Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Systemy internetowe wspomagania zarządzania

Zastosowanie grafowych sieci neuronowych do problemu wyznaczania tras z ograniczoną ładownością

Dariusz Stalewski
Numer albumu 283771

promotor
dr hab. inż. Mariusz Kaleta

WARSZAWA 2025

Streszczenie

Zastosowanie grafowych sieci neuronowych
do problemu wyznaczania tras z ograniczoną ładownością

Celem pracy jest analiza metod głębokiego uczenia pod kątem przydatności tychże metod do znajdowania rozwiązań problemu wyznaczania tras (CVRP - capacitated vehicle routing problem). Zaproponowane zostały następujące algorytmy: grafowa sieć neuronowa, algorytm zachłanny, przeszukiwanie wiązkowe, przeszukiwanie drzewa Monte-Carlo oraz wiązkowe przeszukiwanie drzewa Monte-Carlo. Metody zostały dostosowane do typu problemu, a ich analiza została przeprowadzona pod wieloma kątami, m.in. czasu wykonywania, jakości otrzymywanych rozwiązań oraz wpływu parametrów charakterystycznych dla algorytmu na otrzymywane wyniki. Wyniki pokazały, że algorytm zachłanny znajduje rozwiązanie w najlepszym czasie, ale najlepsze rozwiązania pod kątem jakości znajduje algorytm przeszukiwania drzewa Monte-Carlo.

Słowa kluczowe: CVRP, grafowa sieć neuronowa, algorytm zachłanny, przeszukiwanie wiązkowe, przeszukiwanie drzewa Monte-Carlo, wiązkowe przeszukiwanie drzewa Monte-Carlo

Summary

Application of Graph Neural Networks to the capacitated vehicle routing problem

Goal of this thesis is an analysis of deep learning methods at an angle of their usefulness in finding solutions for the capacitated vehicle routing problem (CVRP). The following algorithms were proposed: graph neural network, greedy algorithm, beam search, Monte-Carlo tree search and Beam Monte-Carlo tree search. Methods were adjusted to the class of the problem and their analysis has been carried out from multiple angles, including execution time, quality of provided results and influence of characteristic parameters on those algorithms. The results showed that the greedy algorithm finds the solution in the best time, but the best solutions in terms of quality are found by the Monte-Carlo tree search algorithm.

Keywords: CVRP, graph neural network, greedy algorithm, beam search, Monte-Carlo tree search, beam Monte-Carlo tree search

załącznik do zarządzenia nr 28/2016 r.
Rektora PW

Politechnika Warszawska

„załącznik nr 3 do zarządzenia nr 24/2016 Rektora PW

.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta"

Spis treści

1. Wstęp . . . 6

2. Sformułowanie problemu . . . 8

2.1. Klasyczny problem wyznaczania tras . . . 8
2.2. Problem wyznaczania tras z ograniczoną ładownością . . . 9
2.3. Przegląd literatury . . . 10

3. Proponowane algorytmy rozwiązujące problem wyznaczania tras z ograniczoną ładownością . . . 12

3.1. Grafowa sieć neuronowa . . . 12
3.1.1. Czym są grafowe sieci neuronowe? . . . 12
3.1.2. Warstwy grafowej sieci neuronowej . . . 12
3.1.3. Epoki w sieciach neuronowych . . . 14
3.1.4. Funkcja straty . . . 15
3.2. Algorytm zachłanny . . . 18
3.3. Przeszukiwanie wiązkowe . . . 18
3.4. Przeszukiwanie drzewa Monte-Carlo . . . 20
3.4.1. Ogólny opis działania . . . 20
3.4.2. Selekcja . . . 21
3.4.3. Ekspansja . . . 22
3.4.4. Symulacja . . . 22
3.4.5. Propagacja wsteczna . . . 22
3.5. Wiązkowe przeszukiwanie drzewa Monte-Carlo . . . 22

4. Organizacja eksperymentów obliczeniowych . . . 24

4.1. Dane testowe . . . 24
4.2. Metodyka badań . . . 25

5. Wyniki analizy algorytmów . . . 26

5.1. Grafowa sieć neuronowa . . . 26
5.1.1. Funkcja straty . . . 26
5.1.2. Średni czas epoki . . . 32
5.2. Przeszukiwanie wiązkowe oraz algorytm zachłanny . . . 33
5.3. Przeszukiwanie drzewa Monte-Carlo . . . 37
5.4. Wiązkowe przeszukiwanie drzewa Monte-Carlo . . . 44
5.5. Porównanie algorytmów . . . 49

6. Zakończenie . . . 52

Spis zawartości repozytorium . . . 54

Bibliografia 55
Spis rysunków 57
Spis tabel 59
Spis algorytmów 60

1. Wstęp

Problem wyznaczania tras to zagadnienie, które często pojawia się w zagadnieniach logistycznych. Znalezienie planu dostarczenia towaru do określonej liczby odbiorców to jeden z ważniejszych problemów, z jakimi borykają się systemy informatyczne w biznesie. Znalezienie dobrego rozwiązania w krótkim czasie może znacznie zwiększyć zysk ze zrealizowanego zamówienia. W rzeczywistości problem ten wykorzystuje się np. do wyznaczania tras autobusów, tras dostaw kurierskich, tras śmieciarek itp.

Problem wyznaczania tras (VRP - vehicle routing problem) jest rozwinięciem klasycznego problemu komiwojażera (TSP - travelling salesman problem). Problem ten polega na wyznaczeniu pewnej liczby tras przy jednoczesnym zachowaniu ograniczeń wynikających z modelu. Pierwszy model został zaproponowany w 1959 r., kiedy Dantzig i Ramser [1] opracowali matematyczną formułę problemu i algorytm dostarczania paliwa do stacji benzynowych. Do dzisiaj problem ten ewoluował i jest obecnie jednym z najczęściej badanych problemów optymalizacji dyskretnej. Celem klasycznej wersji problemu jest znalezienie optymalnych tras dla wielu pojazdów odwiedzających określone lokalizacje. W przypadku braku jakichkolwiek innych ograniczeń najlepszym rozwiązaniem będzie przypisanie tylko jednego pojazdu do wszystkich lokalizacji i znalezienie dla niego najkrótszej trasy (problem TSP). Bardziej zgodnym z rzeczywistością sposobem definiowania optymalnych tras jest zminimalizowanie długości najdłuższej pojedynczej trasy wśród wszystkich pojazdów. To właściwa definicja, jeśli celem jest jak najszybsze obsłużenie wszystkich lokacji. Rozwiązaniem problemu jest zbiór dozwolonych, spełniających ograniczenia tras rozpoczynających się i kończących się w punkcie startowym. W uproszczonym opisie problemu długość trasy przekładała się na koszt przejazdu. W rzeczywistości istnieje wiele więcej wymagań i ograniczeń, takich jak wielkość floty pojazdów, pojemność pojazdów oraz okien czasowych.

W przypadku wprowadzenia ograniczenia na pojemność pojazdów oraz zapotrzebowania do każdego klienta dostajemy problem wyznaczania tras z ograniczoną ładownością (CVRP - Capacitated Vehicle Routing Problem). Pojazd może obsłużyć tak dużo klientów, jak długo ma wystarczająco dużo zasobów by spełnić wymaganie każdego klienta. Po obsłużeniu punktu, pojazd który go obsłużył musi zaktualizować swoją ładowność. W przypadku braku jakiegokolwiek klienta, który ma mniejsze lub równe zapotrzebowanie co aktualna ładowność pojazdu, pojazd ten musi zawrócić do punktu startowego (zajezdnia). Jak wykazali Lenstry i Rinnooy [2] problem ten należy do NP-trudnych problemów optymalizacyjnych.

Dość często liczba klientów połączona z pozostałymi ograniczeniami nie pozwala na do-

kładne rozwiązanie problemu w rozsądnym czasie. W tych sytuacjach można użyć algorytmów heurystycznych. Taki algorytm daje nam rozwiązanie dopuszczalne, ale nie koniecznie optymalne i bez gwarancji jakości.

Praca podejmuje temat rozwiązywania problemu wyznaczania tras z ograniczoną ładownością. Celem pracy jest zaprojektowanie i zaimplementowanie przybliżonych metod bazujących na głębokim uczeniu rozwiązujących ten problem oraz dokonanie analizy sposobu i jakości ich działania na podstawie przeprowadzonych testów. Praca odpowie również na następujące pytanie badawcze: czy grafowe sieci neuronowe są w stanie przynieść korzyści przy rozwiązywaniu problemu wyznaczania tras z ograniczoną ładownością w porównaniu do innych istniejących metod rozwiązywania tego problemu.

W rozdziale drugim pracy zaprezentowane jest sformułowanie klasycznego i rozszerzonego problemu wyznaczania tras. Opisane są ograniczenia i funkcja celu oraz ich znaczenie. Na końcu rozdziału wymienione są znane sposoby rozwiązywania go.

W trzecim rozdziale pracy przedstawione są metody, które zostały zaprojektowane i zaimplementowane w celu rozwiązania rozpatrywanego problemu wyznaczania tras. Omówione są również najważniejsze cechy wszystkich metod oraz ich parametry wejściowe.

W czwartym rozdziale omówiony jest sposób przeprowadzenia eksperymentów obliczeniowych. Przedstawiony jest sposób wygenerowania danych testowych oraz wyznaczenia parametrów wejściowych algorytmów. Następnie opisana jest metodyka badań.

Rozdział piąty obejmuje eksperymenty obliczeniowe i ich opis. Kolejno prezentowane zostają wyniki wszystkich badań nad algorytmami. Wyniki algorytmów zanalizowane są pod kątem jakości rozwiązania, czasu obliczeń i wpływu parametrów wejściowych na zachowanie algorytmów.

Rozdział szósty zawiera krótkie podsumowanie całej pracy oraz możliwe kierunki rozwoju tego tematu.

2. Sformułowanie problemu

2.1. Klasyczny problem wyznaczania tras

Klasyczny problem wyznaczania tras można opisać następująco. Mamy zbiór klientów/lokacji reprezentowanych przez węzły $\{1, \ldots, n\}$, węzeł startowy zwany w naszym przypadku centrum dystrybucyjnym 0 (nazywany typowo węzłem zerowym), sieć połączeń pomiędzy centrum dystrybucyjnym a klientami/lokacjami oraz flotę identycznych pojazdów $\{1, \ldots, p\}$. Dla każdej pary węzłów (i, j), gdzie $i, j \in \{0, \ldots, n\}$ oraz $i \neq j$, przypisany jest koszt przejazdu c_{ij}. W wariancie rozważanym w niniejszej pracy, czyli w symetrycznym problemie wyznaczania tras (graf nieskierowany), koszty przejazdu są identyczne niezależnie od kierunku jazdy, więc $c_{ij} = c_{ji}$.

W celu rozwiązania problemu minimalizujemy funkcję:

$$minC = \sum_{k=1}^{p} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ijk} \tag{2.1}$$

Przy ograniczeniach:

$$x_{ijk} \in \{0, 1\} \qquad \forall k \in \{1, \ldots, p\}, \quad i, j \in \{0, \ldots, n\} \tag{2.2}$$

$$x_{iik} = 0 \qquad \forall k \in \{1, \ldots, p\}, \quad i \in \{0, \ldots, n\} \tag{2.3}$$

$$\sum_{i=0}^{n} x_{ijk} = \sum_{i=0}^{n} x_{jik} \qquad \forall j \in \{0, \ldots, n\}, \quad k \in \{1, \ldots, p\} \tag{2.4}$$

$$\sum_{k=1}^{p} \sum_{i=0}^{n} x_{ijk} = 1 \qquad \forall j \in \{1, \ldots, n\} \tag{2.5}$$

$$\sum_{j=1}^{n} x_{0jk} = 1 \qquad \forall k \in \{1, \ldots, p\} \tag{2.6}$$

$$\sum_{i \in S, j \notin S} x_{ijk} \geq 2 \qquad S \subset \{1, \ldots, n\}, 2 \leq |S| \leq n - 1 \tag{2.7}$$

Gdzie:

k – pojazd należący do zbioru jednorodnych (identycznych) pojazdów,
i, j – wierzchołki pomiędzy, którymi odbywa się przewóz,
c_{ij} – koszt przewozu pomiędzy wierzchołkami i i j,

x_{ijk} – zmienna binarna określająca, czy pomiędzy wierzchołkami i i j pojazd k wykonuje przewóz,
S – dowolny podzbiór lokacji nie zawierający centrum dystrybucyjnego.

Funkcja celu (2.1) minimalizuje nam sumę kosztów wszystkich przewozów. Ograniczenie (2.2) definiuje nam zmienną binarną x_{ijk}. Ograniczenie (2.3) powoduje, że pojazd nie może pojechać bezpośrednio do punktu z którego wyjeżdża. Kolejne ograniczenie (2.4) wymusza wyruszenie pojazdu z punktu do którego przyjechał (liczba wyjazdów jest równa liczbie przyjazdów do lokacji). Ograniczenie (2.5) mówi, że pojazd odwiedza każdego klienta dokładnie raz. Ograniczenie (2.6) wymusza wyruszenie każdego pojazdu z centrum dystrybucyjnego, wraz z ograniczeniem (2.4) wymusza również powrót każdego pojazdu do centrum dystrybucyjnego. Ostatnie ograniczenie (2.7) gwarantuje, że nie powstaną żadne podtrasy (Subtour Elimination Constraint).

2.2. Problem wyznaczania tras z ograniczoną ładownością

Problem rozpatrywany w tej pracy nie zakłada wykorzystania każdego pojazdu, czyli nie ma ograniczenia (2.6), ma za to dodatkowe założenie, czyli ograniczoną ładowność każdego pojazdu oraz zapotrzebowanie każdego klienta, tzn. każdy klient/lokacja ma swoje zapotrzebowanie na towar q_j oraz każdy pojazd ma określoną ładowność Q. W rozpatrywanym problemie zakładamy że każdy pojazd ma tą samą ładowność, ale różni klienci mogą mieć różne zapotrzebowanie.

Sformułowanie problemu wyznaczania tras z ograniczoną ładownością jest następujące:

$$minC = \sum_{k=1}^{p} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ijk} \tag{2.8}$$

Przy ograniczeniach:

$$\sum_{i=0}^{n} \sum_{j=1}^{n} q_i x_{ijk} \leq Q \qquad \forall k \in \{1, \ldots, p\} \tag{2.9}$$

$$u_j - u_i \geq q_j - Q(1 - x_{ijk}) \qquad \forall i, j \in \{1, \ldots, n\}, \quad i \neq j, \quad k \in \{1, \ldots, p\} \tag{2.10}$$

$$q_j \leq u_j \leq Q \qquad \forall i \in \{1, \ldots, n\} \tag{2.11}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall k \in \{1, \ldots, p\}, \quad i, j \in \{0, \ldots, n\} \tag{2.12}$$

$$x_{iik} = 0 \qquad \forall k \in \{1, \ldots, p\}, \quad i \in \{0, \ldots, n\} \tag{2.13}$$

$$\sum_{i=0}^{n} x_{ijk} = \sum_{i=0}^{n} x_{jik} \qquad \forall j \in \{0, \ldots, n\}, \quad k \in \{1, \ldots, p\} \tag{2.14}$$

$$\sum_{k=1}^{p}\sum_{i=0}^{n} x_{ijk} = 1 \qquad \forall j \in \{1, \ldots, n\} \tag{2.15}$$

Używając dodatkowych oznaczeń:

Q – ładowność każdego pojazdu,
q_j – zapotrzebowanie klienta j,
u_j – pomocnicza zmienna powstrzymująca powstanie podtras.

Ograniczenie (2.9) na podstawie dodatkowych zmiennych q_j oraz Q mówi, że jeden pojazd może obsługiwać klientów/lokacje tak długo jak suma ich zapotrzebowań nie jest większa niż ładowność pojazdu. Ograniczenia (2.10) oraz (2.11) pełnią tą samą funkcję co ograniczenie (2.7) czyli gwarantują, że nie powstaną żadne podtrasy (Subtour Elimination Constraints). Ograniczenie od (2.11) do (2.14) są takie same jak ograniczenia od (2.2) do (2.5).

2.3. Przegląd literatury

Złożoność problemu wyznaczania tras pojazdów badali Lenstry i Rinnooy [2] wykazując, że jest to problem NP-trudny. W literaturze możemy się spotkać z różnymi wariantami tego problemu i podejściami do niego. Algorytmy wstawiania (insertion algorithm) dla tego typu problemu zostały przedstawione przez Salomona w [12] oraz Potvina i Rousseaua w [13]. Stosowane były również algorytmy zachłanne [14] oraz algorytmy przeszukiwania sąsiedztwa z zastosowaniem dywersyfikacji w [15]. W [16], Tan oraz inne osoby w celu rozwiązania problemu wyznaczania tras z oknami czasowymi przedstawili różne podejścia metaheurystyczne takie jak algorytm genetyczny, symulowane wyżarzanie oraz przeszukiwanie tabu. W [10] zastosowany został algorytm mrówkowy. Przegląd algorytmów dokładnych dla różnych wariantów problemu wyznaczania tras został przedstawiony w [3]. Niestety problemem tego typu algorytmów jest długi czas działania dla większych rozmiarów problemów. W przypadku narzędzi rozwiązujących inne problemy NP-trudne zaobserwowano wzrost wydajności i dokładności dzięki zastosowaniu metod neuronowych [7] [8], zatem istnieje rzeczywisty potencjał głębokiego uczenia w celu poprawienia istniejących rozwiązań. W pracy [9] udowodniono, że GNN może naśladować programowanie dynamiczne, które jest podstawową metodą dla narzędzi rozwiązujących problemy kombinatoryczne. Dlatego zastosowanie grafowej sieci neuronowej wydaje się być dobrym wyborem w przypadku rozwiązywania problemu wyznaczania tras. Zastosowanie grafowych sieci neuronowych jest opisane w [4] dla problemu komiwojażera. Ponieważ sama sieć neuronowa generuje jedynie macierz prawdopodobieństw istnienia każdej krawędzi, wynik działania takiej sieci trzeba połączyć z dodatkową metodą wyznaczającą kompletne rozwiązanie. W [4] do końcowego wyznaczenia rozwiązania na podstawie takiej macierzy zastosowany został algorytm przeszukiwania drzewa Monte-Carlo (MCTS). Przeszukiwanie wiązkowe dla problemu wyznaczania tras zostało zastosowane w artykule [5]. Ponieważ algorytm przeszukiwania drzewa Monte-Carlo w pierwotnym zastosowaniu służył do przewidywania najlepszych ruchów w grach dwu osobowych, postanowiłem poszukać innych wariantów tego algorytmu,

niekoniecznie stosowanych do rozwiązywania problemów kombinatorycznych. W artykule [6] autorzy Baier oraz Winands połączyli algorytm przeszukiwania drzewa Monte-Carlo z metodą przeszukiwania wiązkowego dzięki czemu udało się im się znacznie poprawić czas na znalezienie rozwiązania. Warto tutaj zaznaczyć, że Baier oraz Winands badali ten algorytm tylko dla typowych problemów rozwiązywanych przez MCTS, czyli gier. Ponieważ jak już wcześniej wspomniałem grafowa sieć neuronowa musi zostać połączona z dodatkową metodą wyznaczającą możliwe rozwiązanie, w niniejszej pracy uwaga jest skupiona głównie na dwóch algorytmach stosowanych dla innych problemów kombinatorycznych: przeszukiwaniu wiązkowemu oraz przeszukiwaniu drzewa Monte-Carlo. Dodatkowo zbadana zostanie metoda nie stosowana wcześniej dla tego typu problemu, a dokładniej połączenie tych dwóch metod nazywaną wiązkowym przeszukiwaniem drzewa Monte-Carlo.

3. Proponowane algorytmy rozwiązujące problem wyznaczania tras z ograniczoną ładownością

3.1. Grafowa sieć neuronowa

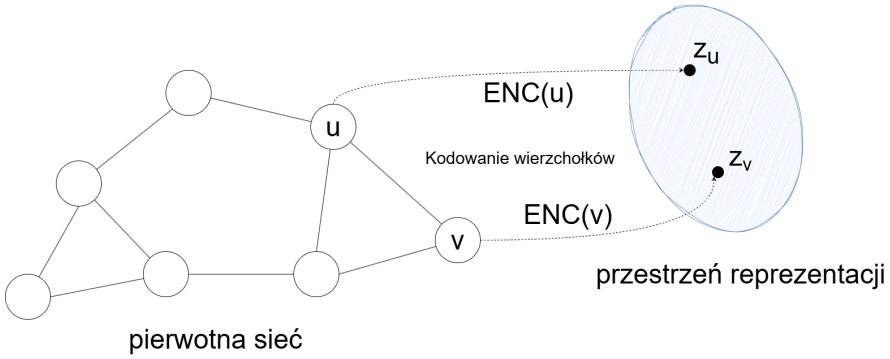
3.1.1. Czym są grafowe sieci neuronowe?

Grafowa sieć neuronowa (GNN - Graph Neural Network) to typ sieci neuronowej, która operuje na grafach. W przeciwieństwie do tradycyjnych sieci neuronowych, które działają na strukturach danych przypominających siatkę, takich jak obrazy lub sekwencje, sieci GNN są w stanie przechwytywać i modelować złożone relacje i zależności między elementami na grafie. Sieci GNN zyskały w ostatnich latach duże zainteresowanie ze względu na ich zdolność do skutecznej obsługi różnych zadań związanych z danymi o strukturze graficznej, takimi jak klasyfikacja węzłów, przewidywanie połączeń i generowanie grafów. Wykorzystując techniki przekazywania i agregacji komunikatów, sieci GNN mogą propagować informacje na grafie i uczyć się reprezentacji węzłów, które przechwytują zarówno lokalne, jak i globalne informacje strukturalne. To sprawia, że sieci GNN są potężnym narzędziem do analizy i zrozumienia złożonych danych relacyjnych.

Grafowe sieci neuronowe dobrze pasują do naszego problemu, ponieważ CVRP jest w naturalny sposób przedstawiany w postaci grafu. Miasta możemy traktować jako węzły w grafie. W takim przypadku przyjmiemy, że graf jest pełny, a każda krawędź będzie zawierała liczbę o wartości dodatniej jako atrybut reprezentujący odległość pomiędzy dwoma miastami, które łączy. Następnie możemy po prostu przekazać graficzną reprezentację problemu do GNN jako dane wejściowe.

3.1.2. Warstwy grafowej sieci neuronowej

Grafowa sieć neuronowa na wejściu może przyjąć dowolny graf, który jest reprezentowany jako zbiór wierzchołków oraz krawędzi. Celem sieci jest wyznaczenie funkcji kodującej każdy wierzchołek na wektor przechowujący informacje o strukturze oraz relacjach wierzchołków. Rysunek 3.1 obrazuje proces kodowania (ENC - encoding) wierzchołków:

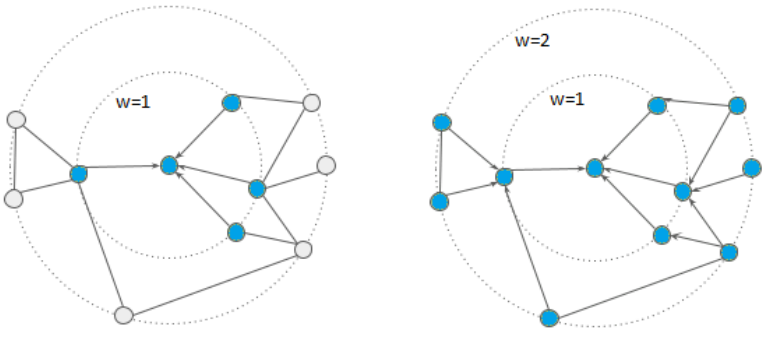


Rysunek 3.1. Kodowanie wierzchołków[1]

Przekazywanie wiadomości do kolejnych części grafu wykonywane jest w trzech krokach:

1. **Wyznaczenie wiadomości :** W przypadku naszej pracy korzystać będziemy z wyznaczonych wektorów przechowujący informacje o strukturze oraz relacjach wierzchołków. Będą to ”wiadomości” wysyłane przez każdy wierzchołek do swoich sąsiadów.
2. **Agregacja:** Każdy wierzchołek otrzymuje wiadomości od swoich sąsiadów. Wiadomości te są scalane przez funkcje agregujące takie jak: suma, średnia, wartość maksymalna itp.
3. **Aktualizacja:** Operacja aktualizacji bierze zagregowane wartości i aktualizuje reprezentacje wierzchołka. Wiele modeli aplikuje na tym etapie dodatkową transformacje liniową.

Większość grafowych sieci neuronowych wykorzystuje wiele warstw. W tym przypadku wyjście wyznaczone przez jedną warstwę jest przekazywane jako wejście do kolejnej warstwy. W ten sposób informacja o wierzchołku jest propagowana po całym grafie, nie tylko do bezpośrednich sąsiadów. Rysunek 3.2 przedstawia kolejne dwa etapy propagacji informacji o wierzchołkach z warstwy 1 do 2. Kolor niebieski pokazuje które wierzchołki już mają informację o wewnętrznych wierzchołkach (czyli o wierzchołkach z warstwy o niższym numerze).



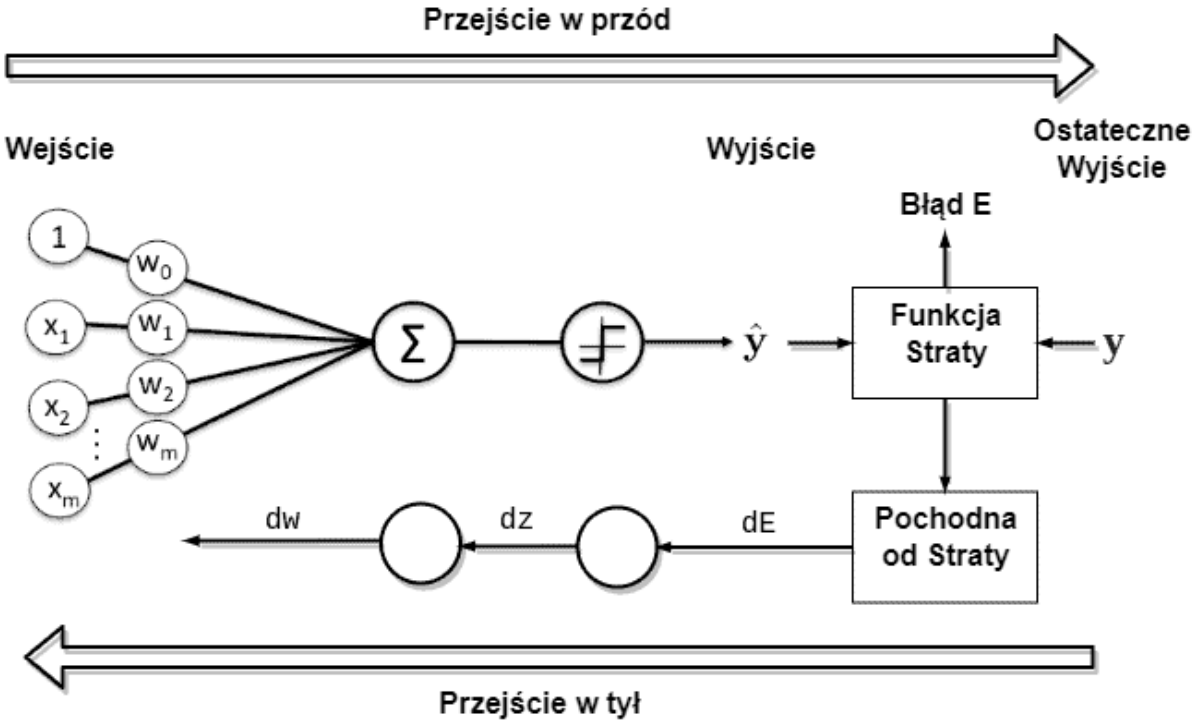
Rysunek 3.2. Propagacja informacji o wierzchołkach grafu[2]

[1] https://medium.com/the-modern-scientist/graph-neural-networks-series-part-3-node-embedding-36613cc967d5

[2] https://medium.com/stanford-cs224w/tackling-the-traveling-salesman-problem-with-graph-neural-networks-b86ef4300c6e

3.1.3. Epoki w sieciach neuronowych

Epoka w sieciach neuronowych oznacza, że sieć przeszła cały jeden cykl wykorzystując wszystkie dane ze zbioru danych treningowych dokładnie raz. Przejście do przodu oraz do tyłu razem liczą się jako jedno przejście. Jak widać na rysunku 3.3 przejście w przód składa się z przetworzenia danych przez aktualny model predykcji oraz obliczenia funkcji straty. Opis funkcji straty znajduje się w rozdziale 3.1.4. Przejście w tył składa się z obliczenia pochodnej od straty, a następnie za pomocą propagacji wstecznej wykonaniu aktualizacji parametrów modelu. W momencie, gdy funkcja straty osiągnie wystarczającą niską wartość lub przekroczymy maksymalną liczbę epok, trening sieci neruonowej zostaje zakończony.

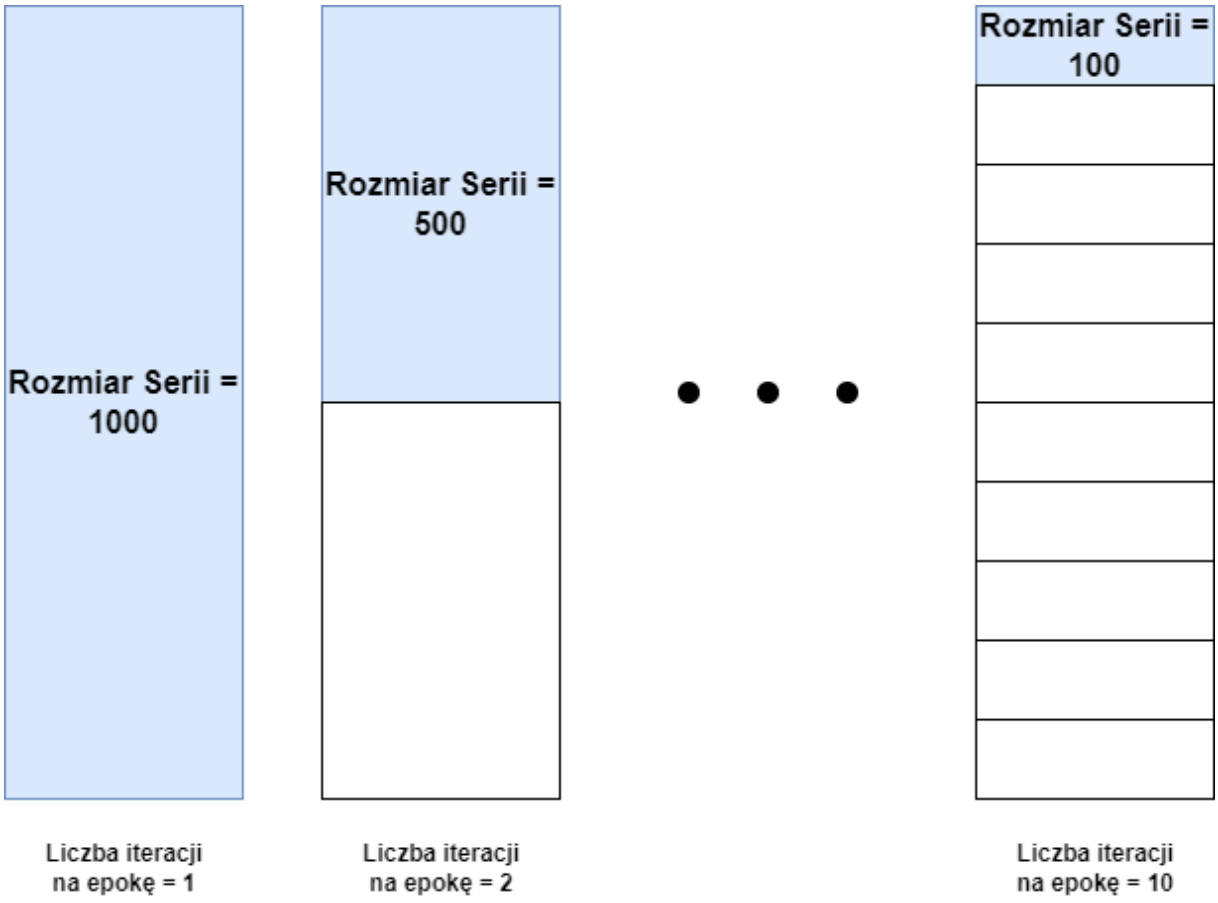


Rysunek 3.3. Schemat przedstawiający zakres działań w ciągu jednej epoki[3]

Epoka składa się z jednej lub większej liczby serii, gdzie serią jest część całego zbioru danych treningowych. Przejście przez jedną serię danych nazywamy iteracją. Pojęcie iteracji jest często mylone z pojęciem epoki. Żeby łatwiej zobrazować oba pojęcia rozpatrzmy rysunek 3.4.

Załóżmy, że rozmiar zbioru treningowego to 1000. Jeśli rozmiar serii jest równy 1000, to jedna epoka minie w ciągu jednej iteracji. Analogicznie, jeśli rozmiar serii jest równy 500 to jedna epoka minie w ciągu dwóch iteracji. Więc, jeśli rozmiar serii jest równy 100 to przejście przez jedną epokę będzie wymagało 10 iteracji. W uproszczeniu dla każdej epoki rozmiar całego zbioru treningowego jest równy liczbie iteracji razy rozmiar serii.

[3] https://www.baeldung.com/cs/epoch-neural-networks



Rysunek 3.4. Zależność między epoką, a iteracją[4]

3.1.4. Funkcja straty

Funkcja straty nazywana również funkcją błędu, jest kluczowym elementem w uczeniu maszynowym, który wyznacza różnicę między przywdzianymi wartościami algorytmu, a rzeczywistymi wynikami, na których trenowana była sieć. W przypadku tej pracy wartość funkcji straty obliczamy porównując rozkład prawdopodobieństwa wyjścia modelu z rozkładem prawdopodobieństwa rzeczywistego, wykorzystując stratę entropii krzyżowej. W bardzo dużym uproszczeniu można powiedzieć, że wartość funkcji straty jest to różnica między rozwiązaniem na którym uczymy model, a rozwiązaniem które model jest w stanie wygenerować tylko na podstawie danych wejściowych.

Wzór dla funkcji straty w oparciu o entropię krzyżową:

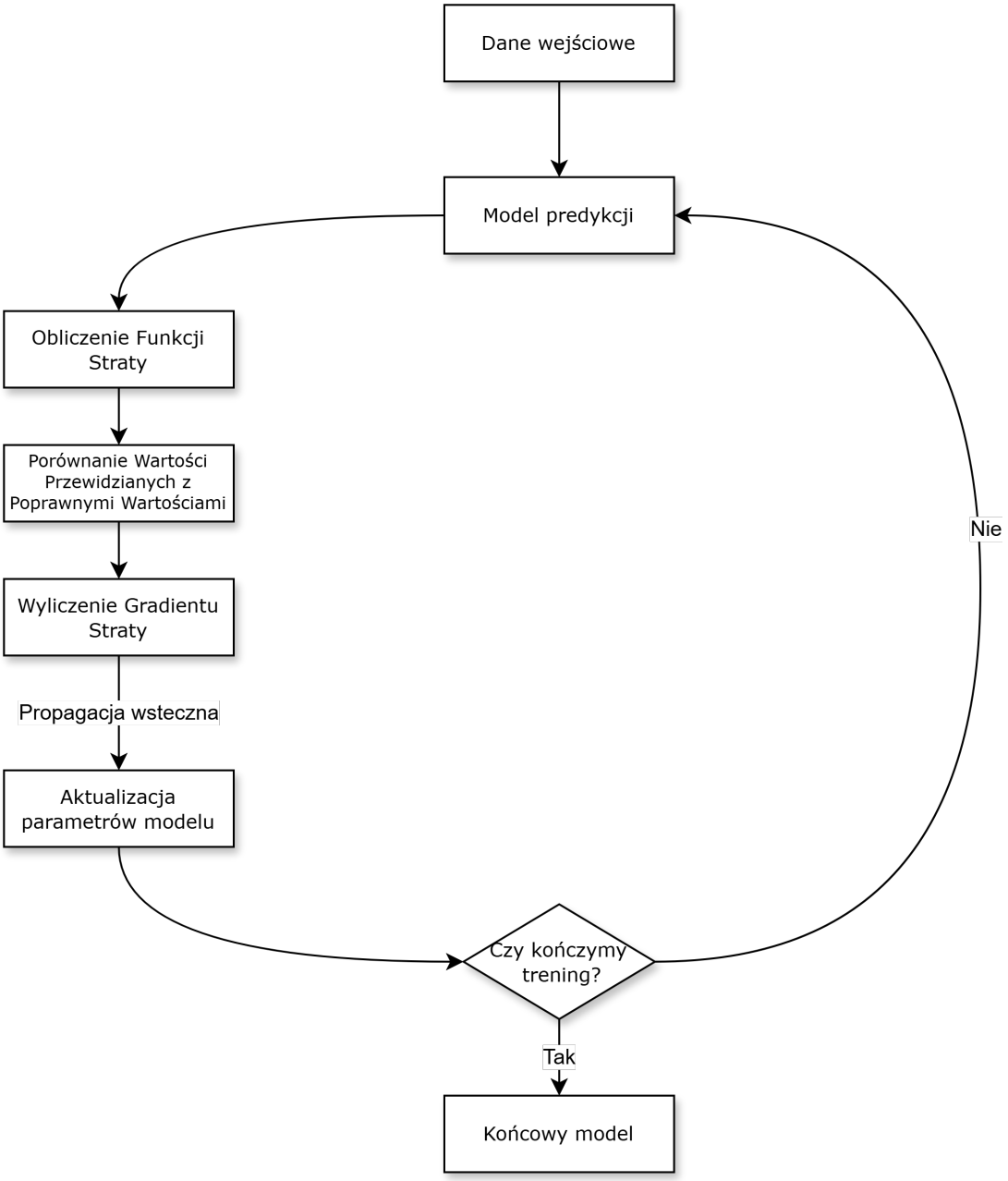
$$Loss = -\frac{1}{|V_L|} \sum_{v \in V_L} \sum_{i=1}^{C} y_{v,i} log(\hat{y}_{v,i}) \tag{3.1}$$

Gdzie:

[4] https://www.baeldung.com/cs/epoch-neural-networks

$|V_L|$ – liczba wierzchołków w zbiorze uczącym (V_L to zbiór wierzchołków z oznaczonymi etykietami).
C – liczba klas w problemie klasyfikacyjnym.
$y_{v,i}$ – rzeczywista etykieta wierzchołka v dla klasy i (zwykle reprezentowana jako one-hot encoding, tj. 1 dla prawdziwej klasy i 0 dla pozostałych).
$\hat{y}_{v,i}$ – przewidywane prawdopodobieństwo przez model dla wierzchołka v i klasy i.
$log(\hat{y}_{v,i})$ – logarytm naturalny z przewidywanego prawdopodobieństwa.

Wartość funkcji straty pokazuje również dokładność predykcji naszego modelu. Im mniejsza jest ta wartość tym model jest bardziej dokładny w swoich przewidywaniach. W trakcie treningu wartość funkcji straty wykorzystywana jest również do dostosowywania wartości parametrów tak, żeby wartość tej funkcji była jak najmniejsza, jednocześnie poprawiając jakość modelu. Schemat poniżej przedstawia ogólną rolę funkcji straty w procesie nauki naszego modelu:



Rysunek 3.5. Schemat działania sieci neuronowej

3.2. Algorytm zachłanny

Grafowa sieć neuronowa na wyjściu generuje nam rozkład prawdopodobieństw wykorzystania każdej krawędzi. Im większe jest to prawdopodobieństwo, tym większa jest szansa, że dana krawędź zostanie użyta w optymalnym rozwiązaniu. Tego typu wyjście nie jest jeszcze pełnym rozwiązaniem i wymaga obróbki przez dodatkowy algorytm. Moim pierwszym podejściem było zastosowanie algorytmu zachłannego.

Algorytm zachłanny polega na wyborze elementu o największym prawdopodobieństwie w danej iteracji.

$$e^* = arg \max_{e \in S} f(e) \tag{3.2}$$

Gdzie:

e^* – wybrany element w danej iteracji.
S – zbiór dostępnych elementów w danej chwili.
$f(e)$ – funkcja oceny elementu e, definiująca kryterium zachłanne (w tym przypadku maksymalizacja prawdopodobieństwa).
arg max - wybór elementu maksymalizującego wartość funkcji f.

W przypadku tej pracy elementem jest wierzchołek w grafie. Ponieważ w rozpatrywanym problemie mamy ograniczoną ładowność pojazdów, przed wybraniem następnego wierzchołka musimy sprawdzać czy zapotrzebowanie lokacji jest nie większe niż aktualnie dostępna ładowność samochodu (wynikająca z posiadanego towaru). Jeśli jest większe, to musimy wybrać kolejny najlepszy wierzchołek, w przypadku braku jakiegokolwiek pasującego wierzchołka wracamy do zajezdni. Kroki algorytmu zostały przedstawione poniżej

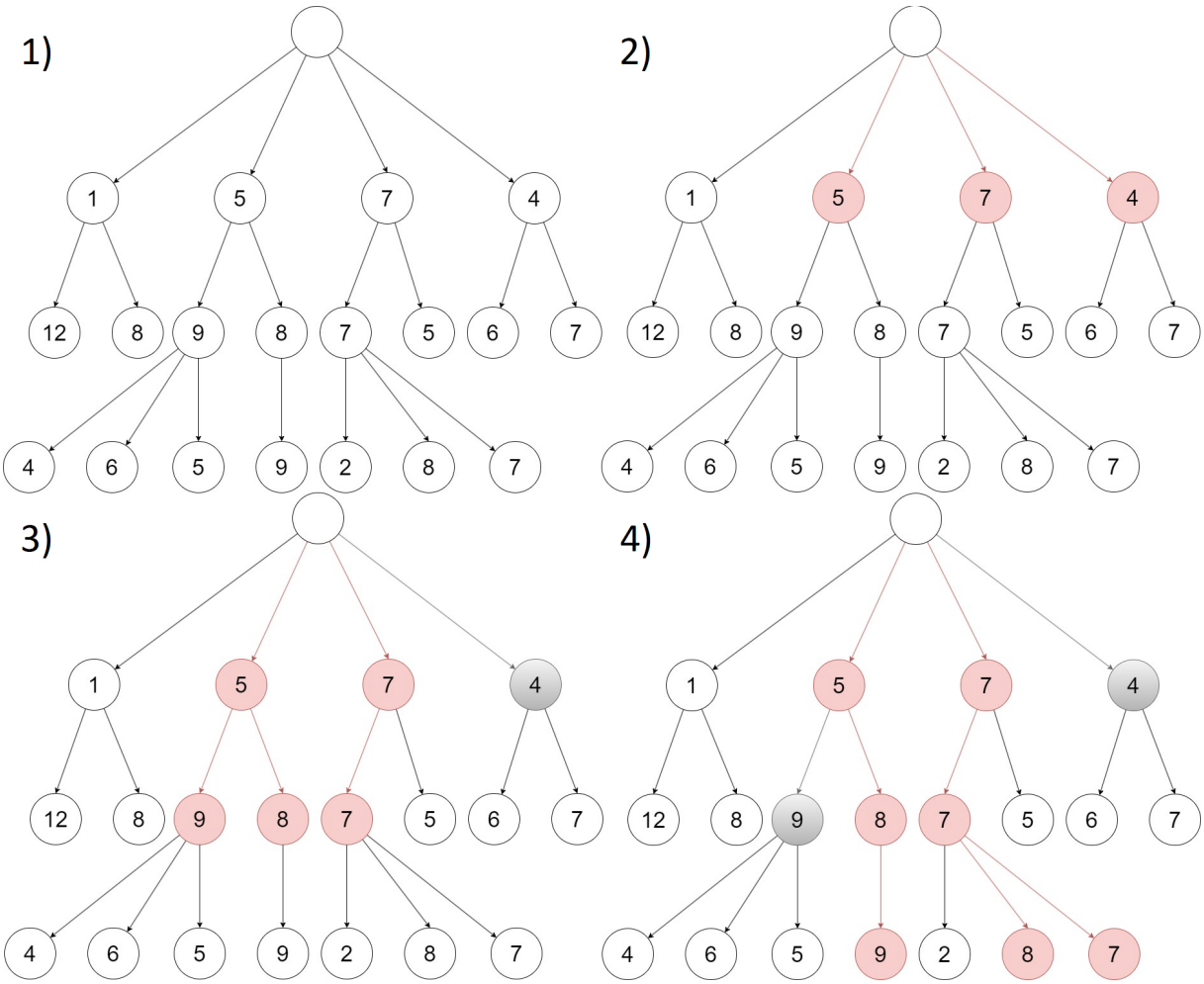
Algorytm 3.1 Pseudokod ilustrujący działanie algorytmu zachłannego dla problemu wyznaczania tras

```
1: function Greedy(rozklad_prawdopodobienst)
2:     zbior_tras ← [0]
3:     while zbior_tras nie jest ukończony do
4:         w ← ZnajdzNajlepszyWierzcholek(rozklad_prawdopodobienst)
5:         zbior_tras ← zbior_tras + w
6:         if w nie jest zajezdnią then
7:             UsunWierzcholekZeZbioru(rozklad_prawdopodobienst, w)
8:     return zbior_tras
```

3.3. Przeszukiwanie wiązkowe

Rozwinięciem algorytmu zachłannego jest przeszukiwanie wiązkowe. Algorytm ten działa na bardzo podobnej zasadzie, ale dochodzi w nim dodatkowy parametr wejściowy definiujący

szerokość wiązki przeszukiwania. Dalej w pracy będziemy się odwoływać do tego parametru jako k. Szerokość wiązki definiuje, jak wiele najlepszych rozwiązań zachowujemy w pamięci co iterację algorytmu. W przypadku $k = 1$ algorytm znajdzie to samo rozwiązanie co algorytm zachłanny, czyli w kolejnych iteracjach będzie przeszukiwał tylko jedno rozwiązanie. Dla większego rozmiaru k, wykonujemy te same operacje co w przypadku algorytmu zachłannego, ale dla każdego rozwiązania, które mamy w pamięci. Rysunek 3.6 obrazuje kolejne fazy przeszukiwania. Dla lepszej wizualizacji problemu w przykładzie zastosowano liczby całkowite zamiast prawdopodobieństw. Na pierwszym etapie drzewo jest w stanie przed rozpoczęciem przeszukiwania. W kolejnym etapie widać, że zostały wybrane 3 elementy o największej wartości. W trzecim etapie element o wartości 4 został usunięty z pamięci, ponieważ pojawiły się 3 inne rozwiązania o większej sumarycznej wartości na tym etapie. W ostatnim etapie widać kolejne usunięcie elementu, ponieważ elementy 5->8->9 (22), 7->7->8(22) oraz 7->7->7(21) mają sumarycznie większe wartości, niż elementy wychodzące od 5->9, czyli od 5->9->4 (18), 5->9->6 (20) oraz 5->9->5 (19).



Rysunek 3.6. Kolejne fazy przeszukiwania wiązkowego, dla k = 3

Algorytm ten w większość przypadków znajduje lepsze rozwiązania niż algorytm zachłanny, ponieważ przeszukuje większą pulę rozwiązań i na ich podstawie wybiera te najlepsze. W

przypadku naszego problemu przeszukiwanie wiązkowe zapisuje k najlepszych rozwiązań określonych na podstawie wyników sieci neuronowej. Następnie oceniamy każde z tych rozwiązań poprzez wyznaczenie długości trasy i wybieramy te z najkrótszą. Poniżej szczegółowy opis kolejnych kroków algorytmu:

Algorytm 3.2 Pseudokod ilustrujący działanie przeszukiwania wiązkowego dla problemu wyznaczania tras

```
1:  function BeamSearch(rozklad_prawdopodobienst, k)
2:      wielu_zbiorow_tras ← [[rozklad_prawdopodobienst, [0]]]
3:      next ← []
4:      while wielu_zbiorow_tras zawiera nie ukończoną trasę do
5:          for zbior_tras in wielu_zbiorow_tras do
6:              if zbior_tras nie jest ukończony then
7:                  for t in GenerujWszsytkieMozliwePrzystanki(zbior_tras) do
8:                      if t.ostatni_przystanek nie jest zajezdnią then
9:                          w ← t.ostatni_przystanek
10:                         UsunWierzcholekZeZbioru(t.rozklad_prawdopodobienst, w)
11:                     next ← next + zbior_tras
12:             else
13:                 next ← next + zbior_tras
14:         wielu_zbiorow_tras ← ZwrocNajlepszeTrasy(next, k)
15:     return ZwrocNajkrotszaTrase(wielu_zbiorow_tras)
```

3.4. Przeszukiwanie drzewa Monte-Carlo

3.4.1. Ogólny opis działania

Przeszukiwanie drzewa Monte-Carlo jest metodą polegającą na inteligentnym przeszukiwaniu drzewa balansując między eksploracją, a eksploatacją. Algorytm ten wykonuje próbkowania w formie symulacji i zapisuje statystyki tych akcji w celu wykonywania lepszych wyborów w kolejnych iteracjach.

Metoda ta przeszukuje tylko kilka warstw w głąb drzewa i priorytetyzuje, które części drzewa przeszuka w następnych iteracjach. Następnie symuluje wyjście, zamiast rozszerzania przestrzeni przeszukiwanych rozwiązań. W ten sposób ogranicza liczbę ewaluacji, którą algorytm musiałby wykonać w celu wyznaczenia całej trasy.

Pojedyncza ewaluacja polega na symulacji, w której algorytm wyznacza rozwiązanie od rozpatrywanego wierzchołka drzewa i zapisuje rezultat tych symulacji na każdym wierzchołku drzewa, który prowadził do rozpatrywanego wierzchołka. W normalnym przypadku metoda ta symuluje ruchy 2 graczy, żeby osiągnąć końcową postać planszy do gry (na przykład w grze szachy). W przypadku problemu wyznaczania tras z ograniczoną ładownością, wykorzystamy

przeszukiwanie wiązkowe oraz algorytm zachłanny, żeby zasymulować wyznaczenie trasy. Po wykonaniu symulacji algorytm przeszukiwania drzewa Monte-Carlo wybiera stan zawierający najbardziej obiecujące rozwiązania.

Metoda ta składa się z następujących faz:

1. Selekcja
2. Ekspansja
3. Symulacja
4. Propagacja wsteczna

Algorytm 3.3 Pseudokod ilustrujący działanie algorytmu przeszukiwania drzewa Monte-Carlo

```
1: function MonteCarloTreeSearch(stan_pocztkowy)
2:     wezel_aktualny ← UtworzWezel(stan_poczatkowy)
3:     while limit_iteracji nie został osiągnięty do
4:         wezel ← Selekcja(wezel_aktualny)
5:         if wezel nie jest liściem then
6:             wezel ← Ekspansja(node)
7:         wynik ← Symulacja(wezel.stan)
8:         PropagacjaWsteczna(wezel, wynik)
9:     return NajlepszyRuch(wezel_aktualny)
```

3.4.2. Selekcja

W tej fazie algorytm używa następującej formuły do obliczenia wartości stanów wszystkich możliwych ruchów i wybiera ten z maksymalną wartością:

$$a^* = arg \max_{a \in A(s)} (Q(s,a) + C * \sqrt{\frac{lgN(s)}{N(s,a)}}) \tag{3.3}$$

Gdzie:

a^* – wybrana akcja w stanie s.
$A(s)$ - zbiór wszystkich dostępnych akcji w stanie s.
$Q(s,a)$ – średnia wartość wierzchołka dla akcji a w stanie s na podstawie dotychczasowych symulacji.
C – współczynnik eksploracji, kontrolujący balans między eksploracją a eksploatacją (zwykle $C > 0$).
$N(s)$ – liczba wszystkich odwiedzeń stanu s.
$N(s,a)$ – liczba odwiedzeń akcji a w stanie s.

Pierwsza część formuły $Q(s,a)$ to element eksploatacji, a druga część formuły $\sqrt{\frac{lgN(s)}{N(s,a)}}$ to element eksploracji. Parametr C określa, jak agresywna ma być eksploracja. Wyższe wartości C faworyzują eksplorację, podczas gdy niższe wartości promują eksploatację. Ten wzór definiuje stosunek ważności między eksploatacją oraz eksploracją.

Na początku, kiedy żaden wierzchołek nie był eksplorowany, metoda wykonuje losową selekcję, ponieważ wzór nie ma żadnych informacji, na podstawie których mógłby podjąć decyzję. Kiedy wierzchołek nie był eksplorowany, tzn. kiedy $N(s,a) = 0$, druga część równania zmierza do ∞ i otrzymuje najwyższą możliwą wartość, przez co automatycznie staje się kandydatem do selekcji. W praktyce stosuje się logikę IF-THEN, tzn. jeśli $N(s,a) = 0$ to automatycznie wybierana jest akcja a. W ten sposób mamy gwarancję, że wszystkie wierzchołki zostaną wybrane co najmniej raz.

3.4.3. Ekspansja

Na tym etapie wykonujemy ekspansję od wybranego wierzchołka drzewa i przechodzimy o poziom niżej. Wierzchołek od którego przeprowadziliśmy ekspansję staje się rodzicem (aktualny stan), a jego dzieci stają się możliwymi ruchami.

3.4.4. Symulacja

W standardowej fazie tego algorytmu symulujemy wynik gry poprzez losowy wybór ruchów obu graczy, aż do osiągnięcia stanu gdzie żaden z graczy nie może wykonać ruchu (stan terminalny). W naszej wersji algorytmu do rozwiązywania problemu wyznaczenia tras użyjemy metody opisanej w poprzednim rozdziale, czyli przeszukiwania wiązkowego. Jeżeli wynik tego algorytmu jest lepszy niż aktualny najlepszy wynik zapisany w rozpatrywanym wierzchołku drzewa lub jest to pierwsza symulacja dla tego wierzchołka to nadpisujemy tą informację.

3.4.5. Propagacja wsteczna

Na tym etapie przepisujemy informacje z symulacji do wszystkich rodziców prowadzących od aktualnie rozpatrywanego wierzchołka. Informacje te są następnie wykorzystywane w kolejnej iteracji algorytmu na etapie selekcji.

3.5. Wiązkowe przeszukiwanie drzewa Monte-Carlo

Ostatni wykorzystany algorytm łączy logikę poprzednich dwóch metod czyli przeszukiwanie wiązkowe z przeszukiwaniem drzewa Monte-Carlo. Algorytm ten nazywamy wiązkowym przeszukiwaniem drzewa Monte-Carlo (Beam Monte-Carlo Tree Search, BMCTS). W metodzie BMCTS poza logiką wprowadzoną w poprzednim rozdziale dochodzi nam licznik na każdej głębokości drzewa, sumujący liczbę symulacji z wszystkich wierzchołków na danej głębokości drzewa. W fazie propagacji wstecznej licznik ten jest porównywany z pierwszym parametrem BMCTS: limitem symulacji. Jeśli którakolwiek głębokość drzewa d osiągnie ten limit, to drzewo

jest ucinane na poziomie d.

Ucinanie ogranicza liczbę wierzchołków drzewa na głębokości d do maksymalnej liczby wierzchołków podanych jako drugi parametr algorytmu BMCTS, nazywany dalej szerokością promienia W. Żeby poprawnie uciąć te wierzchołki trzeba najpierw nadać każdemu wierzchołkowi jakąś przybliżoną wartość. W przypadku naszej wersji algorytmu wartość ta będzie wyznaczana przez przeszukiwanie wiązkowe. Następnie najlepsze W wierzchołków wraz z ich rodzicami zachowujemy, a pozostałe mniej obiecujące wierzchołki zostają usunięte.

Algorytm 3.4 Pseudokod ilustrujący działanie wiązkowego przeszukiwania drzewa Monte-Carlo

```
1:  function BeamMonteCarloTreeSearch(stan_pocztkowy)
2:      wezel_aktualny ← UtworzWezel(stan_poczatkowy)
3:      while limit_iteracji nie został osiągnięty do
4:          liczba_iteracji[d] + +
5:          wezel ← Selekcja(wezel_aktualny)
6:          if wezel nie jest liściem then
7:              wezel ← Ekspansja(node)
8:          wynik ← Symulacja(wezel.stan)
9:          PropagacjaWsteczna(wezel, wynik)
10:         if liczba_iteracji[d] = limit_symulacji then
11:             UtnijDrzewo(d)
12:     return NajlepszyRuch(wezel_aktualny)
```

Algorytm 3.5 Pseudokod ilustrujący działanie funkcji UtnijDrzewo

```
1: function UtnijDrzewo(d)
2:     zestaw_wezlow ← WezlyNaGlebokosci(drzewo, d)
3:     zestaw_wezlow ← NajlpszeWezly(zestaw_wezlow, W)
4:     drzewo ← zestaw_wezlow + RodziceWezlow(zestaw_wezlow)
```

W kolejnych etapach przeszukiwania drzewa nie dodajemy już nowych wierzchołków, aż do głębokości d. Proces selekcji uwzględnia tylko te ruchy, które prowadzą do zachowanych wierzchołków. Poniżej głębokości d drzewo jest przeszukiwane tak jak wcześniej.

4. Organizacja eksperymentów obliczeniowych

4.1. Dane testowe

Na moment pisania tej pracy nie byliśmy w stanie znaleźć wystarczająco dużo danych testowych do treningu opisanej sieci neuronowej. Wzorując się na przykładzie innej pracy [11], postanowiliśmy wygenerować przypadki testowe i rozwiązać je za pomocą narzędzia OR-Tools.

OR-Tools to pakiet oprogramowania typu open source do optymalizacji, umożliwiający rozwiązywanie problemów związanych z wyznaczaniem tras pojazdów, przepływami, programowaniem całkowitoliczbowym i liniowym oraz programowaniem z ograniczeniami.

Dane testowe wygenerowane zostały dla trzech wielkości problemów. Rozpatrywany problem polega na wyznaczeniu pewnej liczby tras, w której każda zaczyna się i kończy w tym samym punkcie (zajezdni). W ramach pracy zdecydowaliśmy się na następujące wielkości problemu: 13 (1+1*12), 25 (1+2*12) oraz 49 (1+4*12). Jak widać każda kolejna wielkość problemu ma 2 razy więcej przystanków do odwiedzenia niż poprzednia. Dla każdej wielkości wygenerowano 10 000 przypadków testowych. Ponieważ OR-Toolsy nie gwarantują znalezienia optymalnego rozwiązania, w parametrach wejściowych trzeba ustalić jak dużo czasu skrypt ma spędzić w celu znalezienia jak najlepszego rozwiązania. Wraz ze wzrostem wielkości problemu wzrasta czas potrzebny na znalezienie optymalnego rozwiązania, dlatego w celu wyznaczenia czasu zrobiliśmy testy dla pierwszych kilkunastu przypadków testowych. Test polegał na dwukrotnym zwiększaniu limitu czasu i sprawdzaniu czy jakość znalezionego rozwiązania się poprawia. W momencie, gdy jakość rozwiązań przestała się poprawiać wybieraliśmy najkrótszy czas umożliwiający znalezienie najlepszej jakości. Wyniki testów umożliwiły nam wybranie następujących czasów dla kolejnych wielkości problemów:

- 13 wierzchołków – 3 sekundy,
- 25 wierzchołków – 15 sekund,
- 49 wierzchołków – 60 sekund.

Po wygenerowaniu rozwiązań do wszystkich danych, podzieliśmy je na kolejne trzy zestawy:

- 8000 – zestawy treningowe,

- 1000 – zestawy testowe,
- 1000 – zestawy walidacyjne.

4.2. Metodyka badań

Testy przeprowadzane były pod kątem wpływu parametrów charakterystycznych na wyniki algorytmów. Dla grafowej sieci neuronowej sprawdzony zostanie wpływ liczby epok oraz liczby warstw na wartość funkcji straty. Dla przeszukiwania wiązkowego sprawdzony został wpływ szerokości promienia na jakość rozwiązania i czas potrzebny do jego znalezienia. Porównany zostanie również bezpośrednio z wynikami wygenerowanymi przez algorytm zachłanny. Dla przeszukiwania drzewa Monte-Carlo (MCTS) sprawdzony zostanie wpływ szerokości promienia dla etapu symulacji na jakość rozwiązania i czas potrzebny do jego znalezienia. Ponieważ w wiązkowym przeszukiwaniu drzewa Monte-Carlo (BMCTS) pojawiają się 2 nowe parametry (limit symulacji oraz szerokość promienia ucinanego drzewa) parametr szerokości promienia dla etapu symulacji zostanie wyznaczony w trakcie testów MCTS. Dla MCTS oraz BMCTS porównany zostanie również czas każdego etapu algorytmu. Maksymalna liczba iteracji (ang. rollout) dla obu algorytmów została wyznaczona na 800 dla wielkości problemu 13 oraz 25, a dla 49 na 1200.

Ponieważ, obecnie nie ma wystarczająco dużej bazy danych z problemami CVRP i optymalnymi rozwiązaniami do nich, dla testów w tej pracy użyte zostanie pierwsze 100 zestawów ze zbioru danych walidacyjnych. Liczba 100 została wybrana ze względu na ograniczone zasoby czasowe. Jakość znalezionych rozwiązań zostanie porównana z jakością rozwiązań znalezioną przez OR-Toolsy. Po zakończeniu eksperymentów dla sieci neuronowej, dla każdej wielkości problemu zostanie wybrana sieć z najniższą wartością funkcji straty, ale dalej ze stabilnym procesem uczenia (funkcja straty dla zbiorów testowych i walidacyjnych nie zaczyna odbiegać od wartości funkcji straty dla zbioru treningowego) i ta sieć zostanie użyta do generacji danych wejściowych dla wszystkich pozostałych algorytmów.

Wszystkie testy przeprowadzone zostały na tym samym sprzęcie z następującym procesorem i kartą graficzną: AMD Ryzen 7 3800X 8-Core Processor 3.90 GHz, NVIDIA GeForce RTX 2080 SUPER. Ma to znaczenie w przypadku badania i porównania czasu pracy powyższych algorytmów.

5. Wyniki analizy algorytmów

5.1. Grafowa sieć neuronowa

5.1.1. Funkcja straty

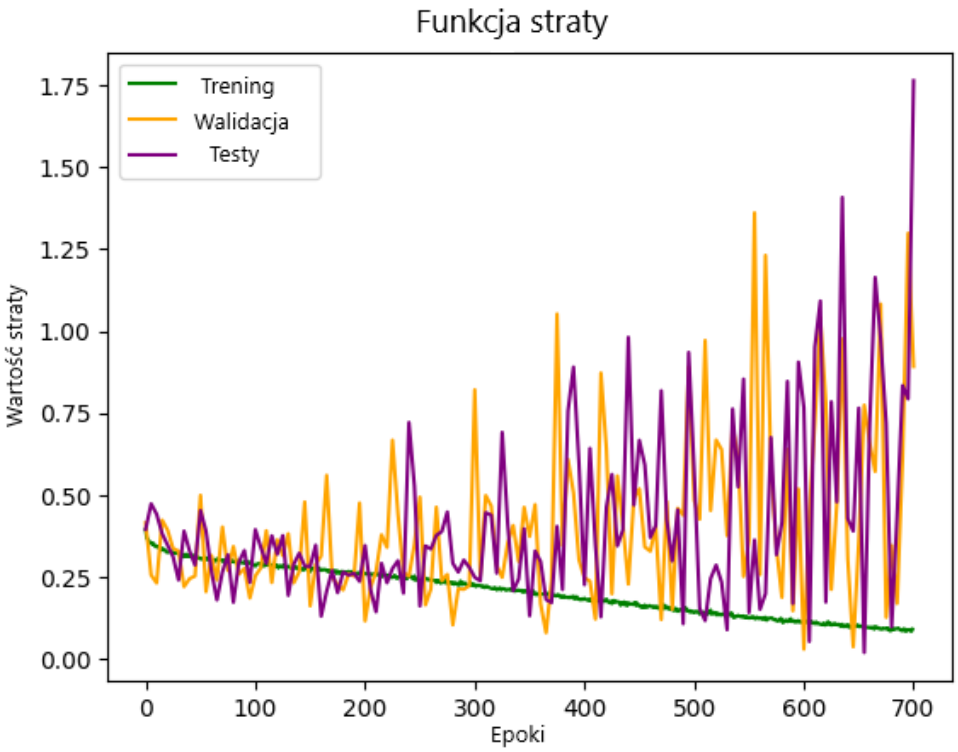
Pierwszy test polegał na zbadaniu wpływu liczby warstw oraz liczby epok na wartość funkcji straty w trakcie uczenia grafowej sieci neuronowej. Testy zostały przeprowadzone dla 9 przypadków:

- wielkość problemu 13, warstwy ukryte 100
- wielkość problemu 13, warstwy ukryte 200
- wielkość problemu 13, warstwy ukryte 300
- wielkość problemu 25, warstwy ukryte 100
- wielkość problemu 25, warstwy ukryte 200
- wielkość problemu 25, warstwy ukryte 300
- wielkość problemu 49, warstwy ukryte 100
- wielkość problemu 49, warstwy ukryte 200
- wielkość problemu 49, warstwy ukryte 300

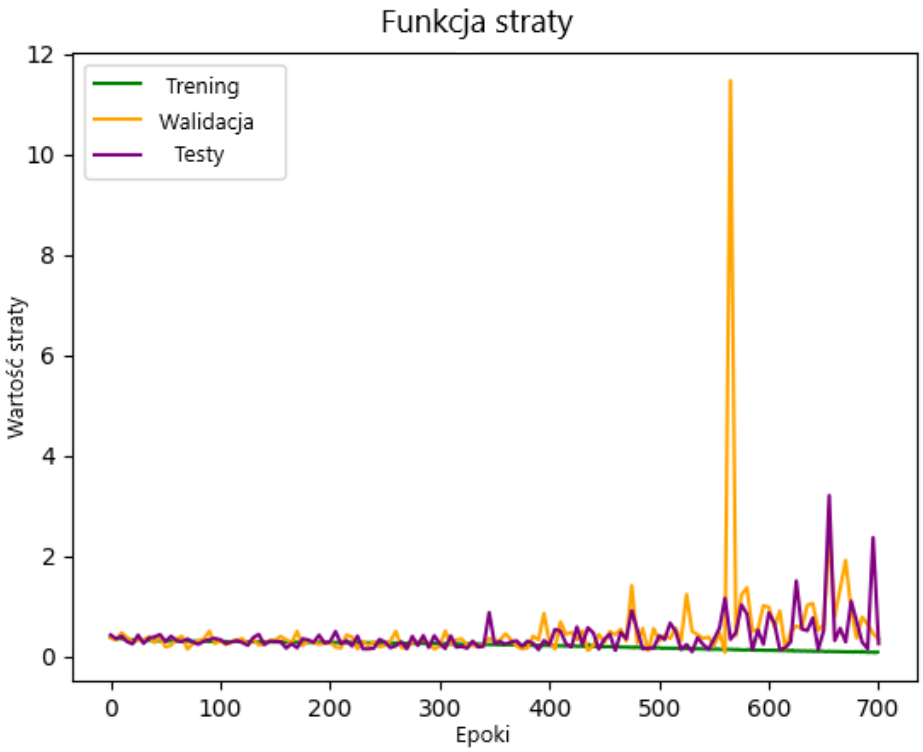
Przy następującej wielkości zbiorów danych:

- zestaw treningowy 8000 przypadków
- zestaw testowy 1000 przypadków
- zestaw walidacyjny 1000 przypadków

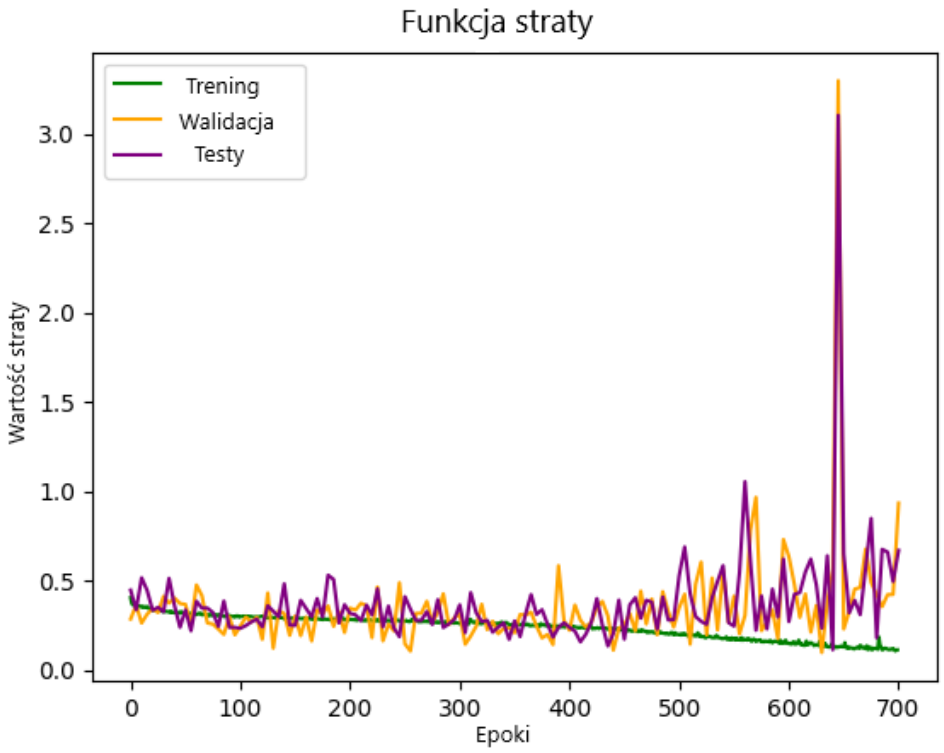
Na stronach 27-31 przedstawiono wykresy funkcji straty z procesu uczenia sieci:



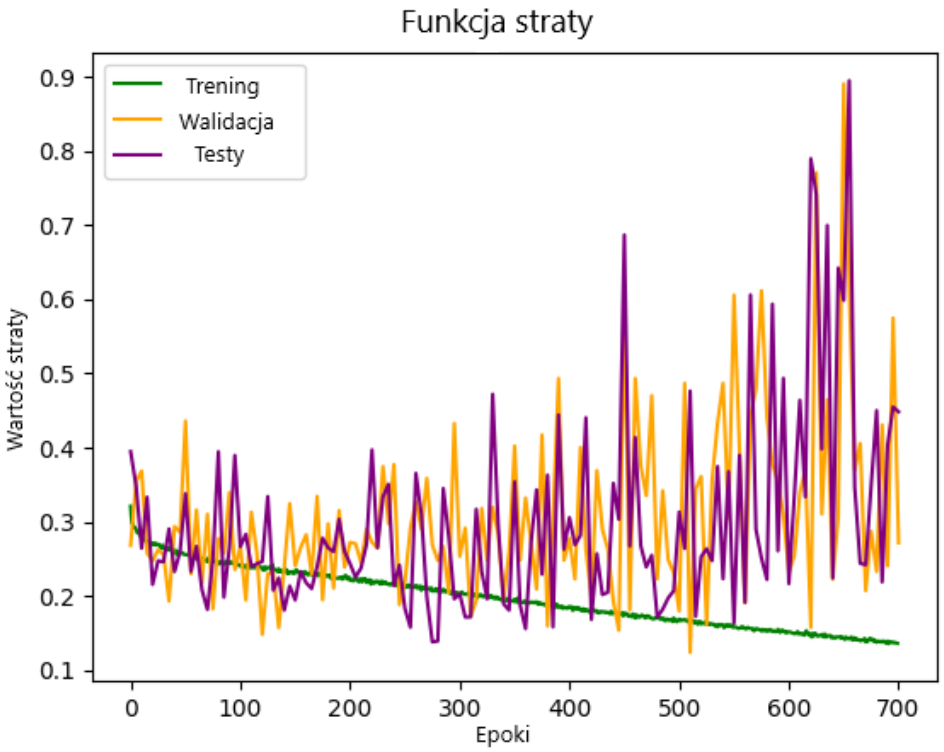
Rysunek 5.1. Funkcja straty (wielkość problemu 13, warstwy ukryte 100)



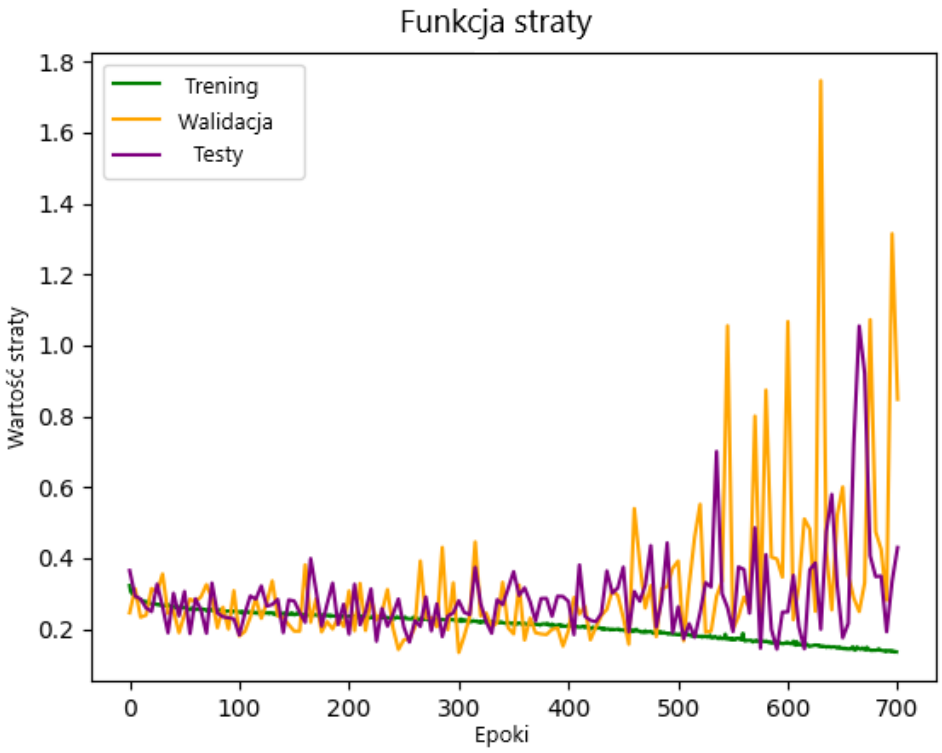
Rysunek 5.2. Funkcja straty (wielkość problemu 13, warstwy ukryte 200)



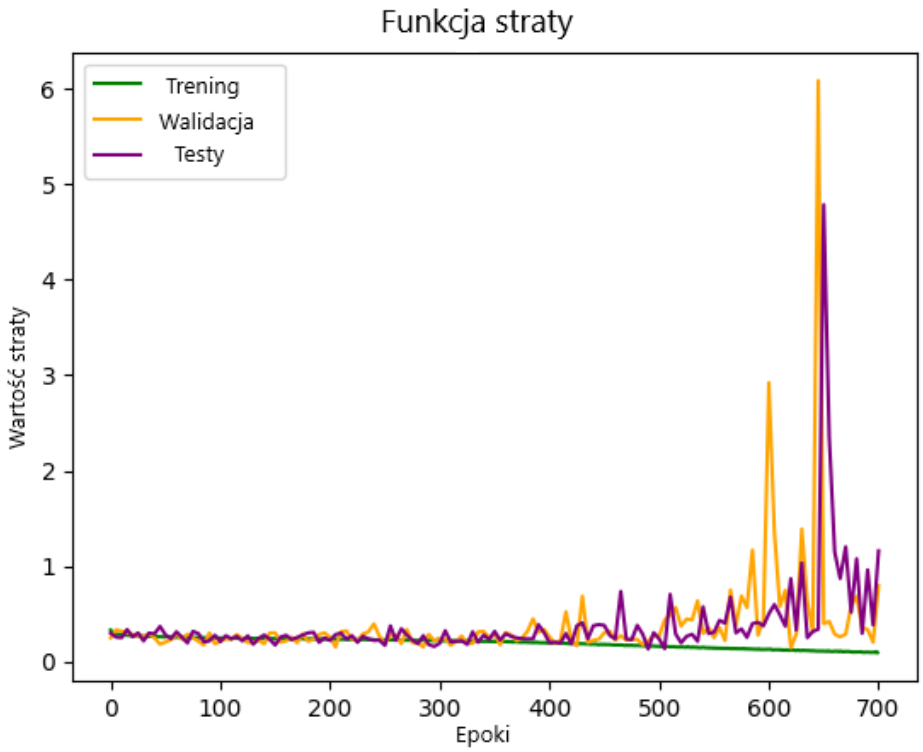
Rysunek 5.3. Funkcja straty (wielkość problemu 13, warstwy ukryte 300)



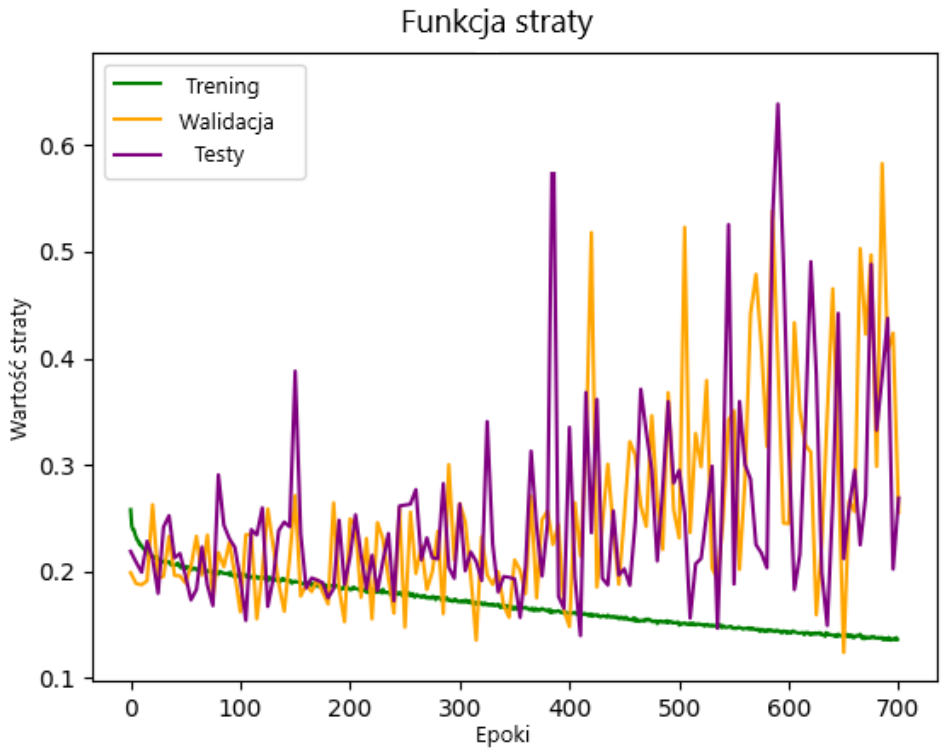
Rysunek 5.4. Funkcja straty (wielkość problemu 25, warstwy ukryte 100)



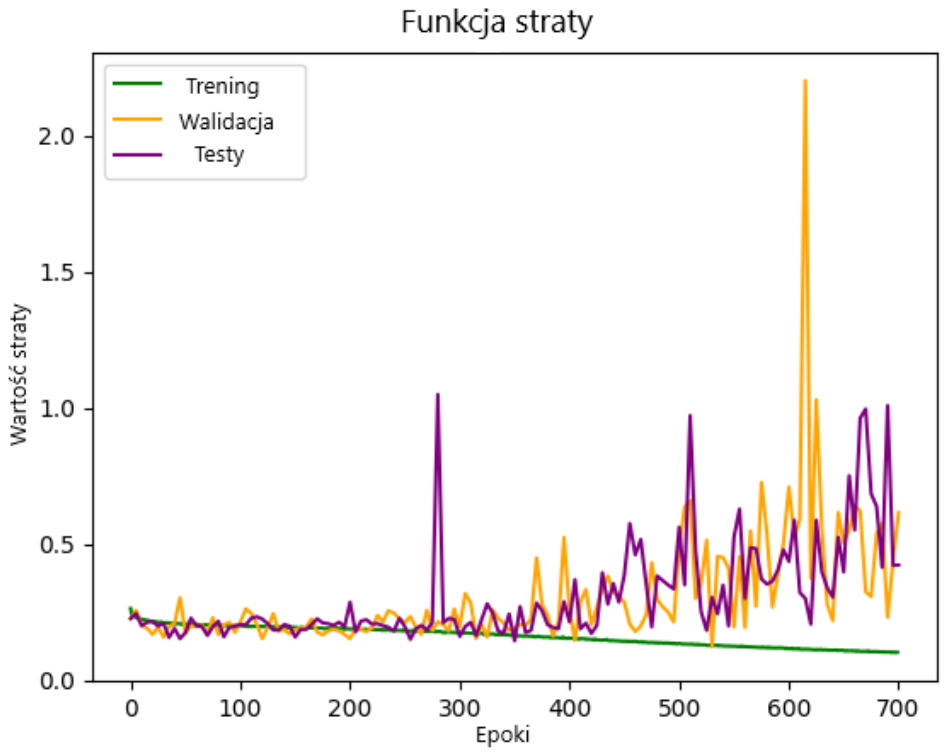
Rysunek 5.5. Funkcja straty (wielkość problemu 25, warstwy ukryte 200)



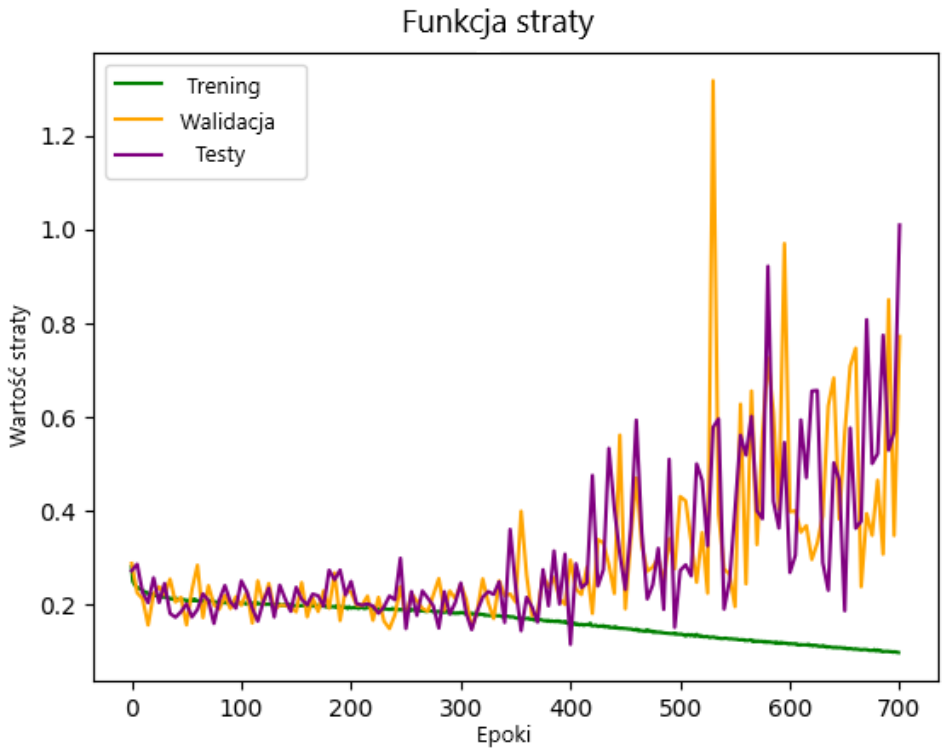
Rysunek 5.6. Funkcja straty (wielkość problemu 25, warstwy ukryte 300)



Rysunek 5.7. Funkcja straty (wielkość problemu 49, warstwy ukryte 100)



Rysunek 5.8. Funkcja straty (wielkość problemu 49, warstwy ukryte 200)



Rysunek 5.9. Funkcja straty (wielkość problemu 49, warstwy ukryte 300)

Z rysunków widać że wraz ze wzrostem liczby epok, wartość funkcji straty dla zbioru trenującego spada. W każdym z rysunków można zauważyć moment, w którym sieć staje się przetrenowana tzn. wartość funkcji straty na zbiorze testowym i walidacyjnym zaczyna odbiegać od wartości funkcji straty na zbiorze treningowym (są to również okresy w których mogą występować gwałtowne skoki wartości funkcji straty dla zbiorów testowych i walidacyjnych). Gdy porównamy wykresy pod kątem wielkości problemu widać, że dla większych problemów (49 wierzchołków) moment, w którym sieć staje się przetrenowana następuje szybciej niż dla najmniejszego problemu (13 wierzchołków). W każdym przypadku wartość funkcji straty kończy się w przedziałach wartości $< 0.1; 0.2 >$. Dodatkowo liczba warstw wpływa na szybkość uczenia się sieci, im więcej warstw tym szybciej spada wartość funkcji straty.

Z otrzymanych wyników oraz dostępnej literatury możemy wywnioskować, że sieć dałoby się lepiej wytrenować gdybyśmy mieli więcej danych. W literaturze przykładowo, dla problemu komiwojażera, często można się spotkać z wielkością zbiorów testowych w okolicach 1 000 000. Niestety dla problemu CVRP obecnie nie ma żadnego publicznego zbioru z tak dużą ilością danych, a czas potrzebny na ich wygenerowanie jest za duży. Używany zbiór dla wielkości problemu 49 został wygenerowany w tydzień (10000 * 60 sekund). Przy tej samej prędkości czas na wygenerowanie miliona przypadków zająłby około 45 lat.

Na podstawie otrzymanych wyników wybraliśmy następujące modele sieci do wykorzysta-

nia w kolejnych eksperymentach (sieci z najniższymi wartościami funkcji straty, ale dalej ze stabilnymi procesami uczenia):

- liczba wierzchołków 13, liczba epok 350, warstwy ukryte 300
- liczba wierzchołków 25, liczba epok 400, warstwy ukryte 300
- liczba wierzchołków 49, liczba epok 300, warstwy ukryte 300

5.1.2. Średni czas epoki

Kolejny test polegał na zbadaniu, jak zmienia się średni czas jednej epoki w zależności od wielkości problemu oraz liczby warstw. Wyniki zostały przedstawione w poniższej tabeli:

Wielkość problemu	Liczba warstw ukrytych	Średni czas epoki (s)
13	100	15,474
13	200	27,953
13	300	41,370
25	100	15,734
25	200	28,486
25	300	42,007
49	100	15,832
49	200	28,944
49	300	42,921

Tabela 5.1. Średnia czasy epoki dla GNN

Z powyższego eksperymentu wynika, że wielkość problemu ma znikomy wpływ na czas potrzebny do treningu sieci w jednej epoce, ale wzrost czasu jest wprost proporcjonalny do wzrostu liczby warstw.

5.2. Przeszukiwanie wiązkowe oraz algorytm zachłanny

Test dla przeszukiwania wiązkowego oraz algorytmu zachłannego polegał na wykorzystaniu rezultatów wygenerowanych przez grafową sieć neuronową w celu wygenerowania macierzy predykcji wystąpienia każdej krawędzi w grafie. Następnie omawiane w tym rozdziale algorytmy przeprocesowują tą macierz w celu wyznaczenia jak najlepszego rozwiązania. Podsumowanie eksperymentów dla pierwszych 100 zestawów ze zbioru walidacyjnego zostało przedstawione w tabeli 5.2:

Wielkość	Szerokość promienia	Średni czas (s)	MAX (%)	MIN (%)	Średnia (%)
13	1 (zachłanny)	0,00026	156,01	96,83	118,86
13	1	0,00987	156,01	96,83	118,86
13	4	0,03202	160,08	93,91	109,83
13	16	0,12763	121,64	84,84	105,42
13	64	0,47976	118,76	84,84	103,27
25	1 (zachłanny)	0,00089	143,11	104,76	121,37
25	1	0,10492	143,11	104,76	121,56
25	4	0,35241	134,55	103,73	114,82
25	16	1,33454	121,55	101,76	108,56
25	64	4,92985	118,92	100,13	107,07
49	1 (zachłanny)	0,00368	135,53	107,83	119,76
49	1	1,37550	135,53	107,83	119,93
49	4	4,61351	134,55	105,94	116,79
49	16	15,61381	127,12	102,73	112,28
49	64	60,29774	120,69	99,12	110,02

Tabela 5.2. Wyniki dla przeszukiwania wiązkowego

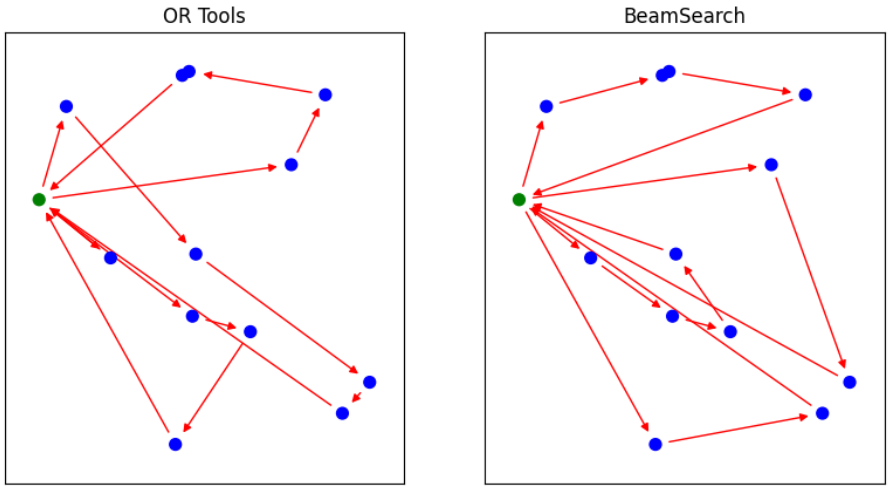
W tabeli 5.2 jakość rozwiązań została przedstawiona jako odchylenie procentowe od rozwiązania wyznaczonego przez OR-Tools dla tego samego zestawu danych (długość trasy wyliczona/długość trasy OR-Tools*100%). ”MAX” przedstawia najgorszy przypadek testowy, ”MIN” przedstawia najlepszy przypadek testowy, a ”Średnia” przedstawia średnie odchylenie na podstawie wszystkich przeprocesowanych przypadków. Użycie algorytmu zachłannego zostało zaznaczone w kolumnie ”Szerokość promienia”. ”Średni czas” przedstawia ile sekund zajęło algorytmowi znalezienie rozwiązania.

Na podstawie wygenerowanych wyników można wyciągnąć następujące wnioski:

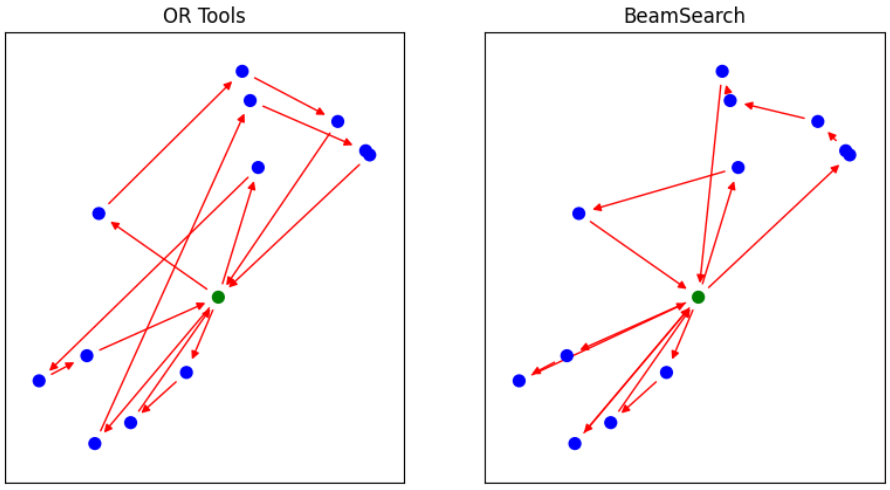
— Dla szerokości promienia równej 1 użycie algorytmu zachłannego jest dużo bardziej opłacalne czasowo, niż użycie przeszukiwania wiązkowego.
— Zwiększanie szerokości promienia wprost proporcjonalnie zwiększa czas potrzebny na znalezienie rozwiązania.

— Wraz ze wzrostem szerokości promienia polepsza się jakość znajdywanych rozwiązań.
— Nawet dla największej wielkości problemu (49) jakość wyników jest zbliżona do tych osiąganych przez OR-Toolsy. Dla szerokości promienia 64, udało się nawet znaleźć rozwiązanie lepsze niż te osiągnięte przez OR-Toolsy.

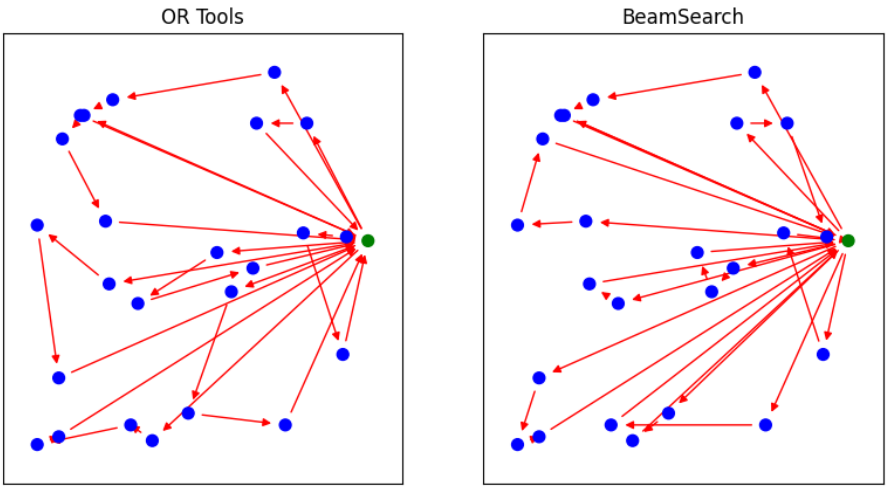
Kolejne rysunki przedstawiają wygenerowane trasy dla najlepszego i najgorszego przypadku znalezionego przez przeszukiwanie wiązkowe z szerokością promienia równą 64. Po lewej stronie przedstawione zostało rozwiązanie znalezione przez OR-Toolsy, po prawej znalezione przez przeszukiwanie wiązkowe.



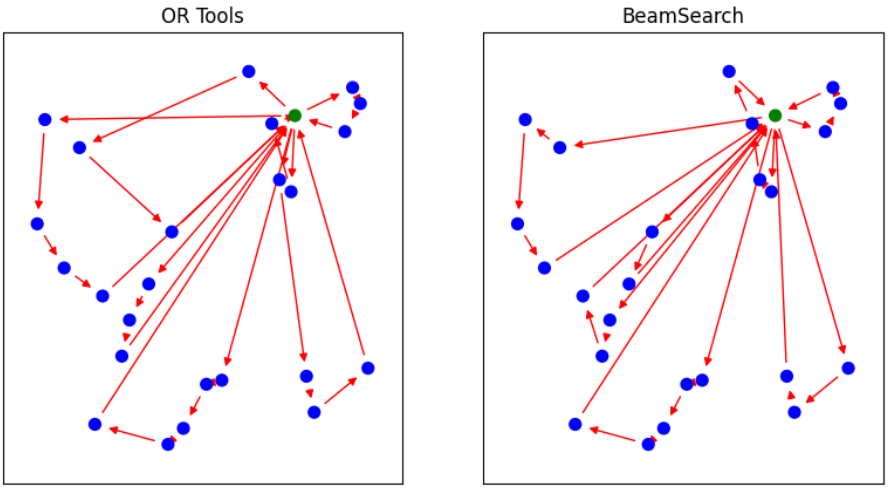
Rysunek 5.10. Wielkość problemu 13, jakość znalezionego rozwiązania 118,76%



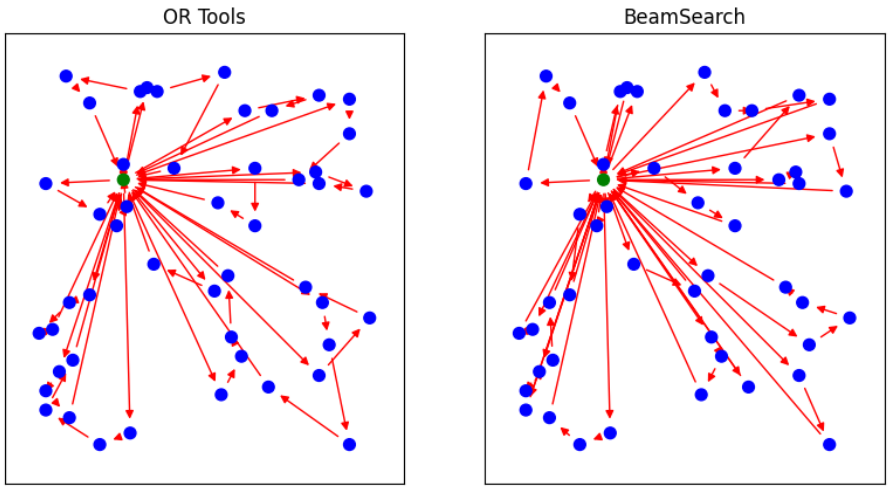
Rysunek 5.11. Wielkość problemu 13, jakość znalezionego rozwiązania 84,84%



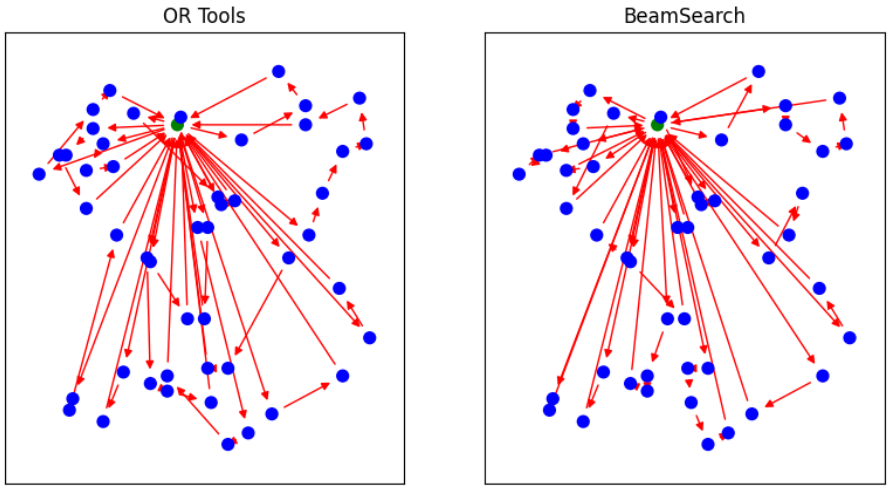
Rysunek 5.12. Wielkość problemu 25, jakość znalezionego rozwiązania 118,92%



Rysunek 5.13. Wielkość problemu 25, jakość znalezionego rozwiązania 100,13%



Rysunek 5.14. Wielkość problemu 49, jakość znalezionego rozwiązania 120,69%



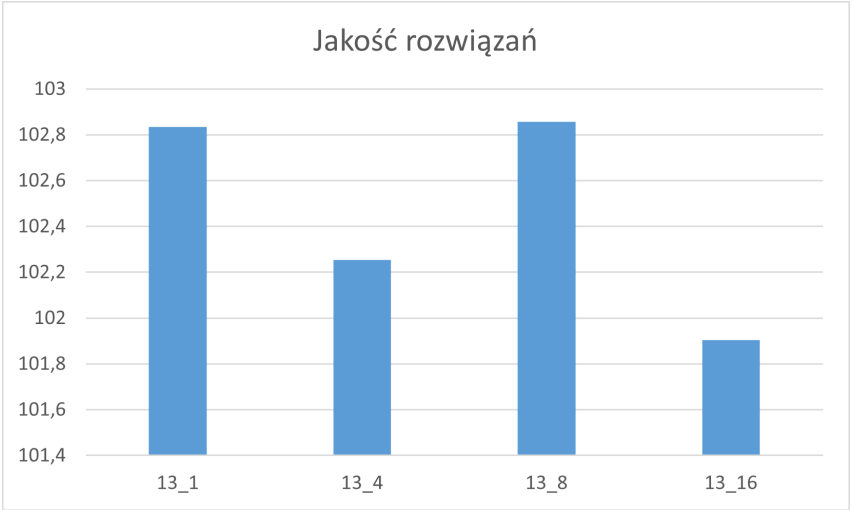
Rysunek 5.15. Wielkość problemu 49, jakość znalezionego rozwiązania 99,12%

5.3. Przeszukiwanie drzewa Monte-Carlo

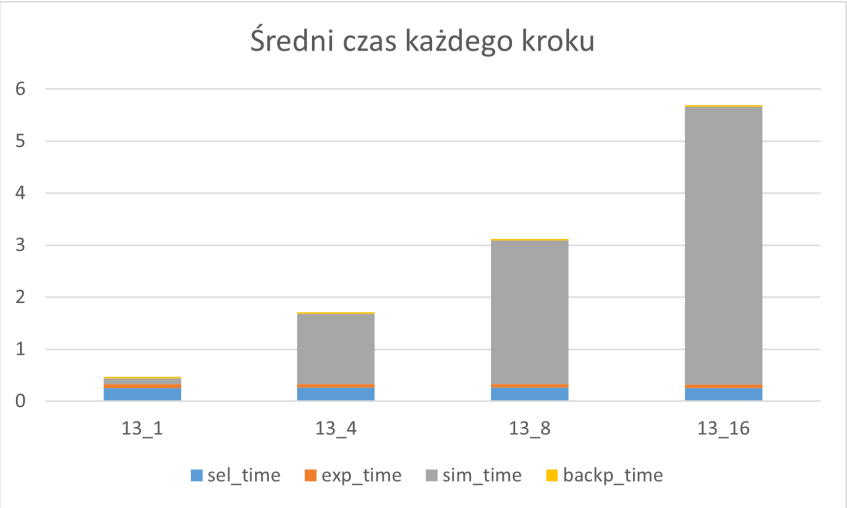
W przypadku testu tego algorytmu zbadane zostały 4 wartości szerokości promienia dla etapu symulacji: 1, 4, 8 oraz 16. Ponieważ z poprzednich testów wywnioskowaliśmy, że przeszukiwanie wiązkowe jest dużo gorsze pod kątem czasowym od algorytmu zachłannego przy szerokości promienia równej 1, w tym teście nie będziemy już tego sprawdzać i dla szerokości równej 1 będziemy używać algorytmu zachłannego. Na kolejnych stronach przedstawione zostały tabele oraz wykresy podsumowujące wyniki:

Wielkość	Szerokość promienia	Średni czas (s)	MAX (%)	MIN (%)	Średnia (%)
13	1 (zachłanny)	0,46957	117,40	85,52	102,83
13	4	1,71116	119,69	84,56	102,25
13	16	3,11964	130,37	85,21	102,86
13	64	5,69477	114,47	87,67	101,90
25	1 (zachłanny)	3,37657	112,33	100,27	104,51
25	4	52,70240	127,58	99,99	106,86
25	16	104,06804	123,33	100,52	108,13
25	64	205,04975	121,08	100	106,21
49	1 (zachłanny)	52,90077	110,29	97,86	104,43
49	4	2413,22258	123,66	102,60	109,24
49	16	4466,47303	120,29	105,22	112,63
49	64	8625,76180	127,82	105,49	115,92

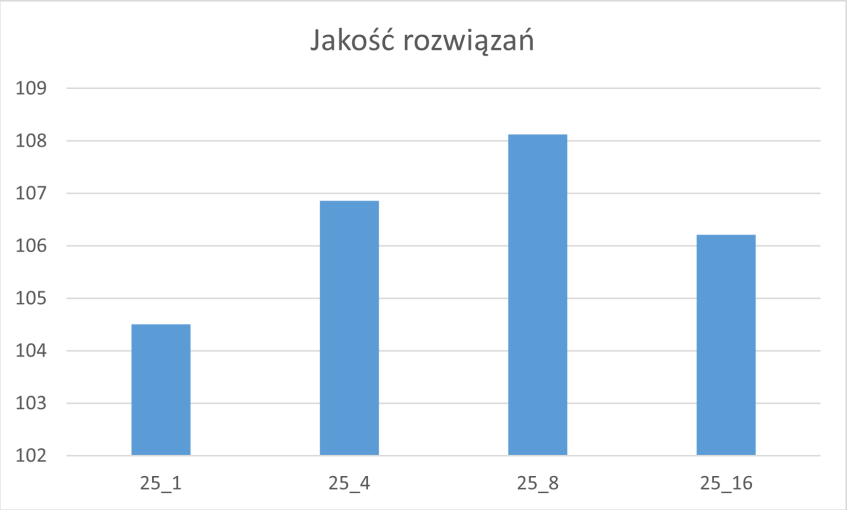
Tabela 5.3. Wyniki dla MCTS



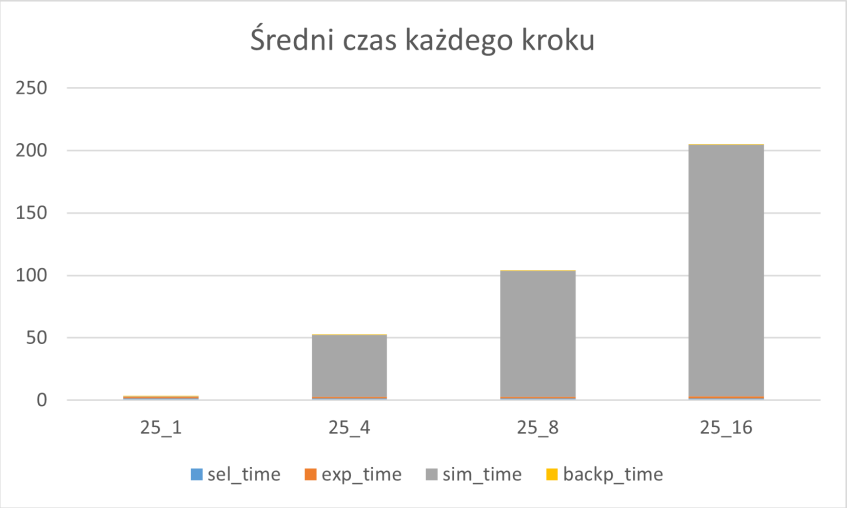
Rysunek 5.16. Jakość rozwiązań w zależności szerokości promienia (13 wierzchołków)



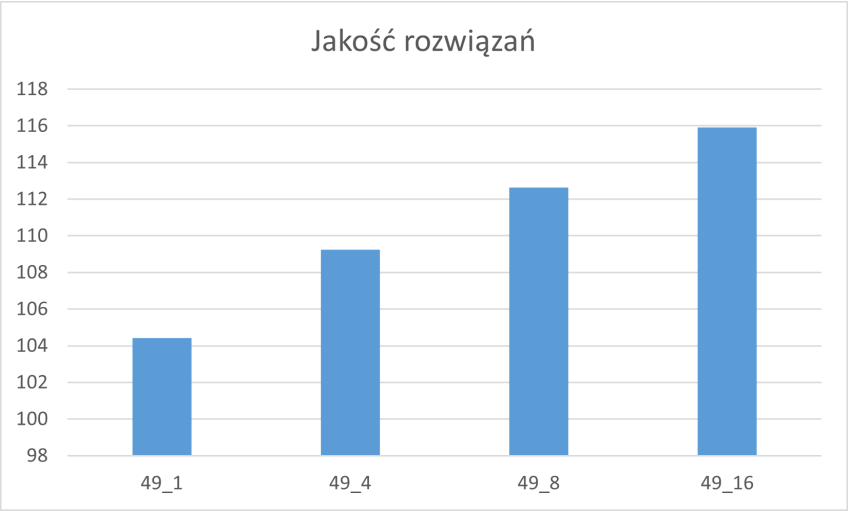
Rysunek 5.17. Średni czas każdego etapu algorytmu (13 wierzchołków)



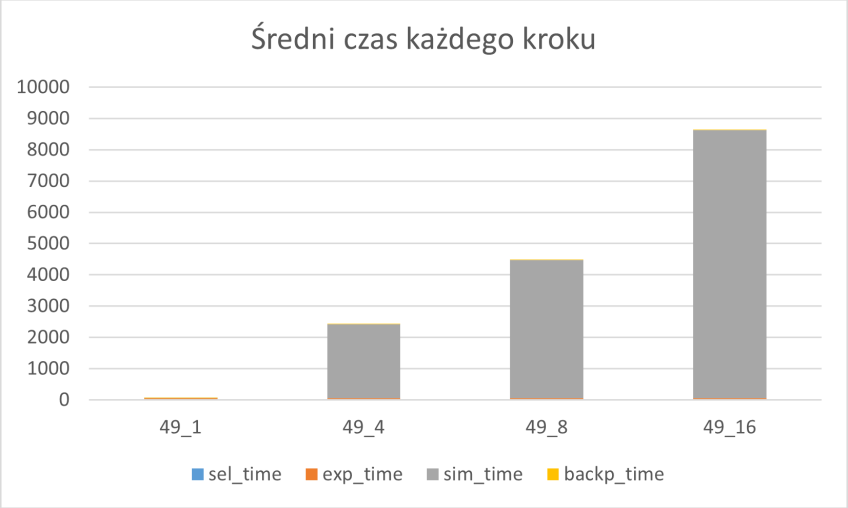
Rysunek 5.18. Jakość rozwiązań w zależności szerokości promienia (25 wierzchołków)



Rysunek 5.19. Średni czas każdego etapu algorytmu (25 wierzchołków)



Rysunek 5.20. Jakość rozwiązań w zależności szerokości promienia (49 wierzchołków)



Rysunek 5.21. Średni czas każdego etapu algorytmu (49 wierzchołków)

Ponieważ testy dla szerokości promienia większej od 1 i wielkości problemu 49 zajmowały dużo więcej czasu niż przewidywano, zdecydowaliśmy się na zmniejszenie próbki testowej danych ze 100 przypadków na 20 dla ostatnich 3 testów.

Na podstawie wygenerowanych wyników można wyciągnąć następujące wnioski:

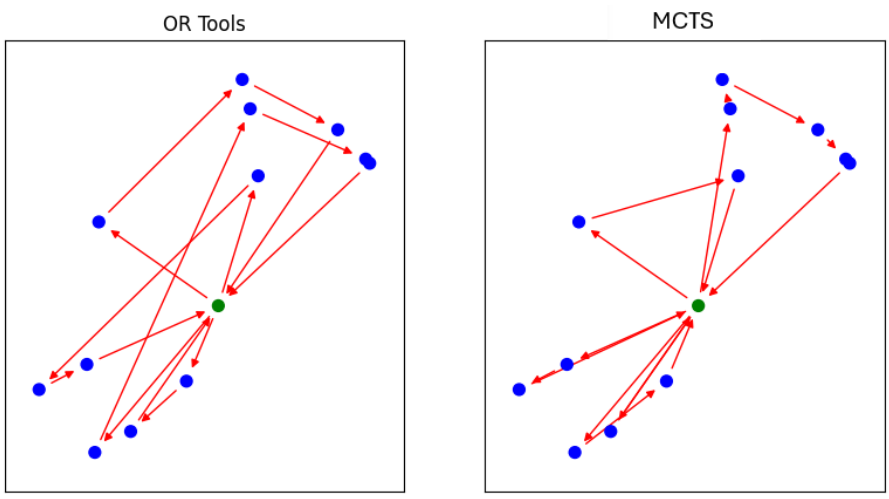
— W przeciwieństwie do przeszukiwania wiązkowego, zwiększanie szerokości promienia w algorytmie przeszukiwania drzewa Monte-Carlo nie zawsze przynosi lepsze rezultaty, a nawet wtedy poprawa wyników nie jest tak duża jak w przypadku pierwszego algorytmu (poprawa o około 2% w przypadku 13 wierzchołków, pogorszenie się wyników o 14% dla 49 wierzchołków, w przypadku przeszukiwania wiązkowego była to poprawa o 15% dla 13 wierzchołków oraz poprawa o 9% dla 49 wierzchołków).
— Zwiększanie szerokości promienia wprost proporcjonalnie zwiększa czas potrzebny na znalezienie rozwiązania, ale jest kilkakrotnie większy niż w przypadku przeszukiwania wiąz-

kowego (średni czas potrzeby na rozwiązanie jednego przypadku 49_16 (49 wierzchołków, szerokość promienia 16) zajął ponad 2,5 godziny). Jest to spowodowane wielokrotnym wywołaniem przeszukiwania wiązkowego (tyle samo wywołań ile iteracji, 800 dla problemów 13 i 25 oraz 1200 dla problemów 49).

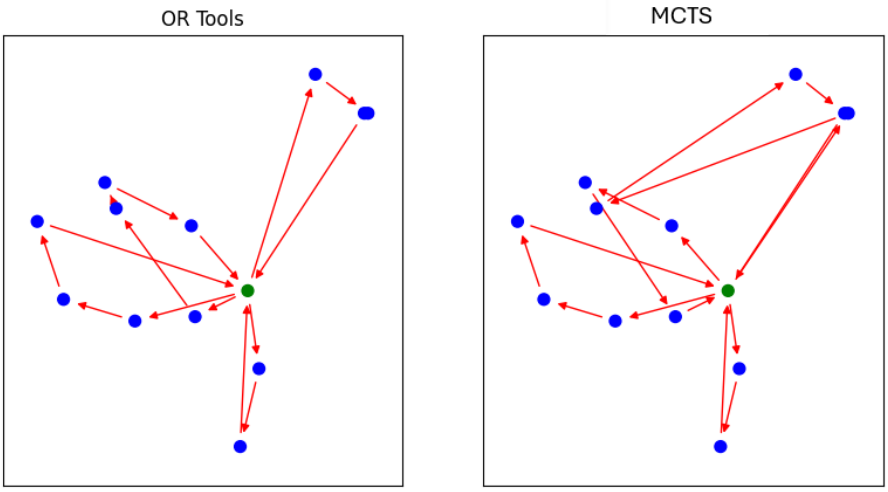
— Dla każdej wielkości problemu udało się poprawić średnią jakość rozwiązań w porównaniu do testów pierwszych algorytmów.

— Ponieważ zwiększanie szerokości promienia nie polepsza znacząco wyników, ale znacząco zwiększa czas, najlepszym wyjściem będzie zastosowanie w fazie symulacji algorytmu zachłannego zamiast przeszukiwania wiązkowego (średni czas polepsza się ponad 100 krotnie, 52,9s dla przypadku 49_1 w przeciwieństwie do 8625,76s dla przypadku 49_64).

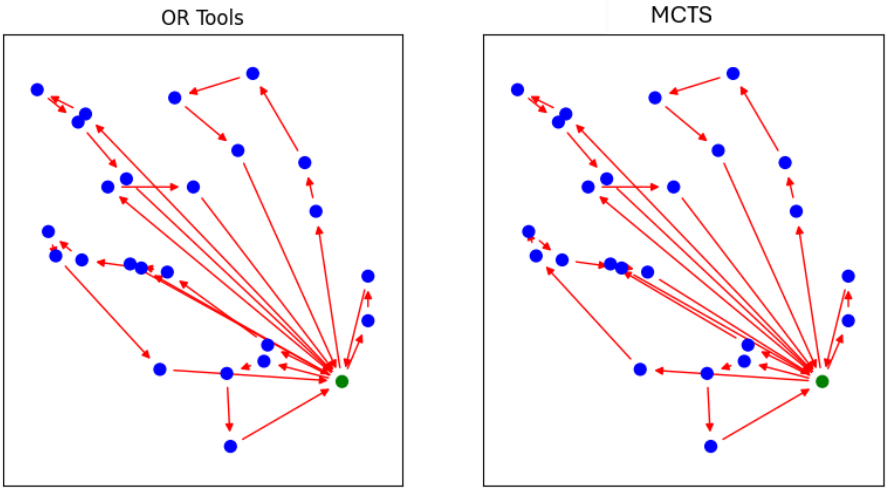
Kolejne rysunki przedstawiają wygenerowane trasy dla najlepszego i najgorszego przypadku znalezionego przez algorytm przeszukiwania drzewa Monte-Carlo. Po lewej stronie przedstawione zostało rozwiązanie znalezione przez OR-Toolsy, po prawej znalezione przez rozpatrywany algorytm.



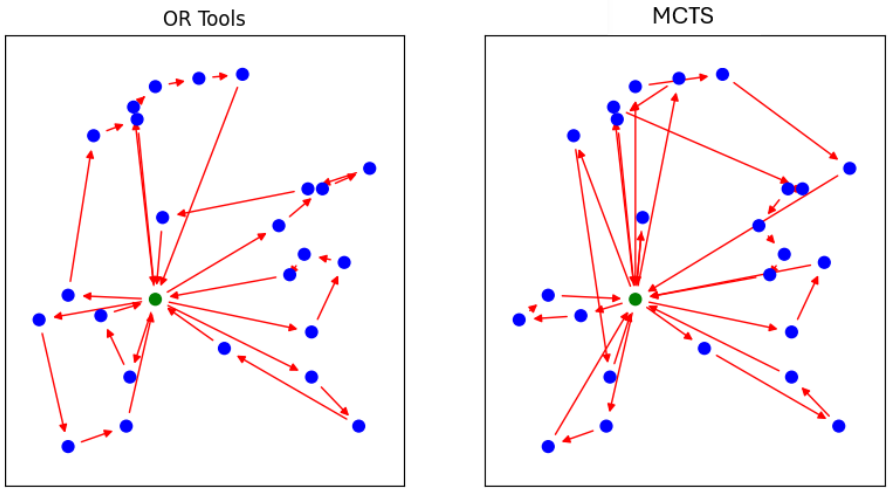
Rysunek 5.22. Przypadek 13_4, jakość znalezionego rozwiązania 84,56%



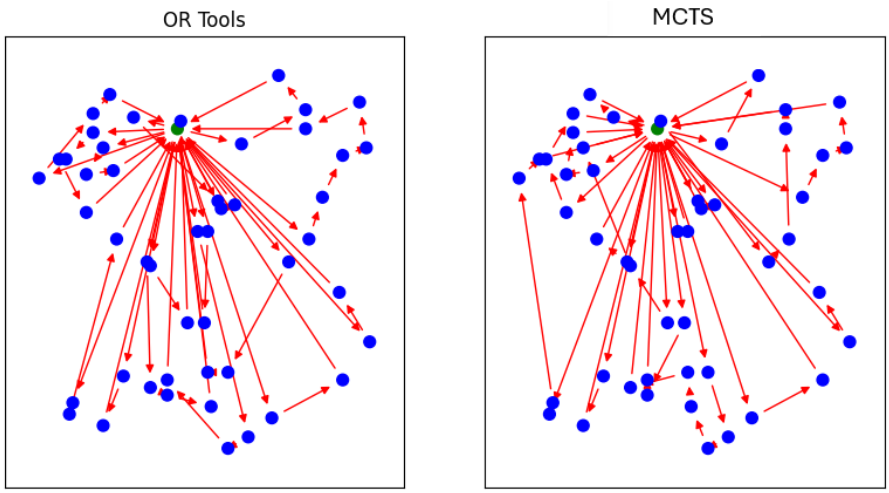
Rysunek 5.23. Przypadek 13_8, jakość znalezionego rozwiązania 130,37%



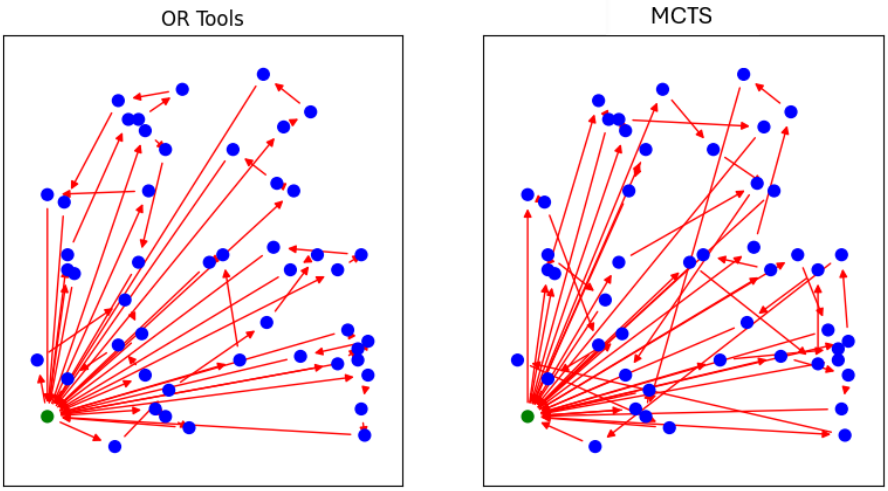
Rysunek 5.24. Przypadek 25_4, jakość znalezionego rozwiązania 99,99%



Rysunek 5.25. Przypadek 25_4, jakość znalezionego rozwiązania 127,58%



Rysunek 5.26. Przypadek 49_1, jakość znalezionego rozwiązania 97,86%



Rysunek 5.27. Przypadek 49_16, jakość znalezionego rozwiązania 127,82%

5.4. Wiązkowe przeszukiwanie drzewa Monte-Carlo

Ostatnie eksperymenty zostały przeprowadzone na algorytmie wiązkowego przeszukiwania drzewa Monte-Carlo. Na podstawie wyników badań wcześniejszego algorytmu, zdecydowaliśmy się użyć algorytmu zachłannego w fazie symulacji. Na kolejnych stronach przedstawione zostały tabele oraz wykresy podsumowujące wynik:

Limit symulacji	Szerokość promienia	Średni czas (s)	MAX (%)	MIN (%)	Średnia (%)
10	1	0,36107	117,40	85,52	103,86
10	4	0,36409	117,40	85,52	104,19
10	16	0,36597	117,40	85,52	104,08
10	64	0,36753	117,40	85,52	104,08
20	1	0,38037	117,40	85,52	102,83
20	4	0,38105	117,40	85,52	102,83
20	16	0,37940	117,40	85,52	102,83
20	64	0,37920	117,40	85,52	102,83
40	1	0,38381	117,40	85,52	102,83
40	4	0,38096	117,40	85,52	102,83
40	16	0,38214	117,40	85,52	102,83
40	64	0,38156	117,40	85,52	102,83
80	1	0,38777	117,40	85,52	102,83
80	4	0,38383	117,40	85,52	102,83
80	16	0,38695	117,40	85,52	102,83
80	64	0,38587	117,40	85,52	102,83

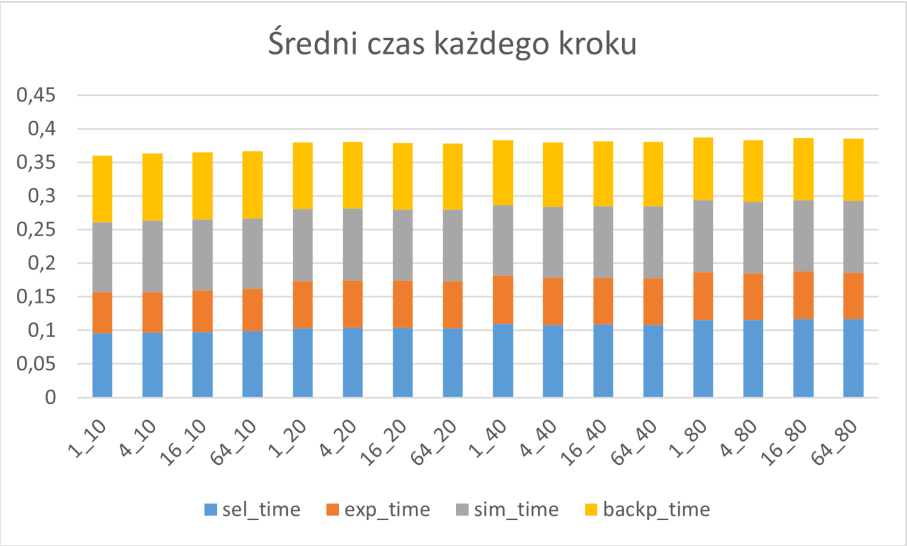
Tabela 5.4. Wyniki dla 13 wierzchołków

Limit symulacji	Szerokość promienia	Średni czas (s)	MAX (%)	MIN (%)	Średnia (%)
10	1	1,91466	133,89	101,10	112,30
10	4	1,92474	133,81	102,03	112,14
10	16	1,94749	133,89	101,10	112,14
10	64	2,01919	133,89	101,10	112,14
20	1	2,60116	115,17	100,27	105,47
20	4	2,60425	115,17	100,27	105,44
20	16	2,64966	115,17	100,27	105,39
20	64	2,68644	115,17	100,27	105,47
40	1	2,94953	112,33	100,27	104,51
40	4	2,93022	112,33	100,27	104,51
40	16	2,90524	112,33	100,27	104,51
40	64	2,83249	112,33	100,27	104,51
80	1	2,88507	112,33	100,27	104,51
80	4	2,84673	112,33	100,27	104,51
80	16	2,81783	112,33	100,27	104,51
80	64	2,81645	112,33	100,27	104,51

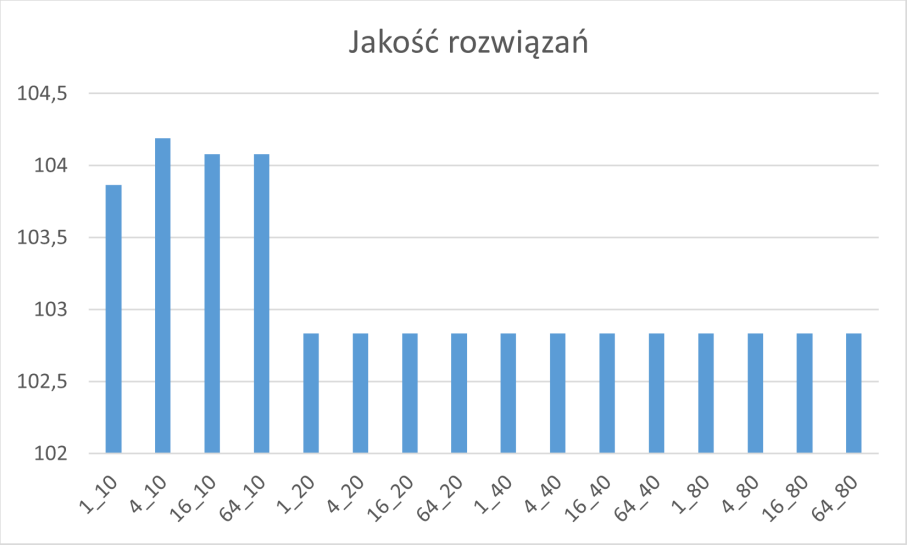
Tabela 5.5. Wyniki dla 25 wierzchołków

Limit symulacji	Szerokość promienia	Średni czas (s)	MAX (%)	MIN (%)	Średnia (%)
10	1	19,01843	141,02	103,31	123,53
10	4	21,14750	144,99	104,01	124,18
10	16	24,17553	153,09	103,80	124,01
10	64	24,07546	153,09	103,80	124,01
20	1	28,87209	130,90	98,94	109,61
20	4	26,98976	130,90	100,70	110,09
20	16	27,45774	130,90	99,10	109,52
20	64	27,44603	130,90	98,94	109,48
40	1	43,01193	110,47	97,86	104,79
40	4	42,82009	110,47	97,86	104,73
40	16	42,96755	110,47	97,86	104,72
40	64	43,90135	110,47	97,86	104,79
80	1	52,60267	110,29	97,86	104,43
80	4	52,37008	110,29	97,86	104,43
80	16	51,98763	110,29	97,86	104,43
80	64	51,90360	110,29	97,86	104,43

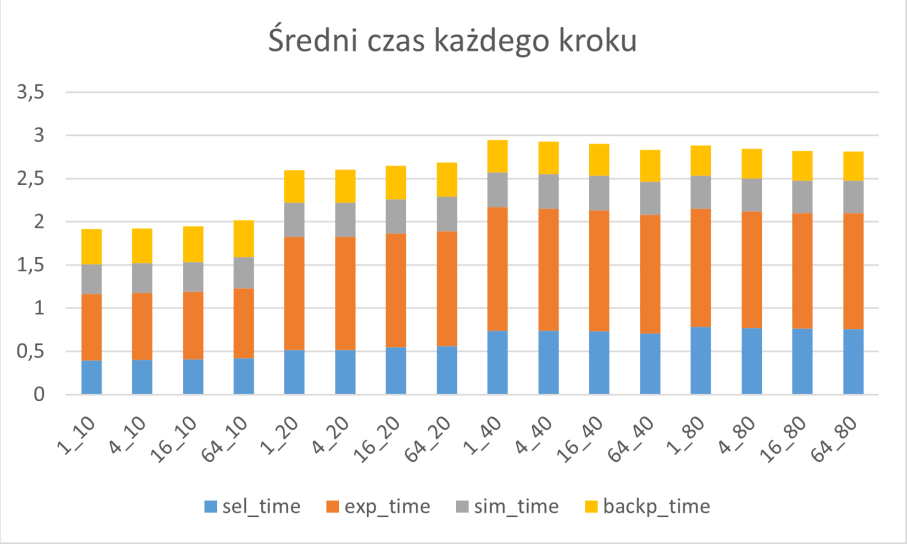
Tabela 5.6. Wyniki dla 49 wierzchołków



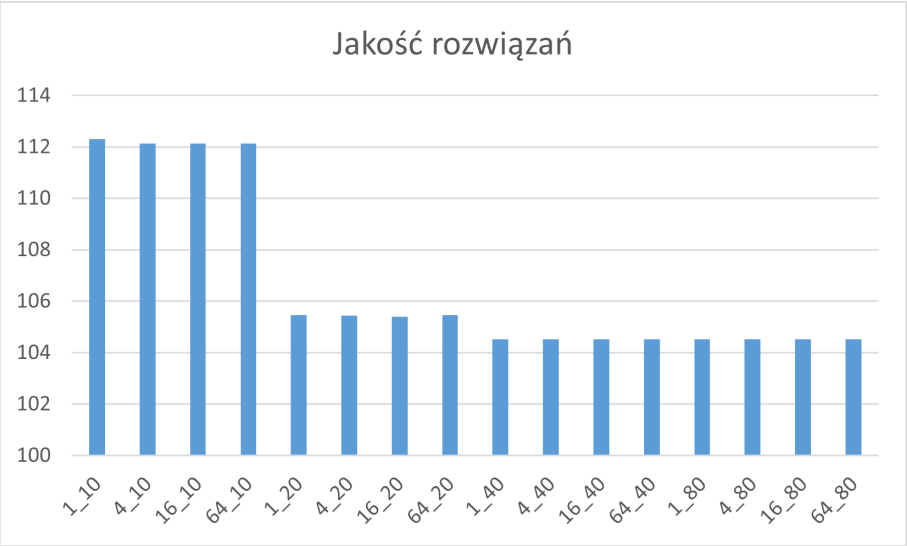
Rysunek 5.28. Średni czas każdego kroku, 13 wierzchołków



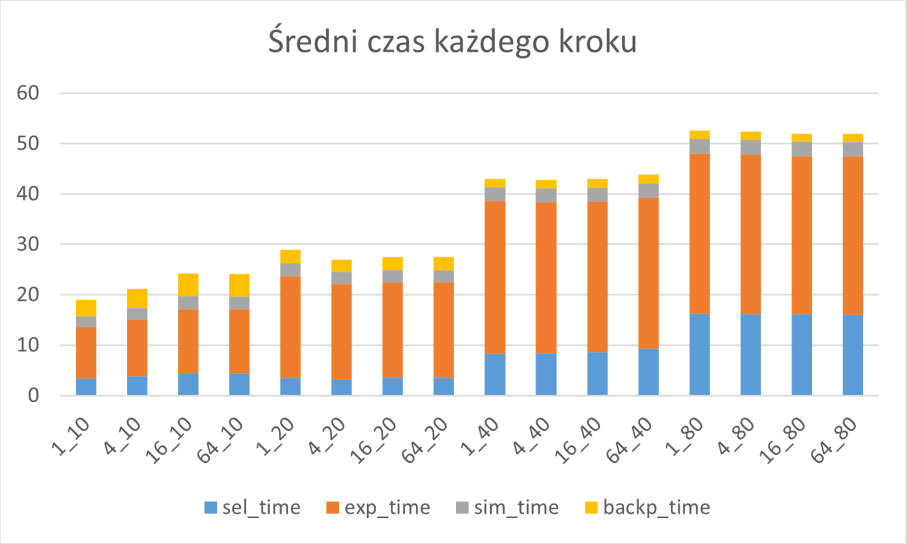
Rysunek 5.29. Jakość rozwiązań, 13 wierzchołków



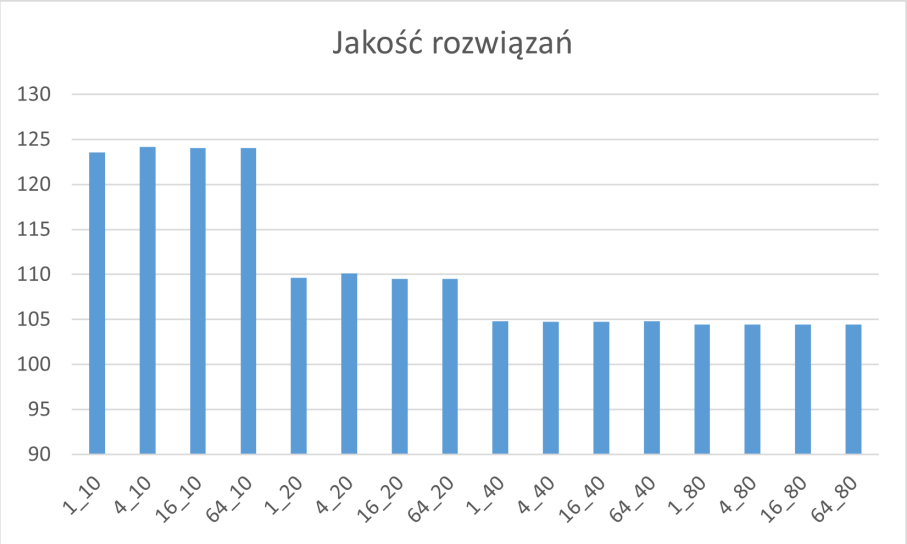
Rysunek 5.30. Średni czas każdego kroku, 25 wierzchołków



Rysunek 5.31. Jakość rozwiązań, 25 wierzchołków



Rysunek 5.32. Średni czas każdego kroku, 49 wierzchołków



Rysunek 5.33. Jakość rozwiązań, 49 wierzchołków

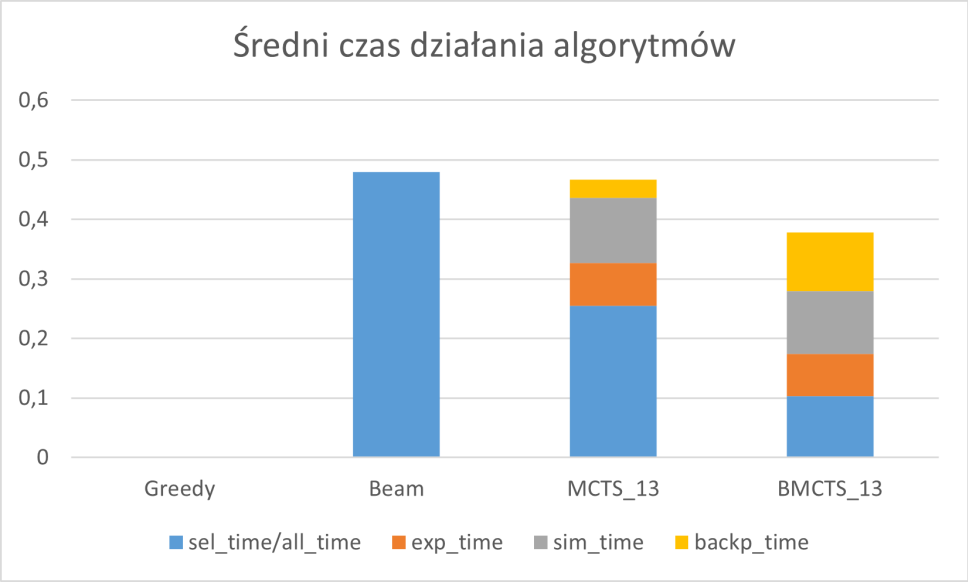
Na podstawie rysunku 5.28 można zobaczyć, że badane parametry mają znikomy wpływ na czas działania algorytmu dla 13 wierzchołków. Z rysunku 5.29 widać natomiast, że zbyt mała wartość limitu symulacji wpływa gwałtownie na jakość otrzymanych rozwiązań. Z kolejnych wykresów widać, że wraz ze wzrostem złożoności problemu granica otrzymywanych jakości rozwiązań przesuwa się. W każdym z przypadków widać, że szerokość promienia ucinania drzewa ma dużo mniejszy wpływ niż limit symulacji. Tylko w przypadku 13 wierzchołków można zauważyć większe wahania jakości rozwiązania przy początkowych zmianach szerokości promienia. Na podstawie rysunku 5.33 widać, że w przypadku 49 wierzchołków najlepsze wyniki zaczęliśmy dostawać przy limicie 40 symulacji. Jak widać osiągamy wtedy wyniki w okolicach 104%, co wynosi tyle samo co najlepsza średnia dla testów MCTS, przy jednoczesnej poprawie czasu działania z 52 sekund do 43 sekund. Dla 13 i 25 wierzchołków najlepszy limit symulacji wyniósł 20 przy czym poprawa czasu działania nie jest tak zauważalna. Jak widać przy ustawieniu wystarczająco dużej wartości badanych parametrów czas jak i jakość rozwiązań dąży do tych samych wartości co te osiągane przez algorytm MCTS.

5.5. Porównanie algorytmów

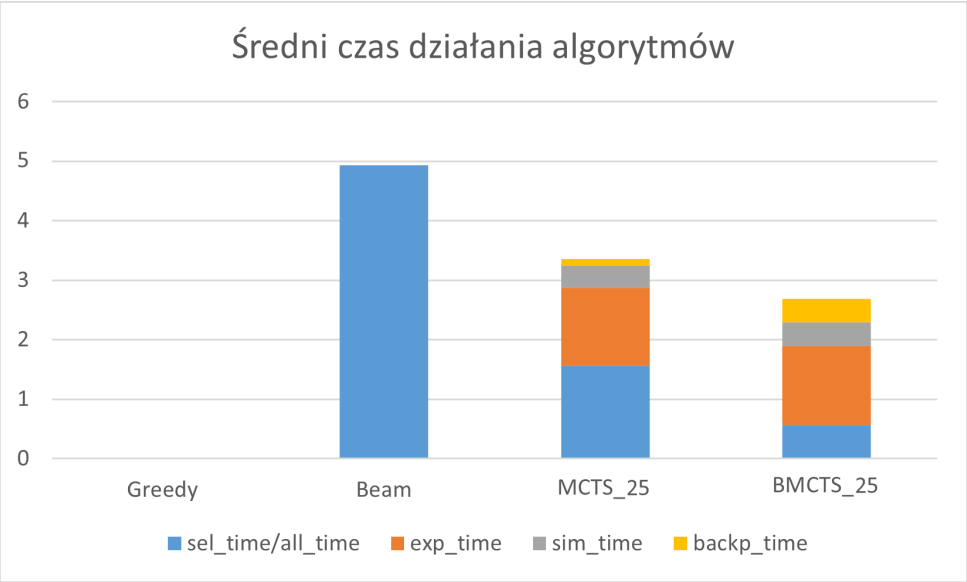
W tym rozdziale porównane będą wyniki i czas działania rozpatrywanych algorytmów. Dla przeszukiwania wiązkowego przedstawiony zostanie przypadek z szerokością promienia 64. Dla algorytmu MCTS oraz BMCTS przedstawiony zostanie przypadek z użyciem algorytmu zachłannego na etapie symulacji. Dodatkowo dla BMCTS zostanie użyta szerokość promienia ucinania drzewa 64 oraz limit symulacji 20 dla 13 i 25 wierzchołków oraz 40 dla 49 wierzchołków.

Algorytm	Wielkość	Średni czas (s)	MAX (%)	MIN (%)	Średnia (%)
Zachłanny	13	0,00026	156,01	96,83	118,86
Przeszukiwanie wiązkowe	13	0,47976	118,76	84,84	103,27
MCTS	13	0,46957	117,40	85,52	102,83
BMCTS	13	0,37920	117,40	85,52	102,83
Zachłanny	25	0,00089	143,11	104,76	121,37
Przeszukiwanie wiązkowe	25	4,92985	118,92	100,13	107,07
MCTS	25	3,37657	112,33	100,27	104,51
BMCTS	25	2,68644	115,17	100,27	105,47
Zachłanny	49	0,00368	135,53	107,83	119,76
Przeszukiwanie wiązkowe	49	60,29774	120,69	99,12	110,02
MCTS	49	52,90077	110,29	97,86	104,43
BMCTS	49	43,90135	110,47	97,86	104,79

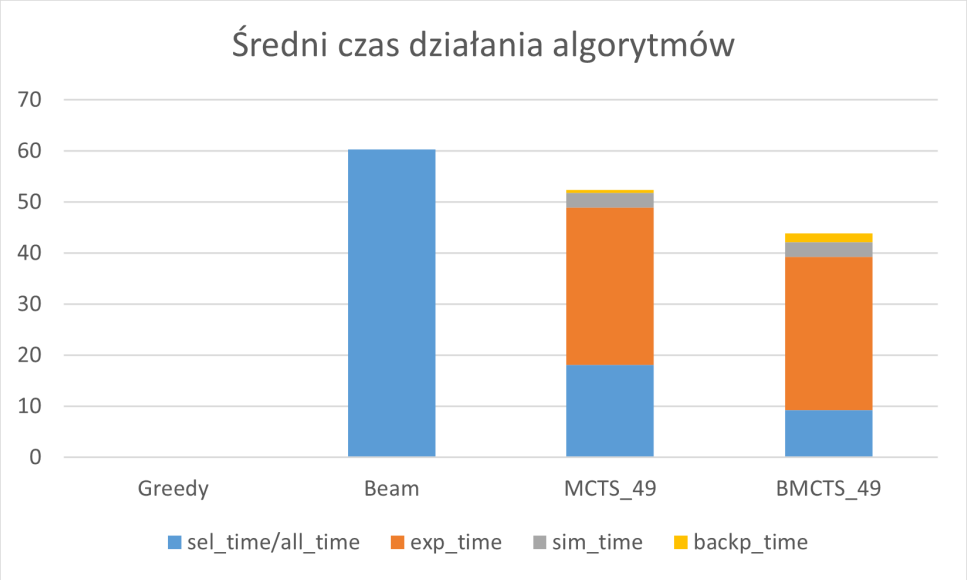
Tabela 5.7. Porównanie algorytmów



Rysunek 5.34. Średni czas działania algorytmów, 13 wierzchołków



Rysunek 5.35. Średni czas działania algorytmów, 25 wierzchołków



Rysunek 5.36. Średni czas działania algorytmów, 49 wierzchołków

W tabeli 5.7 przedstawiono wyniki każdego algorytmu dla najlepszych zestawów parametrów pod kątem czasu działania i jakości otrzymanych wyników. Na podstawie tabeli i rysunków można wyciągnąć następujące wnioski:

— Algorytm zachłanny jest najszybszym, ale najmniej precyzyjnym algorytmem, dla każdej wielkości problemu średnie odchylenie od OR-Tools wynosi 120%. Mimo to w przypadku, gdy chce się otrzymać wyniki jak najszybciej użycie tego algorytmu będzie najlepszym rozwiązaniem.
— Przeszukiwanie wiązkowe z szerokością promienia 64 daje gorsze wyniki w gorszym czasie niż algorytmy MCTS oraz BMCTS.
— Zastosowanie odpowiedniej wartości limitu symulacji sprawia że algorytm BMCTS jest

około 20% szybszy niż algorytm MCTS, a daje prawie te same wyniki (różnica w średnich odchyleniach mniejsza niż 1%). Jest to szczególnie znaczące dla większych wielkości problemów.

— Z rysunków widać, że modyfikacje wprowadzone w algorytmie BMCTS wpłynęły znacząco na skrócenie czasu etapu selekcji, ale wydłużyły czas etapu propagacji wstecznej.

— Biorąc pod uwagę czas potrzebny na osiągnięcie rozwiązań przez OR-Toolsy (3s, 15s, 60s dla kolejnych wielkości problemu) rozpatrywane algorytmy, a zwłaszcza BMCTS mogą być użyteczną alternatywą dla rozwiązywania problemów wyznaczania tras.

6. Zakończenie

Analiza zagadnienia algorytmów rozwiązujących problem wyznaczania tras pojazdów z ograniczoną pojemnością każdego pojazdu jest istotna z punktu widzenia opłacalności. W pracy została zaprojektowana grafowa sieć neuronowa i cztery algorytmy rozwiązujące problem wyznaczania tras pojazdów. Pierwszy z nich to algorytm zachłanny, który generuje rozwiązanie na podstawie macierzy predykcji wygenerowanej przez sieć. Drugim algorytmem jest przeszukiwanie wiązkowe, który jest rozwinięciem poprzedniego algorytmu poprzez dodanie pamięci o więcej niż jednym rozwiązaniu. Trzecim algorytmem jest algorytm przeszukiwania drzewa Monte-Carlo, wzorowany na pracy z [4]. Ostatni algorytm jest modyfikacją opracowanego algorytmu MCTS poprzez dodanie ucinania przeszukiwanego drzewa, dodając parametr limitu symulacji oraz szerokości promienia ucinanego drzewa tak jak zaproponowano w pracy [6]. Przeprowadzone zostały badania wszystkich algorytmów na wygenerowanych zestawach danych. Zbadany został czas, jakość znajdywanych rozwiązań w porównaniu do rozwiązań z OR-Toolsów oraz wpływ na te wartości parametrów charakterystycznych algorytmów. Sprawdzony został także średni czas rozwiązywania zadań w zależności od użytego algorytmu oraz wielkości problemu.

Badania wykazały, że algorytmy MCTS oraz BMCTS dawały zdecydowanie lepsze wyniki niż algorytm zachłanny i przeszukiwanie wiązkowe. Zaobserwowaliśmy również, że w przeciwieństwie do algorytmu z pracy [4], na etapie symulacji dużo bardziej opłaca się użyć algorytmu zachłannego niż przeszukiwania wiązkowego, ponieważ jest dużo szybszy w działaniu (dla 13 wierzchołków prawie 38 razy szybszy, dla 49 wierzchołków 374 razy szybszy), a z powodu struktury algorytmu na etapie symulacji nie potrzebujemy bardzo precyzyjnych wyników tylko przybliżonych (oryginalny MCTS dla gier symulował wyniki gry poprzez losowe wykonywanie ruchów przez obu graczy). Modyfikacja algorytmu MCTS do BMCTS sprawia, że jesteśmy w stanie lepiej kontrolować ile czasu zajmie algorytmowi znalezienie rozwiązania przy jednoczesnym zachowaniu jak najlepszej jakości wyników.

Algorytm zachłanny jest najprostszym oraz najszybszym algorytmem do rozwiązania problemu wyznaczania tras z wykorzystanie sieci neuronowych. Jednocześnie jest najbardziej zależny od jakości wygenerowanego modelu. Możemy się spodziewać, że dla lepiej wyszkolonych sieci algorytm zachłanny będzie dawał również dużo lepsze wyniki. Niestety obecnie nie ma wystarczająco dużych zbiorów testowych z wygenerowanymi optymalnymi rozwiązaniami, w ramach tej pracy w 9 dni wygenerowaliśmy 10 000 przypadków (nie wszystkie są optymalne, ponieważ korzystaliśmy z narzędzia OR-Tools generującego przybliżone rozwiązania). Dla problemów komiwojażera sieci uczone są na milionach przypadków (istnieją już wygenerowane optymalne

rozwiązania „Concorde TSP Solver").

Przeszukiwanie wiązkowe wraz ze wzrostem szerokości promienia daje lepsze rozwiązania niż algorytm zachłanny. Zaletą tego algorytmu jest to, że wraz ze wzrostem szerokości promienia mamy coraz większe szanse na znalezienie optymalnego rozwiązania, nawet jeśli model nie jest najlepszy. Wadą jest proporcjonalny wzrost czasu działania algorytmu do szerokości promienia przez co w większości rzeczywistych przypadków nie opłaca się korzystać z tego algorytmu w celu znalezienia optymalnego rozwiązania.

Lepszą alternatywą od przeszukiwania wiązkowego okazał się algorytm przeszukiwania drzewa Monte-Carlo (MCTS). Jak widać w rozdziale porównującym algorytmy MCTS dawał lepsze wyniki, niż najlepsze wyniki wygenerowane przez przeszukiwanie wiązkowe, jednocześnie zachowując lepszy czas działania. Modyfikacja tego algorytmu do wersji wiązkowego przeszukiwania drzewa Monte-Carlo (BMCTS) pozwoliła skrócić czas działania tego algorytmu o 20% pogarszając średnią wyników o mniej niż 1%. Oba algorytmy wydają się lepiej działać w przypadku gorzej wytrenowanych sieci niż algorytm zachłanny i przeszukiwanie wiązkowe, co wynika ze struktury tych algorytmów (oryginalny algorytm MCTS zakłada pewną losowość w generowaniu wyników na etapie symulacji).

Tematyka wykorzystania głębokiego uczenia do rozwiązywania problemów kombinatorycznych jest wciąż obszarem słabo zbadanym, co stanowi pole do dalszych badań. W przypadku większej dostępnej mocy obliczeniowej oraz większej ilości czasu, warto byłoby zgłębić wpływ różnych stopni wytrenowania sieci na efektywność przedstawionych algorytmów. Dodatkowo, istotnym kierunkiem przyszłych badań byłaby generacja bardziej zróżnicowanych danych testowych. Niestety, w trakcie realizacji tej pracy nie udało się znaleźć odpowiednich, publicznych zbiorów danych, które mogłyby zostać wykorzystane do dalszych eksperymentów, co ograniczyło liczbę przypadków w treningu sieci do 10 000 na rozmiar problemu. Dla porównania, w literaturze dotyczącej problemu komiwojażera zbiorów danych zazwyczaj liczy się w milionach przypadków, co pokazuje potencjał dalszego rozwoju tego obszaru badań. Innym kierunkiem prac który bym proponował to sprawdzenie różnych modyfikacji grafowej sieci neuronowej i wpływ tych modyfikacji na proces uczenia. Ciekawym rozwinięciem mogłoby być sprawdzenie jak zmieniło by się zachowanie sieci w przypadku zmiany sposobu liczenia funkcji straty np. poprzez stworzenie własnej funkcji przystosowanej do rozpatrywanego problemu.

Spis zawartości repozytorium

Repozytorium ”https://github.com/dstalews/Mgr_CVRP_GNN_DS” zawiera następujące katalogi:

\TXT – niniejsza praca dyplomowa magisterska

\SRC – katalog zawierający kod programów oraz dane testowe

\DATA – wyniki testów

Bibliografia

[1] G.B. Dantzig, J.H. Ramser: The Truck Dispatching Problem, Management Science, Tom 6, s. 80–91, 1959.

[2] J. Lenstra, K. Rinnooy: Complexity of vehicle routing and scheduling problems, Networks, Tom 11, s. 221-227, 1981.

[3] R. Baldacci, A. Mingozzi, R. Roberti: Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints, European Journal of Operational Research, Tom 218, s. 1-6, 2012.

[4] Z. Xing, S. Tu: GNN Assisted MCTS Approach to TSP, IEEE Access, Tom 8, s. 108418 - 108428, 2020.

[5] F. Hagström: Finding Solutions to the Vehicle Routing Problem using a Graph Neural Network, School of Science , s. 6–29, 2022.

[6] H. Baier, M. Winands: Beam Monte-Carlo Tree Search, 10.1109/CIG.2012.6374160, s. 227-233. 2012.

[7] I. Drori, A. Kharkar, W. R. Sickinger, B. Kates, Q. Ma, S. Ge, E. Dolev, B. Dietrich, D. P. Williamson, M. Udell: Learning to solve combinatorial optimization problems on real-world graphs in linear time. In 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE, Tom 8, s. 19-24, 2020.

[8] Z. Li, Q. Chen, V. Koltun: Combinatorial optimization with graph convolutional networks and guided tree search, Advances in neural information processing systems 31, s. 1-13, 2018.

[9] K. Xu, J. Li, M. Zhang, S. S. Du, K. Kawarabayashi, S. Jegelka: What can neural networks reason about, arXiv preprint arXiv:1905.13211, s. 1-18. 2019.

[10] J.E. Bell, P.R. McMullen: Ant colony optimization techniques for the vehicle routing problem, Advanced Engineering Informatics, tom 18, s. 41-48, 2004.

[11] F. Hagström: Finding Solutions to the Vehicle Routing Problem using a Graph Neural Network, Aalto University, School of Science, 2022.

[12] M. M. Solomon: Algorithms for vehicle-routing and scheduling problems with time window

constraints, Operations Research, tom 35, s. 254-265, 1987.

[13] J. Potvin, J. Rousseau: A parallel route building algorithm for the vehicle routing and scheduling problem with time windows, European Journal of Operational Research, tom 66, s. 331-340, 1993.

[14] G. Loannou, M. Kritikos, G. Prastacos: A greedy look-ahead heuristic for the vehicle routing problem with time windows, Journal of the Operational Reasearch Society, tom 52, s. 523-537, 2001.

[15] A. Imran, S. Salhi, N. A. Wassan: A variable neighborhood-based heuristic for the heterogeneous fleet vehicle routing problem, European Journal of Operational Research, tom 197, s. 509-518, 2008.

[16] K. C. Tan, L. H. Lee, Q. L. Zhu, K. Ou: Heuristic methods for vehicle routing problem with time windows, Artificial Intelligence in Engineering, tom 15, s. 281-295, 2000.

Spis rysunków

3.1. Caption for LOF . . . 13
3.2. Caption for LOF . . . 13
3.3. Caption for LOF . . . 14
3.4. Caption for LOF . . . 15
3.5. Schemat działania sieci neuronowej . . . 17
3.6. Kolejne fazy przeszukiwania wiązkowego, dla k = 3 . . . 19

5.1. Funkcja straty (wielkość problemu 13, warstwy ukryte 100) . . . 27
5.2. Funkcja straty (wielkość problemu 13, warstwy ukryte 200) . . . 27
5.3. Funkcja straty (wielkość problemu 13, warstwy ukryte 300) . . . 28
5.4. Funkcja straty (wielkość problemu 25, warstwy ukryte 100) . . . 28
5.5. Funkcja straty (wielkość problemu 25, warstwy ukryte 200) . . . 29
5.6. Funkcja straty (wielkość problemu 25, warstwy ukryte 300) . . . 29
5.7. Funkcja straty (wielkość problemu 49, warstwy ukryte 100) . . . 30
5.8. Funkcja straty (wielkość problemu 49, warstwy ukryte 200) . . . 30
5.9. Funkcja straty (wielkość problemu 49, warstwy ukryte 300) . . . 31
5.10. Wielkość problemu 13, jakość znalezionego rozwiązania 118,76% . . . 34
5.11. Wielkość problemu 13, jakość znalezionego rozwiązania 84,84% . . . 34
5.12. Wielkość problemu 25, jakość znalezionego rozwiązania 118,92% . . . 35
5.13. Wielkość problemu 25, jakość znalezionego rozwiązania 100,13% . . . 35
5.14. Wielkość problemu 49, jakość znalezionego rozwiązania 120,69% . . . 36
5.15. Wielkość problemu 49, jakość znalezionego rozwiązania 99,12% . . . 36
5.16. Jakość rozwiązań w zależności szerokości promienia (13 wierzchołków) . . . 37
5.17. Średni czas każdego etapu algorytmu (13 wierzchołków) . . . 38
5.18. Jakość rozwiązań w zależności szerokości promienia (25 wierzchołków) . . . 38
5.19. Średni czas każdego etapu algorytmu (25 wierzchołków) . . . 38
5.20. Jakość rozwiązań w zależności szerokości promienia (49 wierzchołków) . . . 39
5.21. Średni czas każdego etapu algorytmu (49 wierzchołków) . . . 39
5.22. Przypadek 13_4, jakość znalezionego rozwiązania 84,56% . . . 40
5.23. Przypadek 13_8, jakość znalezionego rozwiązania 130,37% . . . 41
5.24. Przypadek 25_4, jakość znalezionego rozwiązania 99,99% . . . 41
5.25. Przypadek 25_4, jakość znalezionego rozwiązania 127,58% . . . 42
5.26. Przypadek 49_1, jakość znalezionego rozwiązania 97,86% . . . 42
5.27. Przypadek 49_16, jakość znalezionego rozwiązania 127,82% . . . 43
5.28. Średni czas każdego kroku, 13 wierzchołków . . . 46
5.29. Jakość rozwiązań, 13 wierzchołków . . . 46
5.30. Średni czas każdego kroku, 25 wierzchołków . . . 46
5.31. Jakość rozwiązań, 25 wierzchołków . . . 47

5.32. Średni czas każdego kroku, 49 wierzchołków 47
5.33. Jakość rozwiązań, 49 wierzchołków 47
5.34. Średni czas działania algorytmów, 13 wierzchołków 49
5.35. Średni czas działania algorytmów, 25 wierzchołków 50
5.36. Średni czas działania algorytmów, 49 wierzchołków 50

Spis tabel

5.1. Średnia czasy epoki dla GNN 32
5.2. Wyniki dla przeszukiwania wiązkowego 33
5.3. Wyniki dla MCTS 37
5.4. Wyniki dla 13 wierzchołków 44
5.5. Wyniki dla 25 wierzchołków 45
5.6. Wyniki dla 49 wierzchołków 45
5.7. Porównanie algorytmów 49

Spis algorytmów

3.1 Pseudokod ilustrujący działanie algorytmu zachłannego dla problemu wyznaczania tras 18
3.2 Pseudokod ilustrujący działanie przeszukiwania wiązkowego dla problemu wyznaczania tras 20
3.3 Pseudokod ilustrujący działanie algorytmu przeszukiwania drzewa Monte-Carlo 21
3.4 Pseudokod ilustrujący działanie wiązkowego przeszukiwania drzewa Monte-Carlo 23
3.5 Pseudokod ilustrujący działanie funkcji UtnijDrzewo 23