] M. Tomassini, S. Verel, oraz G. Ochoa, *Complex-network analysis of combinatorial spaces: The nk landscape case*, Physical Review E, 78 (2008).

[283] P. Toth oraz D. Vigo, *Branch-and-bound algorithms for the capacitated vrp*, w The vehicle routing problem, SIAM, 2002, str. 29–51.

[284] S. Tsubakitani oraz J. R. Evans, *Optimizing tabu list size for the traveling salesman problem*, Computers & Operations Research, 25 (1998), str. 91–97.

[285] P. D. Turney, *Increasing evolvability considered as a large-scale trend in evolution*, arXiv preprint cs/0212042, (2002).

[286] P. Vaishnav, N. Choudhary, oraz K. Jain, *Traveling Salesman Problem Using Genetic Algorithm: A Survey*, International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 3 (2017), str. 105–108.

[287] L. Van der Maaten oraz G. Hinton, *Visualizing data using t-sne.*, Journal of machine learning research, 9 (2008).

[288] H. Van Seijen, H. Van Hasselt, S. Whiteson, oraz M. Wiering, *A theoretical and empirical analysis of expected sarsa*, w 2009 ieee symposium on adaptive dynamic programming and reinforcement learning, IEEE, 2009, str. 177–184.

[289] V. K. Vassilev, T. C. Fogarty, oraz J. F. Miller, *Information characteristics and the structure of landscapes*, Evolutionary computation, 8 (2000), str. 31–60.

[290] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, oraz I. Polosukhin, *Attention is all you need*, Advances in neural information processing systems, 30 (2017).

[291] S. Verel, F. Daolio, G. Ochoa, oraz M. Tomassini, *Local optima networks with escape edges*, w International Conference on Artificial Evolution (Evolution Artificielle), Springer, 2011, str. 49–60.

[292] S. Verel, G. Ochoa, oraz M. Tomassini, *Local optima networks of nk landscapes with neutrality*, IEEE Transactions on Evolutionary Computation, 15 (2010), str. 783–797.

[293] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al., *Grandmaster level in starcraft ii using multi-agent reinforcement learning*, Nature, 575 (2019), str. 350–354.

[294] O. Vinyals, M. Fortunato, oraz N. Jaitly, *Pointer networks*, Advances in neural information processing systems, 28 (2015).

[295] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, oraz G. Hinton, *Grammar as a foreign language*, Advances in neural information processing systems, 28 (2015).

[296] G. P. Wagner oraz L. Altenberg, *Perspective: complex adaptations and the evolution of evolvability*, Evolution, 50 (1996), str. 967–976.

[297] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, oraz N. Freitas, *Dueling network architectures for deep reinforcement learning*, w International conference on machine learning, PMLR, 2016.

[298] C. J. Watkins oraz P. Dayan, *Q-learning*, Machine learning, 8 (1992), str. 279–292.

[299] C. J. C. H. Watkins, *Learning from delayed rewards*, PhD thesis, University of Cambridge, Cambridge, United Kingdom, 1989.

[300] E. Weinberger, *Correlated and uncorrelated fitness landscapes and how to tell the difference*, Biological cybernetics, 63 (1990), str. 325–336.

[301] J. Weston, S. Chopra, oraz A. Bordes, *Memory networks*, arXiv preprint arXiv:1410.3916, (2014).

[302] D. Whitley, R. B. Heckendorn, oraz S. Stevens, *Hyperplane ranking, nonlinearity and the simple genetic algorithm*, Information Sciences, 156 (2003), str. 123–145.

[303] M. Widrich, B. Schäfl, M. Pavlović, H. Ramsauer, L. Gruber, M. Holzleitner, J. Brandstetter, G. K. Sandve, V. Greiff, S. Hochreiter, et al., *Modern hopfield networks and attention for immune repertoire classification*, Advances in Neural Information Processing Systems, 33 (2020).

[304] R. J. Williams, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Machine learning, 8 (1992), str. 229–256.

[305] Y. Wu, W. Song, Z. Cao, J. Zhang, oraz A. Lim, *Learning improvement heuristics for solving routing problems..*, IEEE transactions on neural networks and learning systems, (2021).

[306] J. Xie, L. Gao, K. Peng, X. Li, oraz H. Li, *Review on flexible job shop scheduling*, IET Collaborative Intelligent Manufacturing, 1 (2019), str. 67–77.

[307] Z. Xing oraz S. Tu, *A graph neural network assisted monte carlo tree search approach to traveling salesman problem*, IEEE Access, 8 (2020), str. 108418–108428.

[308] B. Xu, N. Wang, T. Chen, oraz M. Li, *Empirical evaluation of rectified activations in convolutional network*, arXiv preprint arXiv:1505.00853, (2015).

[309] L. Yang oraz A. Shami, *On hyperparameter optimization of machine learning algorithms: Theory and practice*, arXiv preprint arXiv:2007.15745, (2020).

[310] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, oraz D. N. Metaxas, *Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks*, w Proceedings of the IEEE international conference on computer vision, 2017, str. 5907–5915.

[311] J. Zhang, G. Ding, Y. Zou, S. Qin, oraz J. Fu, *Review of job shop scheduling research and its new perspectives under industry 4.0*, Journal of Intelligent Manufacturing, 30 (2019), str. 1809–1830.

[312] A. Zheng oraz A. Casari, *Feature engineering for machine learning: principles and techniques for data scientists*, O'Reilly Media, Inc., 2018.

[313] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, oraz M. Sun, *Graph neural networks: A review of methods and applications*, AI Open, 1 (2020), str. 57–81.

[314] Z.-H. Zhou, *Machine learning*, Springer Nature Singapore, 2021.

Spis tabel

Spis rysunków

Spis algorytmów