Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **INA**
Specjalność: **brak**

PRACA DYPLOMOWA
INŻYNIERSKA

Analiza i porównanie algorytmów rozwiązujących problem komiwojażera

Maciej Sieroń

Opiekun pracy
dr Marcin Zawada

Słowa kluczowe: Problem komiwojażera, Algorytmy metaheurystyczne

WROCŁAW 2023

STRESZCZENIE

Celem pracy jest przegląd wybranych algorytmów rozwiązujących problem komiwojażera, eksperymentalny dobór ich parametrów oraz porównanie ich między sobą.

ABSTRACT

The aim of that thesis is to overview selected algorithms solving the travelling salesman problem, experimentally select their parameters and compare them with each other.

SPIS TREŚCI

WPROWADZENIE

CEL PRACY

Celem pracy jest przegląd wybranych algorytmów rozwiązujących problem komiwojażera, eksperymentalny dobór ich parametrów oraz porównanie ich między sobą.

ZAKRES PRACY

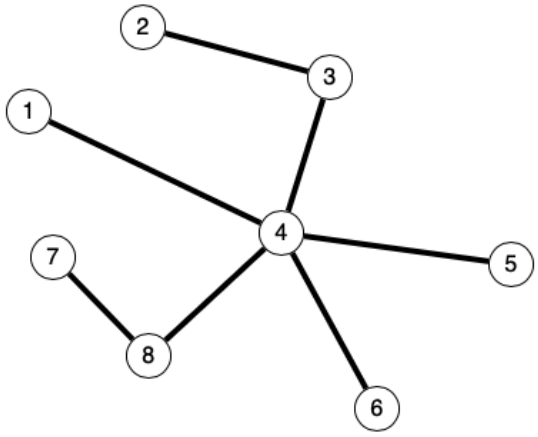
W pierwszych czterech rozdziałach opisane zostały kolejno podstawy teoretyczne, sam problem komiwojażera, wybrane algorytmy oraz sposób implementacji ich i programu, który został wykorzystany do przeprowadzenia badań. Następnie przedstawione zostały wyniki pierwszych badań, mających na celu dobór parametrów dla każdego z algorytmów. W ostatnim rozdziale zostały zaprezentowane oraz omówione wyniki kolejnych badań, w których algorytmy zostały porównane między sobą. Wykorzystane zostały tutaj dobrane wcześniej parametry.

1. PODSTAWY TEORETYCZNE

1.1. GRAF

1.1.1. Definicja grafu

Graf to para $G = (V, E)$, gdzie V to zbiór wierzchołków, E to zbiór krawędzi łączących je ze sobą. Wierzchołki są etykietowanymi punktami, najczęściej oznaczanymi za pomocą liczb całkowitych lub liter. Krawędzie to pary wierzchołków, które reprezentują połączenia między nimi. Na rysunku 1.1 pokazany został przykładowy graf [13].



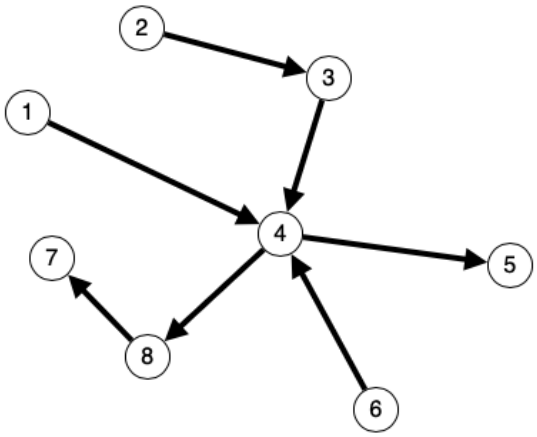
Rys. 1.1. Graf

Przedstawiony graf zawiera zbiór ośmiu wierzchołków $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$ oraz zbiór ośmiu krawędzi $E = \{\{2, 3\}, \{3, 4\}, \{1, 4\}, \{4, 5\}, \{4, 8\}, \{7, 8\}, \{4, 6\}\}$. Wierzchołki to kolejne liczby naturalne, a krawędzie to, w tym przypadku, ich nieuporządkowane pary. Istnieje wiele rodzajów grafów. Część z nich pojawia się w tej pracy i została opisana w tym podrozdziale.

1.1.2. Graf skierowany i nieskierowany

Graf przedstawiony na rysunku 1.1 nazywany jest grafem nieskierowanym, ponieważ jego krawędzie to pary nieuporządkowane. Oznacza to, że nie mają one określonego kierunku. Przykładowo krawędź $\{2, 3\}$ jest równoważna krawędzi $\{3, 2\}$. Istnieją także grafy skierowane, których krawędzie są parami uporządkowanymi. Wówczas krawędź prowadzi od pierwszego elementu takiej pary do drugiego, ale nie odwrotnie. Krawędzie

skierowane są reprezentowane graficznie za pomocą strzałek. Na rysunku 1.2 przedstawiono przykład takiego grafu [13].



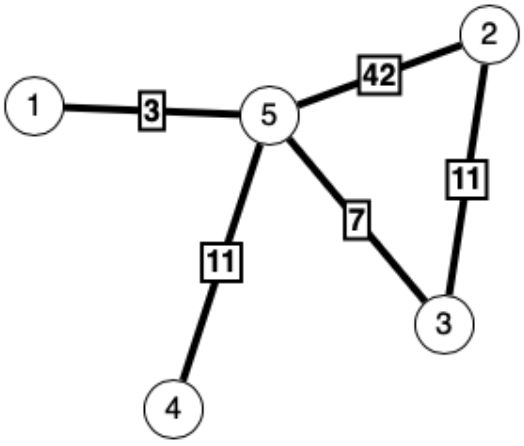
Rys. 1.2. Graf skierowany

1.1.3. Graf ważony

Krawędzie mogą posiadać także dodatkowy parametr, czyli wagę. Graf będzie wówczas nazywany grafem ważonym. Krawędzie nadal pozostają uporządkowanymi bądź nieuporządkowanymi parami wierzchołków, ale cała struktura musi zawierać dodatkowo funkcję wagową

$$w : E \to \mathbb{R}.$$

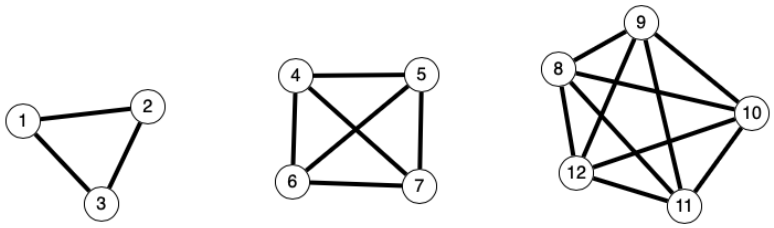
Funkcja wagowa każdemu elementowi ze zbioru krawędzi E przyporządkowuje pewną liczbę rzeczywistą. Na rysunku 1.3 przedstawiony został przykładowy graf ważony [8].



Rys. 1.3. Graf ważony

1.1.4. Graf pełny

Graf, w którym każdy wierzchołek jest połączony ze wszystkimi innymi wierzchołkami nazywany jest grafem pełnym. Na rysunku 1.4 przedstawione zostały trzy przykłady takich grafów [13].



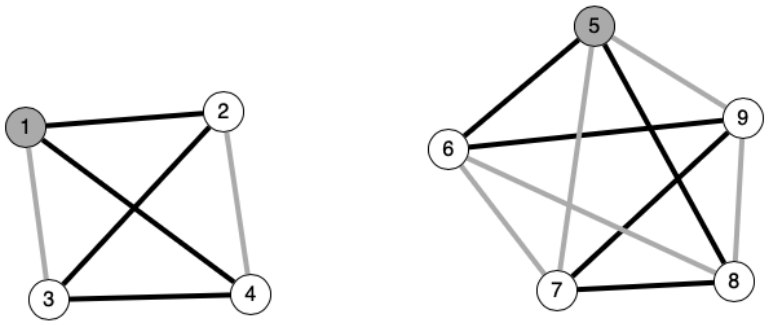
Rys. 1.4. Grafy pełne

1.1.5. Trasa, droga, ścieżka oraz cykl

Skończony ciąg krawędzi grafu nazywamy trasą. Przedstawić go można równoważnie jako ciąg wierzchołków. Jeżeli wszystkie krawędzie w trasie są różne (nie powtarzają się) to nazywana jest ona ścieżką. Jeżeli dodatkowo żaden wierzchołek nie występuje w niej więcej niż raz (dopuszczalne jest jedynie żeby trasa zaczynała się i kończyła tym samym wierzchołkiem), to nazywana jest wtedy drogą. Jeśli droga bądź ścieżka rozpoczyna się i kończy tym samym wierzchołkiem, mówi się, że jest ona zamknięta. Ścieżki zamknięte, które posiadają co najmniej jedną krawędź, nazywane są cyklami [13].

1.1.6. Cykl Hamiltona

Cykle, w których każdy wierzchołek grafu występuje dokładnie jeden raz, nazywane są cyklami Hamiltona. Na rysunku 1.5 przedstawione zostały ich dwa przykłady [14].



Rys. 1.5. Cykle Hamiltona

Cykle Hamiltona mogą być reprezentowane jako uporządkowany zbiór wierzchołków. Na przykład $(5, 8, 7, 9, 6)$.

1.2. ALGORYTM

1.2.1. Definicja algorytmu

Nieformalnie rzecz ujmując algorytm to metoda postępowania, która dla zadanego problemu ma doprowadzić do jego rozwiązania. Algorytmem jest na przykład przepis kucharski lub instrukcja obsługi. Reprezentowane są one często w postaci listy kroków. Istnieje wiele rodzajów algorytmów. Niektóre z nich, które pojawiają się w tej pracy, zostały krótko opisane w tym podrozdziale [10].

1.2.2. Algorytm heurystyczny

Wiele problemów posiada dużo możliwych rozwiązań, z których niektóre są lepsze, a niektóre gorsze. Algorytmy heurystyczne, nazywane także heurystykami, to algorytmy, które nie zapewniają znalezienia optymalnego (najlepszego możliwego) rozwiązania. Stosuje się je najczęściej dla problemów, których dokładne rozwiązanie zajęło by zbyt długo [9].

1.2.3. Algorytm lokalnej optymalizacji

Algorytmy lokalnej optymalizacji to algorytmy, które przeglądają zbiór dopuszczalnych rozwiązań, w poszukiwaniu najlepszego z nich, według pewnego iteracyjnego schematu. Powtarzają one w kółko pewien zestaw kroków aż do momentu spełnienia ustalonego z góry warunku stopu [11].

1.2.4. Algorytm zachłanny

Algorytmy zachłanne to algorytmy, które w każdym kroku podejmują decyzję o dalszym działaniu na podstawie oceny aktualnej sytuacji, bez uwzględnienia jej dalszych konsekwencji. Wybierają one zawsze tą opcję, która w danej chwili wydaje się być najkorzystniejsza, mimo że w kolejnych iteracjach może okazać się ona złym wyborem. Algorytmy zachłanne to podzbiór algorytmów heurystycznych [12].

1.2.5. Algorytm metaheurystyczny

Algorytmy metaheurystyczne, nazywane także metaheurystykami, to algorytmy, a w zasadzie metody ich konstruowania, które można stosować do rozwiązywania wielu różnych problemów definiowalnych za pomocą z góry określonych pojęć. Są one często inspirowane zjawiskami z fizyki oraz biologi [9].

1.3. POZOSTAŁE DEFINICJE

1.3.1. PRD

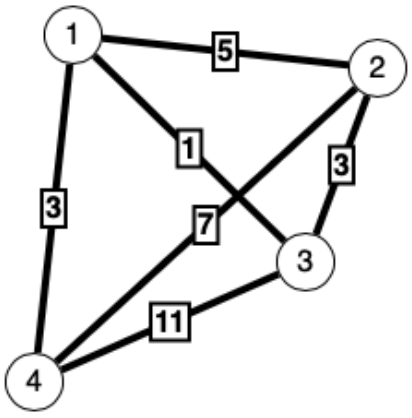
PRD (z angielskiego Percentage Relative Difference) to względna procentowa różnica między pewnym wynikiem x, a wynikiem optymalnym o obliczana według wzoru 1.1:

$$PRD(x) = \left(\frac{o - x}{o}\right) * 100\%, \tag{1.1}$$

gdzie o oznacza ocenę rozwiązania optymalnego, a x ocenę rozwiązania, dla którego obliczane jest PRD.

2. PROBLEM KOMIWOJAŻERA

Problem komiwojażera polega na znalezieniu cyklu Hamiltona o najmniejszej sumie wag krawędzi w grafie pełnym ważonym. Formalnie problem można zdefiniować w następujący sposób. Dla grafu $G = (V, E)$ oraz funkcji odległości $d : V \times V \to \mathbb{R}$, należy znaleźć cykl C o minimalnym koszcie, który zawiera wszystkie wierzchołki ze zbioru V. Koszt cyklu definiuje się jako $\sum_{e \in C} d(e)$, gdzie e to kolejne krawędzie należące do C [5]. W praktyce najczęściej wierzchołki grafu reprezentują miasta, a wagi krawędzi odległości między nimi. Celem komiwojażera jest odwiedzenie każdego z miast dokładnie jeden raz, a następnie powrót do miasta początkowego, w taki sposób aby przebyta przez niego droga była możliwie najkrótsza. Odległości między miastami, czy też wagi krawędzi grafu, przedstawić można za pomocą macierzy odległości, gdzie $d(i, j)$ oznacza odległość między miastami i oraz j. Rysunki 2.1 i 2.2 przedstawiają przykład grafu oraz odpowiadającej mu macierzy odległości.



Rys. 2.1. Graf dla problemu komiwojażera

0	5	1	3
5	0	3	7
1	3	0	11
3	7	11	0

Rys. 2.2. Macierz odległości

Graf przedstawiony w przykładzie jest grafem nieskierowanym. Dla każdej pary wierzchołków v, u krawędzie (v, u) oraz (u, v) (odpowiadające krawędzi $\{v, u\}$) są równoważne, więc ich waga jest taka sama. W takiej sytuacji macierz odległości jest macierzą symetryczną. Jej wyrazy są położone symetrycznie względem przekątnej. Innymi słowy $d(i, j) = d(j, i)$ dla wszystkich i oraz j. W takim przypadku instancja nazywana jest instancją symetryczną. Instancje problemu komiwojażera mogą być także reprezentowane

przez grafy skierowane. Jeżeli dla chociaż jednej pary wierzchołków $d(i,j) \neq d(j,i)$ to instancja nazywana jest instancją asymetryczną. Wyróżniamy także podzbiór instancji symetrycznych, nazywany instancjami euklidesowymi. Muszą one dodatkowo być możliwe do przedstawienia na płaszczyźnie euklidesowej [14].

3. WYBRANE ALGORYTMY

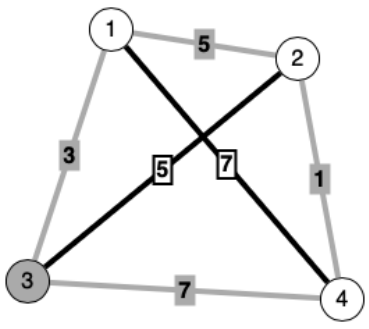
3.1. WSTĘP

Ze względu na to, że dokładne rozwiązanie problemu komiwojażera jest bardzo czasochłonne do jego rozwiązania stosowane są najczęściej heurystyki. W tym rozdziale opisane zostały wybrane przeze mnie algorytmy, dla których w dalszej części pracy przeprowadzone zostaną badania. Część z nich to metaheurystyki, które można stosować także dla innych problemów optymalizacyjnych, ale na potrzeby pracy niektóre z nich opisane zostały konkretnie dla problemu komiwojażera. Dodatkowo dla każdego z algorytmów zamieszczony został jego pseudokod. Pojawia się w nich funkcja $evaluate$, która zwraca łączny dystans danego rozwiązania.

3.2. ALGORYTM NAJBLIŻSZEGO SĄSIADA

3.2.1. Opis

Algorytm najbliższego sąsiada to algorytm zachłanny przeznaczony do rozwiązywania problemu komiwojażera. Na samym początku wybrane musi zostać, na przykład losowo, miasto początkowe. Następnie jako kolejne wybrane zostaje to, które znajduje się najbliżej (waga prowadzącej do niego krawędzi jest najmniejsza). Proces ten powtarzany jest, do momentu otrzymania pełnego rozwiązania, z uwzględnieniem faktu, że każde miasto musi być odwiedzone dokładnie jeden raz [7].



Rys. 3.1. Algorytm najbliższego sąsiada

Na rysunku 3.1 przedstawiony został przykład gdzie, jako wierzchołek startowy wybrany został wierzchołek numer 3. Jako drugi wybrany został wierzchołek numer 1, ponieważ ze wszystkich trzech krawędzi, które wychodzą z wierzchołka startowego, $\{3, 1\}$ ma najmniejszą wagę. Finalnie otrzymaliśmy trasę $(3, 1, 2, 4)$. Istnieje także rozszerzona wersja tego algorytmu nazywana powtarzalnym algorytmem najbliższego sąsiada. Polega ona na wykonaniu algorytmu wielokrotnie, wybierając każde miasto po kolei jako początkowe. Na końcu zwracane jest najlepsze z otrzymanych rozwiązań [7].

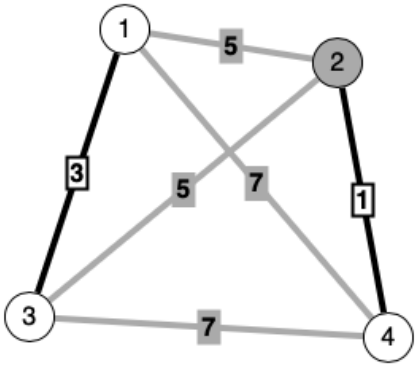
3.2.2. Pseudokod

1: **Input:** $start, size$
2: $solution \leftarrow [start]$
3: **while** length($solution$) < $size$ **do**
4: $\quad nearest \leftarrow$ nearest vertex, which is not in $solution$
5: $\quad solution \leftarrow solution$ + [$nearest$]
6: **end while**
7: **return** $solution$

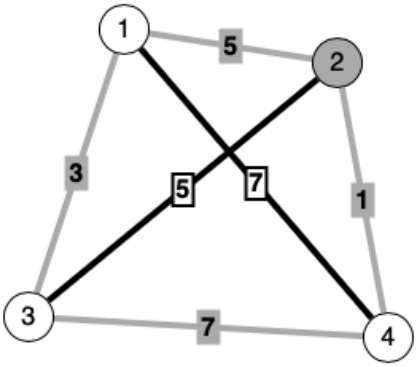
3.3. ALGORYTM 2-OPT

3.3.1. Opis

Algorytm 2-OPT to algorytm lokalnej optymalizacji stosowany do rozwiązywania problemu komiwojażera. Algorytm ten, w odróżnieniu na przykład od algorytmu najbliższego sąsiada, nie konstruuje rozwiązania od zera, tylko ulepsza już istniejące. Potrzebuje on zatem rozwiązania początkowego, które następnie będzie iteracyjnie poprawiał. Wygenerować je można na przykład losowo lub korzystając z innych algorytmów. Ulepszanie rozwiązania przebiega w następujący sposób. Przeglądane jest całe sąsiedztwo rozwiązania, które ma zostać ulepszone. W kontekście tego algorytmu są to rozwiązania mu bliskie, które mogą być otrzymane poprzez usunięcie dwóch krawędzi i zastąpienie ich dwoma innymi, w taki sposób aby otrzymane rozwiązanie dalej spełniało warunki trasy komiwojażera. W praktyce efekt ten może być osiągnięty na przykład poprzez zamianę miejscami dwóch wierzchołków w rozwiązaniu.
Na rysunku 3.3 zamieszczone zostało przykładowo wygenerowane sąsiednie rozwiązanie dla trasy $(2, 1, 4, 3)$, przedstawionej na rysunku 3.2. Zostało ono otrzymane poprzez usunięcie krawędzi $(1, 4)$ oraz $(3, 2)$ i zastąpienie ich krawędziami $(1, 3)$ oraz $(4, 2)$. W ten sposób powstało rozwiązanie $(2, 1, 3, 4)$. Łączny dystans poprawił się z 24 na 16. Równoważną operacją jest zamiana w rozwiązaniu miejscami dwóch ostatnich wierzchołków. Następnie ze wszystkich sąsiadów wybierany jest ten, który ma najmniejszą całkowitą długość trasy. Proces ten może być powtarzany aż do momentu znalezienia minimum lokalnego,



Rys. 3.2. Ulepszane rozwiązanie



Rys. 3.3. Rozwiązanie sąsiednie

czyli rozwiązania, w którego sąsiedztwie znajdują się same rozwiązania gorsze od niego. Algorytm może też zakończyć swoje działanie wcześniej jeżeli zostanie mu narzucona dodatkowo maksymalna liczba iteracji lub maksymalny czas działania [4].

3.3.2. Pseudokod

```
1:  Input: operator
2:  solution ← random solution
3:  hasImproved ← false
4:  while hasImproved == true do
5:      hasImproved ← false
6:      bestNeighbor ← best neighbor of solution received by operator
7:      if evaluate(bestNeighbor) < evaluate(solution) then
8:          solution = bestNeighbor
9:          hasImproved ← true
10:     end if
11: end while
12: return solution
```

3.4. ALGORYTM PRZESZUKIWANIA TABU

3.4.1. Opis

Jest to algorytm metaheurystyczny, który może być stosowany między innymi do rozwiązywania problemu komiwojażera. Polega na iteracyjnym ulepszaniu pewnego rozwiązania początkowego. Przeszukuje on jego sąsiedztwo w poszukiwaniu najlepszego rozwiązania, które się w nim znajduje. Sposób jego generowania nie jest określony, ale przykładowo

może to być zamiana kolejnością dwóch wierzchołków w cyklu. Proces ten jest powtarzany, przyjmując za każdy razem najlepsze rozwiązanie w sąsiedztwie jako nową trasę do ulepszania, do momentu spełnienia warunku stopu, którym może być na przykład maksymalna liczba iteracji lub maksymalny czas. Czasami stosowany jest też tak zwany mechanizm wykrywania stagnacji, czyli narzucenie maksymalnej liczby iteracji bez poprawy rozwiązania. Dodatkowo algorytm wykorzystuje tak zwaną listę tabu. Pod koniec każdej iteracji dodawany jest do niej ruch, który wykonaliśmy aby otrzymać wybrane rozwiązanie. Ruch jest rozumiany jako konkretne parametry wykonanej operacji, przykładowo indeksy wierzchołków, które zamieniliśmy ze sobą w rozwiązaniu. Ruchy, które znajdują się na tej liście nie są rozpatrywane podczas przeszukiwania sąsiedztwa. Nie trafiają tam one jednak na zawsze. Lista ma określoną maksymalną długość i kiedy ją osiągnie najdawniej dodane ruchy są z niej usuwane aby zrobić miejsce dla nowych. Lista ma służyć temu, aby zapobiegać wpadaniu w cykle, czyli powtarzaniu w kółko tej samej serii ruchów. Ze względu na to, że w kolejnych iteracjach wynik może się pogarszać, ponieważ znalezienie minimum lokalnego nie jest warunkiem stopu, algorytm tabu korzysta z pamięci długoterminowej, której podstawowym zastosowaniem jest przechowywanie najlepszego rozwiązania znalezionego w czasie działania algorytmu. Często implementowane jest także kryterium aspiracji, czyli mechanizm który w określonych przypadkach zezwala na wykonanie ruchu z listy tabu. Przykładowo wtedy, kiedy rozwiązanie, które byśmy otrzymali jest lepsze od najlepszego dotychczas znalezionego w czasie działania algorytmu. Warto wspomnieć, że algorytm przeszukiwania tabu jest w zasadzie bardziej ideą niż algorytmem, więc część wyznaczonych kroków może być modyfikowana na potrzeby konkretnego problemu bądź konkretnej instancji [6].

3.4.2. Pseudokod

1: **Input:** $maxLength$, $operator$
2: $solution \leftarrow$ random solution
3: $bestSolutionEver \leftarrow solution$
4: $tabu \leftarrow []$
5: **while** $stopCondition$ == false **do**
6: $bestNeighbor \leftarrow$ best neighbor $\notin tabu$ of $solution$ received by $operator$
7: $bestMove \leftarrow$ move, which leads to $bestNeighbor$
8: $solution = bestNeighbor$
9: **if** evaluate($solution$) < evaluate($bestSolutionEver$) **then**
10: $bestSolutionEver = solution$
11: **end if**
12: $tabu \leftarrow tabu$ + [$bestMove$]
13: **if** length($tabu$) > $maxLength$ **then**
14: $tabu \leftarrow tabu$ without first element
15: **end if**
16: **end while**
17: **return** $solution$

3.5. ALGORYTM KOLONII MRÓWEK

3.5.1. Opis

Jest to metaheurystyka, którą można stosować między innymi do rozwiązywania problemu komiwojażera. Konstruuje on rozwiązanie od zera, więc nie wymaga rozwiązania początkowego. Algorytm jest zainspirowany rzeczywistym zachowaniem mrówek, które poszukując pożywienia zostawiają po sobie ślad feromonowy, za którym później podążają pozostałe mrówki. Na krótszych trasach (bardziej optymalnych) mniej feromonu zdąży odparować przez co są one częściej wybierane przez całą kolonię. Algorytm ma do dyspozycji określoną liczbę mrówek. W każdej iteracji dla każdej z nich wybierane jest losowe miasto początkowe. Następnie przemierzają one kolejne miasta, do momentu uzyskania pełnego rozwiązania, w taki sposób aby nie odwiedzić tego samego miasta dwa razy. Kolejne wierzchołki są wybierane na podstawie prawdopodobieństw, które są wyliczane według wzoru 3.1:

$$f^{\alpha} \cdot \left(\frac{1}{w}\right)^{\beta}, \tag{3.1}$$

gdzie f oznacza ilość feromonu na krawędzi do niego prowadzącej, a w jej wagę. Początkowa ilość feromonu na krawędziach może być równomiernie rozłożona, losowana

dla każdej krawędzi lub dobrana w inny sposób. Pod koniec iteracji, dla każdej mrówki zwiększana jest ilość feromonu na krawędziach, po których przechodziła, o odwrotność łącznej długości przebytej przez nią trasy. Następnie część feromonu jest odparowywana dla każdej krawędzi w grafie. O tym jak dużo feromonu ma odparować po każdej iteracji decyduje z góry ustalany współczynnik odparowywania, który jest liczbą rzeczywistą z przedziału od zera do jeden. Nowy poziom feromonu jest wyliczany według wzoru 3.2:

$$f' = f \cdot (1 - o), \tag{3.2}$$

gdzie f to aktualny poziom feromonu, a o to wartość współczynnika odparowywania. Algorytm jest powtarzany do momentu spełnienia określonego warunku stopu, którym może być na przykład przekroczenie maksymalnej liczby iteracji. Na końcu zwracane jest najlepsze rozwiązanie otrzymane w czasie działania całego algorytmu [9].

3.5.2. Pseudokod

Input: $maxIterations$, $numAnts$, $evaporationRate$, $initialPheromoneLevels$
$i \leftarrow 0$
$bestSolutionEver \leftarrow$ []
$bestDistanceEver \leftarrow \infty$
$pheromoneLevels \leftarrow initialPheromoneLevels$
while $i <$ $maxIterations$ **do**
    $solutions \leftarrow$ solutions generated by $numAnts$ ants
    $pheromoneLevels \leftarrow$ new pheromone levels updated by ant routes
    $pheromoneLevels \leftarrow pheromoneLevels \cdot (1 - evaporationRate)$
    $bestSolution \leftarrow$ best solution in this iteration
    if evaluate($bestSolution$) < $bestDistanceEver$ **then**
        $bestSolutionEver \leftarrow bestSolution$
        $bestDistanceEver \leftarrow$ evaluate($bestSolution$)
    end if
    $i \leftarrow i + 1$
end while
return $bestSolutionEver$

3.6. ALGORYTM SYMULOWANEGO WYŻARZANIA

3.6.1. Opis

Jest to metaheurystyka, za pomocą której możemy rozwiązywać między innymi problem komiwojażera. Inspirowana jest ona rzeczywistym zjawiskiem obserwowanym w metalurgii,

a mianowicie wzrostem plastyczności metalu wraz ze wzrostem jego temperatury. Algorytm polega na iteracyjnym ulepszaniu pewnego rozwiązania początkowego. W każdej iteracji rozwiązanie jest modyfikowane według ustalonego ruchu (na przykład zamianie miejscami dwóch wierzchołków w rozwiązaniu). Parametry ruchu (na przykład indeksy zamienianych wierzchołków) są wybierane losowo. Po dokonaniu modyfikacji otrzymane rozwiązanie może być przyjęte jako nowe lub odrzucone. Jeżeli jest lepsze od modyfikowanego to zawsze zostanie zaakceptowane. W przeciwnym wypadku prawdopodobieństwo jego przyjęcia P jest obliczane według wzoru 3.3:

$$P = e^{\left(\frac{\Delta d}{T}\right)}, \tag{3.3}$$

gdzie Δd to różnica całkowitych dystansów rozwiązań, e to liczba Eulera, a T to temperatura. Temperatura początkowo przyjmuje określoną wartość, a następnie jest zmniejszana co iterację. To jak szybko maleje zależy od współczynnika schładzania. Jest on określoną liczbą rzeczywistą z przedziału od zera do jednego, a nową temperaturę T' oblicza się według wzoru 3.4:

$$T' = T \cdot c, \tag{3.4}$$

,gdzie T to aktualna temperatura, a c to wartość współczynnika schładzania. Jeśli współczynnik schładzania jest niski temperatura maleje szybciej. Im temperatura jest niższa tym mniejsze jest prawdopodobieństwo przyjęcia gorszego rozwiązania. Rośnie ono jednak wraz ze spadkiem różnicy dystansów. W obrębie jednej iteracji (przed każdym obniżeniem temperatury) rozwiązanie może być modyfikowane wielokrotnie, w zależności od przyjętych ustaleń. Warunkiem stopu jest najczęściej przekroczenie maksymalnej liczby iteracji lub maksymalnego czasu [3].

3.6.2. Pseudokod

```
1: Input: maxIterations, coolingRate, operator
2: i ← 0
3: solution ← random solution
4: bestSolutionEver ← solution
5: while i < maxIterations do
6:     neighbor ← random neighbor of solution received by operator
7:     difference ← evaluate(solution) − evaluate(neighbor)
8:     randomFloat ← random float between 0 and 1
9:     if difference < 0 or randomFloat < exp(−difference/temperature) then
10:        solution ← neighbor
11:        if evaluate(solution) < evaluate(bestSolutionEver) then
12:            bestSolutionEver ← solution
13:        end if
14:    end if
15:    i ← i + 1
16:    temperature ← temperature · coolingRate
17: end while
18: return bestSolutionEver
```

4. IMPLEMENTACJA

4.1. WYKORZYSTANE NARZĘDZIA

4.1.1. Julia

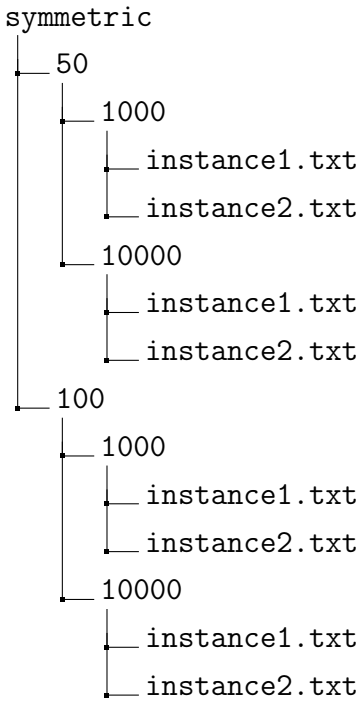
Julia to język programowania, który został stworzony z myślą o szybkości jego działania. Jego główne zastosowania to analiza numeryczna oraz obliczenia naukowe, ale dobrze sprawdza się także w innych dziedzinach. Algorytmy badane w pracy zostały zaimplementowane właśnie w tym języku, ze względu na jego wydajność, która pozwoliła na zbadanie odpowiedniej liczby instancji w sensownym czasie [1].

4.1.2. Python

Jest to język programowania, który charakteryzuje się prostą i czytelną składnią kodu źródłowego. Python jest jednym z najpopularniejszych języków na świecie stosowanym w wielu dziedzinach. Posiada bardzo dużą liczbę bibliotek, zarówno standardowych jak i zewnętrznych, które ułatwiają tworzenie oprogramowania. Jego wadą jest wydajność, która utrudnia tworzenie aplikacji wymagających bardzo szybkiego działania. Program do generowania instancji, badań oraz wizualizacji został napisany właśnie w tym języku, ponieważ nie wymagał dużej wydajności, a prosta składnia oraz dostępne biblioteki pozwoliły na jego wygodną implementację [2].

4.2. PROGRAM DO BADAŃ

Pierwsza funkcja programu to wygenerowanie zestawu instancji o zadanych parametrach jako drzewa folderów z plikami tekstowymi w odpowiednim formacie. Można określić ich typy, wielkości i zakresy wartości oraz ilość generowanych instancji dla każdej z takich trójek. Do generowania liczb losowych wykorzystany został moduł „random”, a do generowania folderów i plików tekstowych moduł „os”. Na rysunku 4.1 przedstawiona została struktura plików przykładowego zestawu instancji.



Rys. 4.1. Struktura plików zestawu instancji

Coraz głębsze foldery oznaczają kolejno typ, wielkość oraz zakres wartości instancji. Przez wielkość rozumiemy ilość wierzchołków, a przez zakres wartości maksymalną wagę pojedynczej krawędzi. Na listingu 4.1 jest pokazany przykładowy plik tekstowy reprezentujący pojedynczą instancję.

```
1   symmetric                   #typ
2   8                           #rozmiar
3   100                         #zakres wartosci
4   0 70 1 87 16 42 51 2        #macierz odleglosci
5   70 0 86 4 5 73 29 33
6   1 86 0 96 28 90 86 35
7   87 4 96 0 67 99 1 11
8   16 5 28 67 0 19 71 54
9   42 73 90 99 19 0 1 38
10  51 29 86 1 71 1 0 40
11  2 33 35 11 54 38 40 0
```

Listing 4.1: Przykład pliku z pojedynczą instancji

W pliku z pojedynczą instancją trzy pierwsze wiersze oznaczają także typ, wielkość i zakres wartości. Następnie w kolejnych wierszach zawarta jest macierz odległości. Kolejna funkcjonalność to porównywanie różnych konfiguracji parametrów dla wybranego algo-

rytmu na stworzonym wcześniej zestawie instancji zbierając takie informacje jak średnie wyniki, czas czy zużyta pamięć. Są one również zapisywane do plików tekstowych. Skrypty w języku Julia są uruchamiane za pomocą modułu „subprocess", a przeglądanie plików obsługuje moduł „glob". Na listingu 4.2 zamieszczony został przykład pliku wynikowego dla pojedynczej instancji.

```
1  symmetric                    #typ
2  60                           #rozmiar
3  1000                         #zakres wartosci
4  20                           #liczba powtorzen
5  1 5995.0 16.892197650000004 30184.03125 5331.0
6  2 2944.4 77.87081449999998 131200.78125 2574.0
```

Listing 4.2: Plik z wynikami

Pierwsze cztery wiersze oznaczają typ, wielkość, zakres wartości instancji oraz ilość powtórzeń algorytmu. Następnie każdy wiersz zawiera wyniki, a oddzielone spacjami wartości reprezentują kolejno konfigurację parametrów, średni wynik, średni czas, średnie zużycie pamięci oraz najlepszy wynik. Poza tym dla każdego folderu generowany jest plik z uśrednionymi wynikami dla wszystkich instancji, które się w nim znajdują. Ma on bardzo podobny format.

```
7  symmetric                      #typ
8  60                             #rozmiar
9  1000                           #zakres wartosci
10 1 6342.7350001 16.59735 29645.7375 135.4815 4766.0
11 2 3110.1925 73.017688 129070.36875 15.11750 2277.0
```

Listing 4.3: Plik ze średnimi wynikami

Różni się tylko tym, że nie mamy tu już informacji o ilości powtórzeń oraz dodana jest dodatkowa statystyka (jako przedostatnia), a mianowicie średnie PRD. Jako, że nie są znane optymalne rozwiązania dla losowo generowanych instancji jako rozwiązanie optymalne przyjęte zostało najlepsze rozwiązanie jakie udało się otrzymać dla danej instancji. Dodatkowo, po przetestowaniu wszystkich instancji w danym zestawie, można wygenerować plik tekstowy zawierający najlepsze konfiguracje parametrów dla różnych rodzajów instancji.

```
12      "('symmetric', '60', '1000')": "2",
13      "('symmetric', '20', '1000')": "2",
14      "('symmetric', '80', '1000')": "2",
15      "('symmetric', '100', '1000')": "2",
16      "('symmetric', '40', '1000')": "2",
17      "('asymmetric', '60', '1000')": "1",
18      "('asymmetric', '20', '1000')": "1",
19      "('asymmetric', '80', '1000')": "1",
20      "('asymmetric', '100', '1000')": "1",
21      "('asymmetric', '40', '1000')": "1"
```

Listing 4.4: Plik z dobranymi parametrami

W tym przypadku testowany algorytm posiada tylko jeden parametr, więc dla każdego rodzaju instancji (typ, wielkość, zakres wartości) została przypisana pojedyncza liczba całkowita reprezentująca jego wariant. Jeżeli wygenerowane zostaną takie pliki dla każdego z algorytmów, z ich wykorzystaniem można dla wybranego zestawu instancji porównać algorytmy między sobą, otrzymując podobne pliki wynikowe jak dla porównywania parametrów. Różnić się one będą tylko tym, że zamiast konfiguracji parametrów, pierwsza wartość w wierszach z wynikami reprezentuje algorytm. Ostatnia funkcjonalność to generowanie arkuszów kalkulacyjnych z tabelami zawierającymi średnie wyniki dla całego zestawu instancji. Umożliwiają one ich wygodny przegląd, wizualizację i interpretację. Skorzystałem tutaj z modułu „pandas”. W osobnych zakładkach arkusza przedstawione są tabele dla różnych statystyk (średni wynik, średni czas, średnie zużycie pamięci, średnie PRD oraz najlepszy wynik). Następnie z wykorzystaniem modułu "matplotlib" można wygenerować na jego podstawie wykresy.

5. ANALIZA PARAMETRÓW ALGORYTMÓW

5.1. WSTĘP

W tym rozdziale zawarte jest porównanie działania algorytmów dla różnych parametrów. Na podstawie badań, przeprowadzonych z wykorzystaniem napisanego programu, dla każdego z nich dobrane zostały na podstawie średniego PRD najlepsze otrzymane parametry dla określonych konfiguracji typu, rozmiaru i zakresu wartości instancji. Algorytmy zostały zbadane na osobnych zestawach instancji, ale każdy z nich był taki sam pod kątem ich rodzajów oraz ilości. Zbadane zostały wszystkie trzy typy (asymetryczny, symetryczny, euklidesowy) w rozmiarach $20, 40, 60, 80, 100$. Zakres wartości pojedynczej krawędzi dla każdej instancji wynosił 1000. Dla każdej z konfiguracji wygenerowane zostało po 20 instancji, co daje łącznie 300 instancji na zestaw. Dodatkowo dla każdej instancji zebrane zostały średnie wyniki z 20 prób. Na niektórych wykresach PRD widać niezgodność ze średnimi wynikami. Jest to spowodowane tym, że nie są znane rzeczywiste rozwiązania optymalne, w związku z czym do obliczania PRD zostały przyjęte najlepsze otrzymane rozwiązania. Z tego względu, jeżeli dla któregoś typu instancji otrzymano bardzo dobre najlepsze wyniki to PRD średnich wyników będzie zawyżone. Analogicznie jeżeli otrzymane najlepsze wyniki były niskie, PRD średnich wyników będzie zaniżone. Jednostka czasu to milisekundy, jednostka zużycia pamięci to bajty, a jednostka PRD to procenty. Wartości współczynników (odparowywania feromonu dla algorytmu kolonii mrówek oraz schładzania dla algorytmu symulowanego wyżarzania) wyrażone są również w procentach jako liczba całkowita od jednego do stu (liczba 50 oznacza więc współczynnik o wartości 0.5).

5.2. ALGORYTM NAJBLIŻSZEGO SĄSIADA

Dla tego algorytmu nie zostały przeprowadzone wstępne badania, ponieważ jest on mało złożony i nie posiada żadnych parametrów. Implementacja zawiera rozszerzoną wersję tego algorytmu, czyli rozszerzony algorytm najbliższego sąsiada.

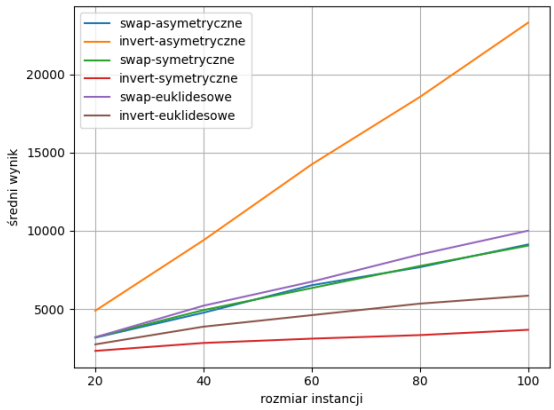
5.3. ALGORYTM 2-OPT

5.3.1. Wstęp

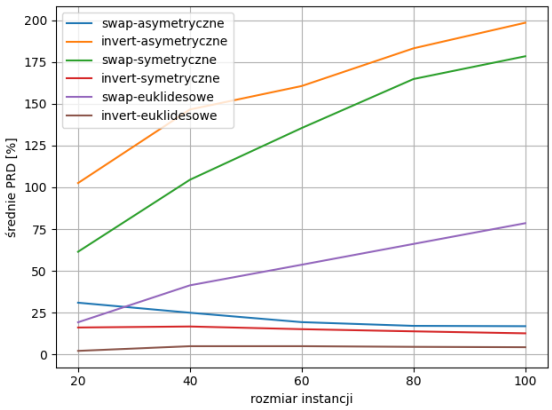
Dla tego algorytmu został zbadany tylko jeden parametr, a mianowicie operator modyfikacji rozwiązania. Porównane zostały jego dwa warianty – $swap$ oraz $invert$. Pierwszy z nich polega na zamianie miejscami dwóch wierzchołków w rozwiązaniu. Przykładowo stosując operację $swap(1, 3)$ na rozwiązaniu $(5, 2, 3, 1, 4)$ otrzymane zostanie rozwiązanie $(3, 2, 5, 1, 4)$, ponieważ zamienione zostaną wierzchołki znajdujące się na pierwszym i trzecim miejscu. Drugi natomiast polega na odwróceniu kolejności całego fragmentu rozwiązania pomiędzy dwoma wierzchołkami (włącznie). Przykładowo stosując operację $invert(2, 4)$ na rozwiązaniu $(5, 2, 3, 1, 4)$ otrzymane zostanie rozwiązanie $(5, 1, 3, 2, 4)$, ponieważ odwrócona zostanie kolejność wierzchołków znajdujących się na drugim, trzecim i czwartym miejscu. Oba te operatory są równoważne usunięciu dwóch krawędzi z cyklu i dodaniu dwóch innych na ich miejsce, czyli są zgodne z założeniami generowania sąsiedztwa przez algorytm 2-OPT. Rozwiązanie początkowe generowane było zawsze losowo. Jako warunek stopu przyjęte zostało znalezienie minimum lokalnego.

5.3.2. Operator modyfikacji

5.3.2.1. Średnie wyniki oraz PRD



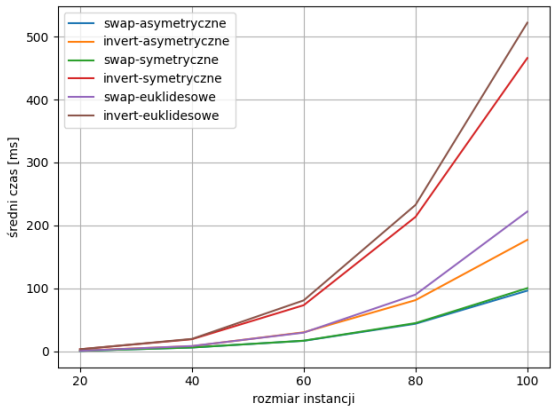
Rys. 5.1. Operator modyfikacji - wyniki



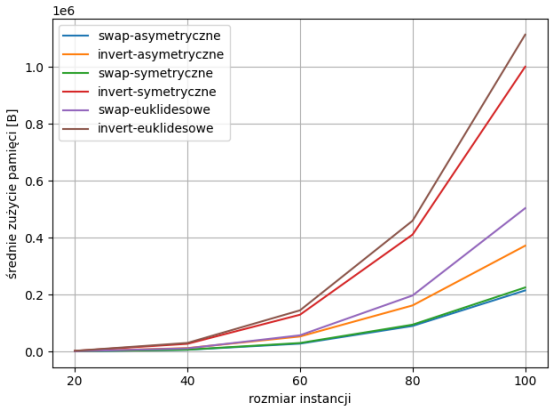
Rys. 5.2. Operator modyfikacji - PRD

Operator $invert$ zdecydowanie najgorzej działa dla instancji asymetrycznych, natomiast operator $swap$ jest niewrażliwy na symetryczność instancji. Da obu operatorów wyniki dla instancji euklidesowych są trochę gorsze niż dla symetrycznych (oraz asymetrycznych w przypadku operatora $swap$). Odwrotny wniosek jest widoczny na wykresie PRD, ale wynika to z tego, że nie są znane rzeczywiste rozwiązania optymalne. Podsumowując, operator $invert$ zwraca lepsze wyniki dla instancji symetrycznych oraz euklidesowych, a dla asymetrycznych $swap$.

5.3.2.2. Średni czas i zużycie pamięci



Rys. 5.3. Operator modyfikacji - czas



Rys. 5.4. Operator modyfikacji - pamięć

Algorytm z operatorem $invert$ zdecydowanie najkrócej działa dla instancji asymetrycznych. Dla obu operatorów algorytm działa najdłużej dla instancji euklidesowych. Dla każdego z trzech typów instancji algorytm krócej działa dla operatora $swap$. Wykres zużycia pamięci jest niemal identyczny jak wykres czasu, więc wnioski są dla niego dokładnie takie same.

5.3.3. Dobrane parametry

Typ	Rozmiar	Operator
symetryczny	20	$invert$
symetryczny	40	$invert$
symetryczny	60	$invert$
symetryczny	80	$invert$
symetryczny	100	$invert$
asymetryczny	20	$swap$
asymetryczny	40	$swap$
asymetryczny	60	$swap$
asymetryczny	80	$swap$
asymetryczny	100	$swap$
euklidesowy	20	$invert$
euklidesowy	40	$invert$
euklidesowy	60	$invert$
euklidesowy	80	$invert$
euklidesowy	100	$invert$

Tabela 5.1. 2OPT - dobrane parametry

W tabeli 5.1 przedstawione zostały parametry dobrane dla konkretnych typów i rozmiarów instancji. Pokrywają się one z wyciągniętymi wnioskami. Dla instancji asymetrycznych dobrany został operator $swap$, a dla pozostałych $invert$.

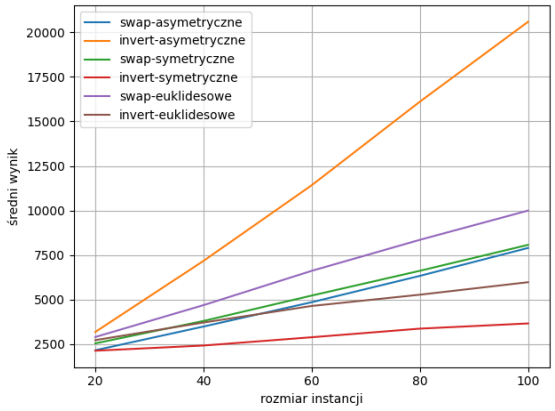
5.4. ALGORYTM PRZESZUKIWANIA TABU

5.4.1. Wstęp

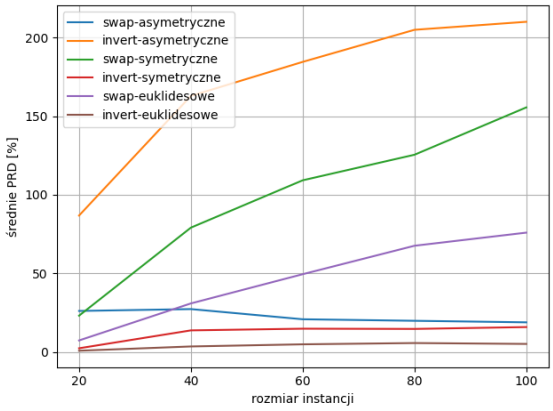
Dla tego algorytmu zbadane zostały dwa parametry. Porównane zostało działanie operatorów modyfikacji $swap$ i $invert$ oraz różne długości listy tabu. Zbadane zostały długości 5, 12 oraz 25. Zaimplementowanym warunkiem stopu została maksymalna liczba iteracji wynosząca 70. Jeżeli w czasie tych siedemdziesięciu iteracji nie zostało znalezione ani jedno minimum lokalne to algorytm kontynuował swoje działania do momentu jego znalezienia. Rozwiązania początkowe generowane były zawsze losowo. Zaimplementowane zostało także kryterium aspiracji, które dopuszczało wybór ruchu z listy tabu jeżeli dane rozwiązanie było lepsze od najlepszego dotychczas znalezionego.

5.4.2. Porównanie operatorów $swap$ oraz $invert$

5.4.2.1. Średnie wyniki oraz PRD



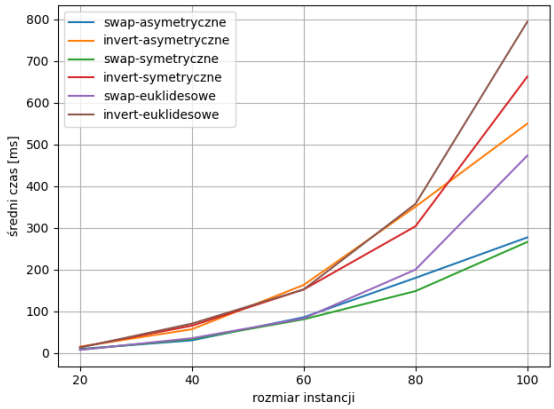
Rys. 5.5. Operator modyfikacji - wyniki



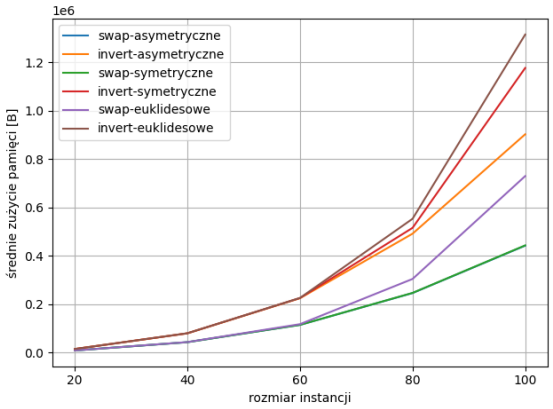
Rys. 5.6. Operator modyfikacji - PRD

Operator $invert$ zdecydowanie najgorzej działa dla instancji asymetrycznych, natomiast operator $swap$ jest niewrażliwy na symetryczność instancji. Dla obu operatorów wyniki dla instancji euklidesowych są trochę gorsze niż dla symetrycznych (oraz asymetrycznych w przypadku operatora $swap$). Odwrotny wniosek jest widoczny na wykresie PRD, ale wynika to z tego, że nie są znane rzeczywiste rozwiązania optymalne. Podsumowując, operator $invert$ zwraca lepsze wyniki dla instancji symetrycznych oraz euklidesowych, a dla asymetrycznych $swap$.

5.4.2.2. Średni czas i zużycie pamięci



Rys. 5.7. Operator modyfikacji - czas

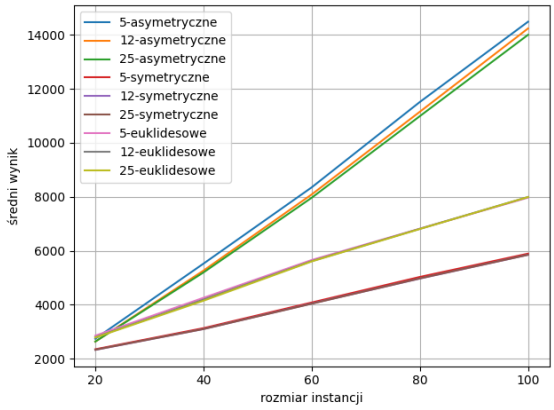


Rys. 5.8. Operator modyfikacji - pamięć

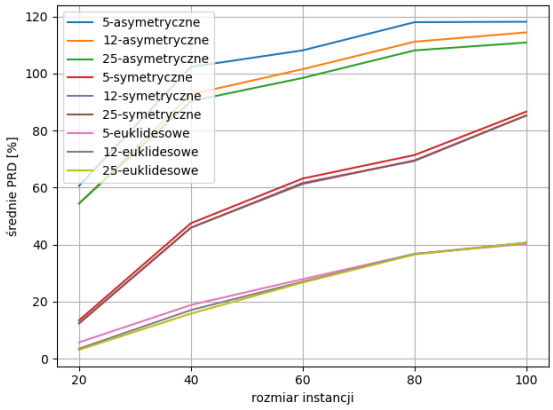
Algorytm z operatorem $swap$ działa najkrócej dla instancji symetrycznych, nieznacznie dłużej dla asymetrycznych, a najpóźniej kończy działanie dla instancji euklidesowych. Algorytm z operatorem $invert$ działa najdłużej dla instancji euklidesowych, a dla instancji symetrycznych i asymetrycznych działa trochę krócej. Różnice te są niezauważalne dla małych instancji, ale robią się mocno widoczne wraz ze wzrostem ich rozmiaru. Algorytm działa zdecydowanie krócej dla operatora $swap$ niezależnie od typu instancji. Jeżeli chodzi o zużycie pamięci wykresy są podobne, z wyjątkiem dwóch różnic. Dla operatora $invert$ więcej pamięci algorytm zużywa dla instancji symetrycznych niż asymetrycznych. Dla operatora $swap$ instancje symetryczne oraz asymetryczne zużywają tyle samo pamięci.

5.4.3. Porównanie maksymalnej długości listy tabu

5.4.3.1. Średnie wyniki oraz PRD



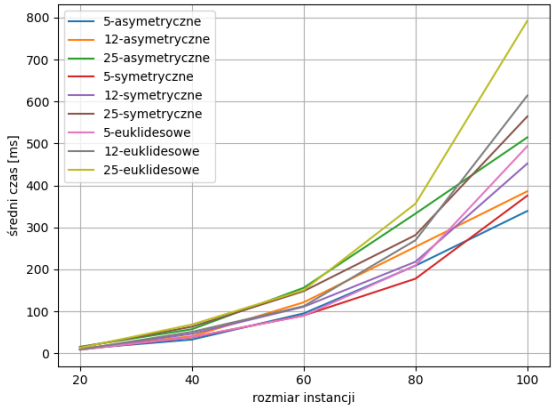
Rys. 5.9. Długość listy tabu - wyniki



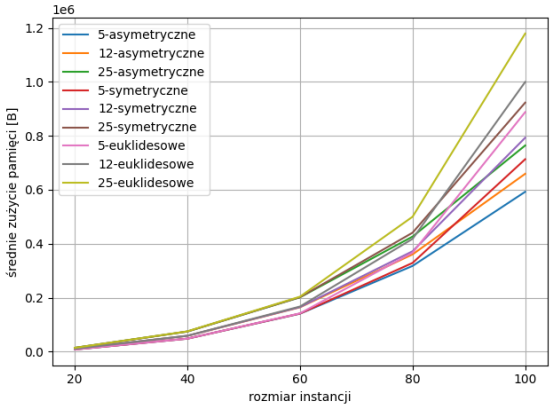
Rys. 5.10. Długość listy tabu - PRD

Dla instancji asymetrycznych jakość rozwiązań rośnie wraz ze wzrostem długości listy tabu. Dla pozostałych typów instancji algorytm zwraca najgorsze wyniki dla długości 5, ale dla pozostałych długości nie widać żadnych zależności.

5.4.3.2. Średni czas i zużycie pamięci



Rys. 5.11. Długość listy tabu - czas



Rys. 5.12. Długość listy tabu - pamięć

Zarówno średni czas jak i zużycie pamięci rosną wraz ze wzrostem maksymalnej długości listy tabu.

5.4.4. Dobrane parametry

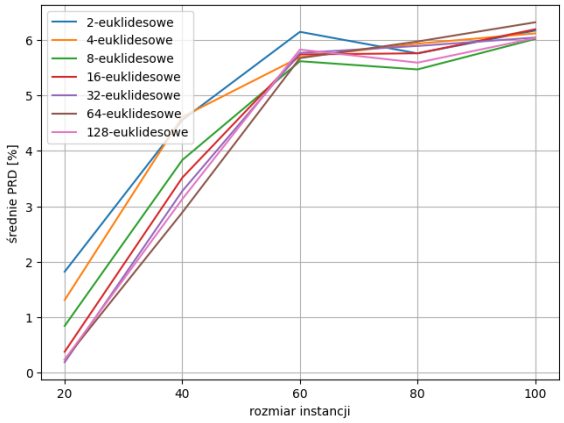
Typ	Rozmiar	Operator	Długość listy
symetryczny	20	*invert*	25
symetryczny	40	*invert*	25
symetryczny	60	*invert*	25
symetryczny	80	*invert*	12
symetryczny	100	*invert*	25
asymetryczny	20	*swap*	5
asymetryczny	40	*swap*	12
asymetryczny	60	*swap*	25
asymetryczny	80	*swap*	25
asymetryczny	100	*swap*	25
euklidesowy	20	*invert*	25
euklidesowy	40	*invert*	25
euklidesowy	60	*invert*	25
euklidesowy	80	*invert*	12
euklidesowy	100	*invert*	25

Tabela 5.2. Algorytm przeszukiwania tabu - dobrane parametry

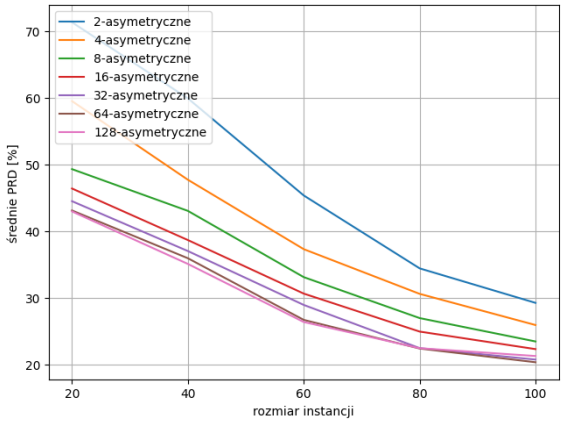
Większość parametrów została dobrana zgodnie z wyciągniętymi wnioskami. Dla instancji asymetrycznych wybrany został operator $swap$, dla symetrycznych i euklidesowych $invert$. Dla żadnej z instancji symetrycznych i euklidesowych nie została dobrana długość tabu 5. Dla większości instancji asymetrycznych została dobrana długość tabu 25.

5.4.5. Dodatkowe badania

Dodatkowo żeby dokładniej zbadać zależności wyników od maksymalnej długości listy tabu wygenerowany został dodatkowy zestaw instancji, dla którego zbadane zostały kolejne potęgi dwójki jako jej maksymalne długości. Liczba iteracji została zmniejszona do 60, a jako operator modyfikacji rozwiązania przyjęto $invert$.



Rys. 5.13. Instancje symetryczne - PRD



Rys. 5.14. Instancje euklidesowe - PRD

Dla instancji asymetrycznych widać ponownie, że większa długość listy tabu poprawia wyniki. Różnice maleją wraz ze wzrostem rozmiaru instancji oraz maksymalnej długości listy tabu. Jeśli chodzi o instancje euklidesowe, dla dużych rozmiarów instancji nie widać żadnej zależności wyników od maksymalnej długości listy, ale dla mniejszych instancji można wyciągnąć podobne wnioski jak dla instancji asymetrycznych. Szczególnie dobrze widać to dla najmniejszych długości - 2 oraz 4, dla których zwracane są zdecydowanie gorsze wyniki. Dla wykresu instancji symetrycznych wnioski są bardzo podobne jak dla euklidesowych, ale nie został on zamieszczony w pracy.

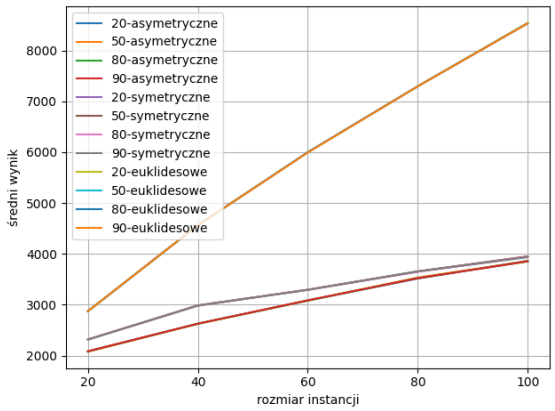
5.5. ALGORYTM KOLONII MRÓWEK

Dla algorytmu kolonii mrówek zbadane zostały trzy parametry. Pierwszy to współczynnik odparowywania feromonu. Porównane zostały dla niego wartości $20, 50, 80, 90$ procent. Drugi to sposób generowania początkowego rozkładu feromonu. Porównany został rozkład losowy (liczba zmiennoprzecinkowa z przedziału od zera do jeden dla każdej krawędzi) oraz równomierny (równy jeden dla każdej krawędzi). Ostatni to współczynniki α oraz β.

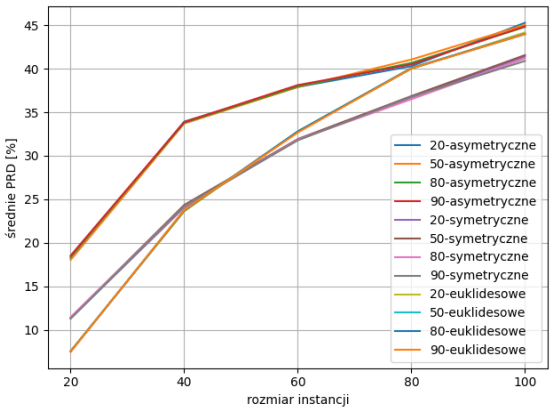
Współczynnik α był równy jeden, a dla współczynnika β zbadane zostały wartości $2, 5$ oraz 10. Liczba iteracji była równa 10, a liczba mrówek 50.

5.5.1. Współczynnik odparowywania feromonu

5.5.1.1. Średnie wyniki oraz PRD



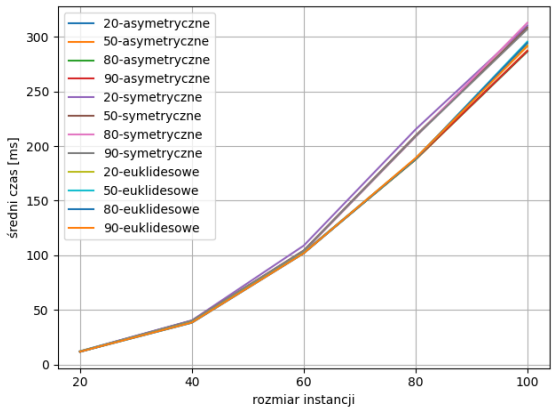
Rys. 5.15. Wsp. odparowywania - wyniki



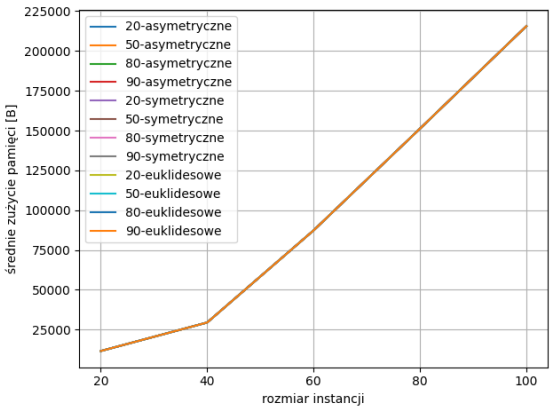
Rys. 5.16. Wsp. odparowywania - PRD

Nie widać żadnej zależności pomiędzy współczynnikiem odparowywania feromonu, a wynikami. Wygenerowany został także dodatkowy zestaw instancji, gdzie zbadano współczynniki odparowywania o wartościach $0, 10, 20.., 100$, ale również nie pokazało to żadnej zależności. Najlepsze wyniki algorytm zwraca dla instancji asymetrycznych, trochę gorsze dla symetrycznych, a zdecydowanie najgorsze dla euklidesowych. Trochę inne wnioski można wyciągnąć patrząc na wykres PRD, ale wynika to z braku znajomości rozwiązań optymalnych.

5.5.1.2. Średni czas i zużycie pamięci



Rys. 5.17. Wsp. odparowywania - czas

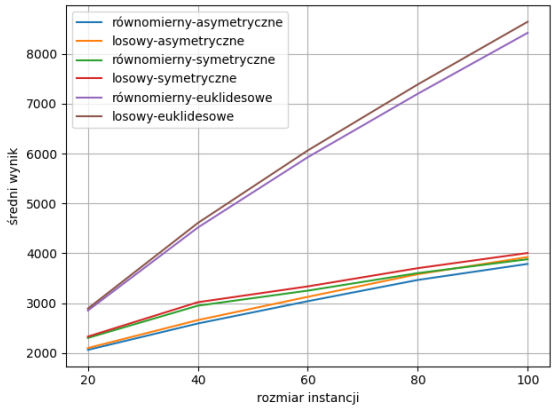


Rys. 5.18. Wsp. odparowywania - pamięć

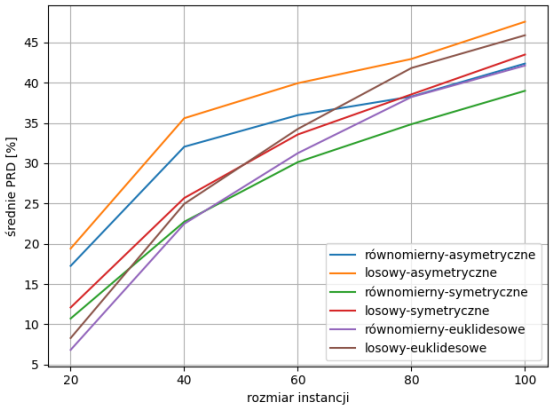
Algorytm zużywa tyle samo pamięci niezależnie od współczynnika odparowywania czy typu instancji. Jeśli chodzi o czas najdłużej algorytm działa dla instancji symetrycznych.

5.5.2. Początkowy rozkład feromonu

5.5.2.1. Średnie wyniki oraz PRD



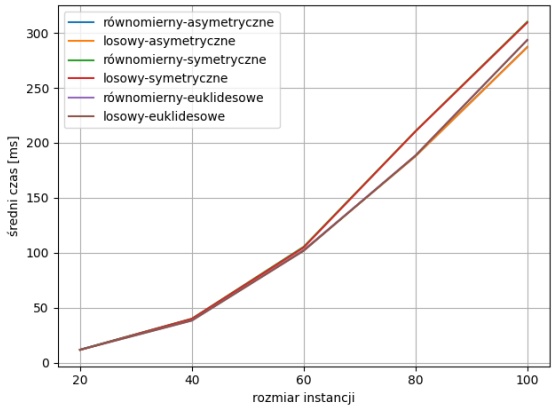
Rys. 5.19. Rozkład feromonu - wyniki



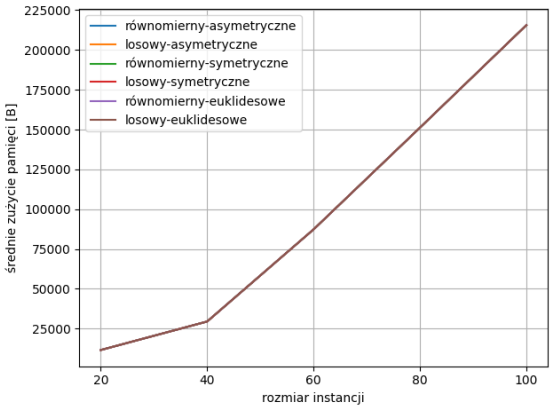
Rys. 5.20. Rozkład feromonu - PRD

Algorytm działa lepiej dla równomiernego początkowego rozkładu feromonu niż dla losowego. Różnice te rosną wraz ze wzrostem wielkości instancji.

5.5.2.2. Średni czas i zużycie pamięci



Rys. 5.21. Rozkład feromonu - czas

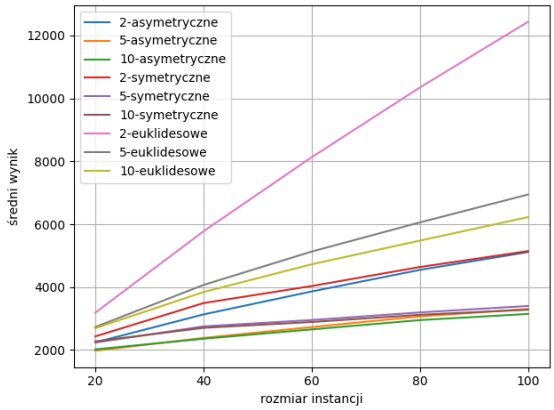


Rys. 5.22. Rozkład feromonu - pamięć

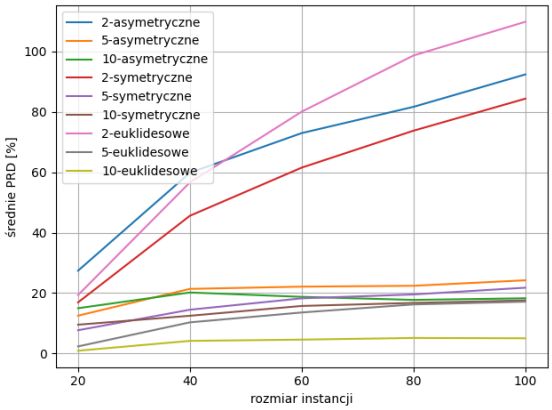
Zużycie pamięci pozostaje niezmienne. Początkowy rozkład feromonu nie wydaje się także wpływać na czas działania algorytmu. Najkrócej algorytm działa dla instancji asymetrycznych, a najdłużej dla symetrycznych.

5.5.3. Współczynnik β

5.5.3.1. Średnie wyniki oraz PRD



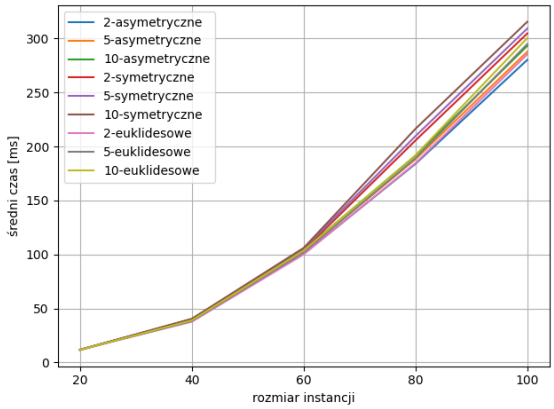
Rys. 5.23. Wsp. β - wyniki



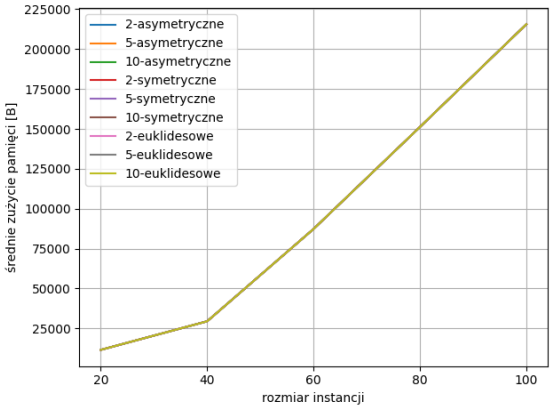
Rys. 5.24. Wsp. β - PRD

Dla tak dobranych wielkości współczynnika β jakość wyników rośnie wraz z jego zwiększaniem. Jedynie dla najmniejszych instancji czasami lepsze wyniki zwraca dla wartości 5.

5.5.3.2. Średni czas i zużycie pamięci



Rys. 5.25. Wsp. β - czas



Rys. 5.26. Wsp. β - pamięć

Współczynnik β nie ma wpływu na zużytą pamięć, ale czas działania algorytmu rośnie wraz z jego zwiększaniem.

5.5.4. Dobrane parametry

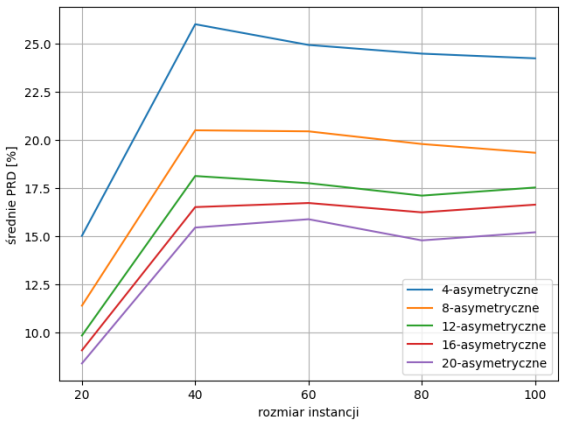
Dobrane parametry są zgodne z oczekiwaniami. Współczynnik odparowania jest dobrany bez żadnego schematu. Parametr β jest najczęściej równy 10, a czasem także 5. Początkowy rozkład feromonu jest prawie zawsze równomierny.

Typ	Rozmiar	W. odparowywania	P. rozkład feromonu	W. β
symetryczny	20	50	równomierny	5
symetryczny	40	80	równomierny	10
symetryczny	60	20	równomierny	10
symetryczny	80	80	losowy	10
symetryczny	100	90	równomierny	10
asymetryczny	20	50	równomierny	5
asymetryczny	40	80	równomierny	10
asymetryczny	60	90	równomierny	10
asymetryczny	80	20	równomierny	10
asymetryczny	100	80	równomierny	10
euklidesowy	20	80	równomierny	10
euklidesowy	40	90	równomierny	10
euklidesowy	60	20	równomierny	10
euklidesowy	80	20	równomierny	10
euklidesowy	100	80	równomierny	10

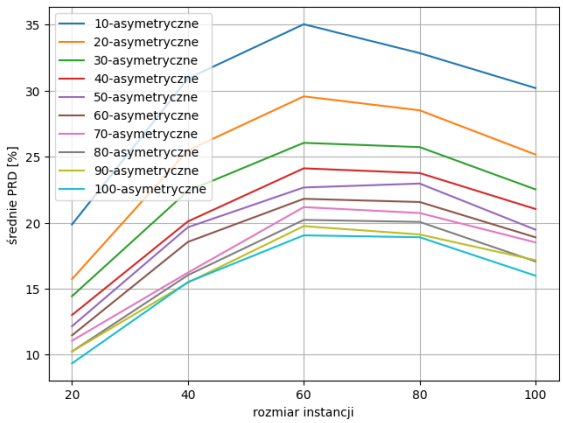
Tabela 5.3. Algorytm kolonii mrówek - dobrane parametry

5.5.5. Dodatkowe badania

Wygenerowane zostały osobno dwa zestawy instancji aby upewnić się czy zwiększanie liczby iteracji oraz liczby mrówek poprawia wyniki. Powinno się tak okazać, ponieważ algorytm zapamiętuje najlepsze rozwiązanie znalezione w czasie działania algorytmu, a zwiększanie tych dwóch parametrów zwiększa liczbę przeglądanych rozwiązań.



Rys. 5.27. Liczba iteracji - PRD



Rys. 5.28. Liczba mrówek - PRD

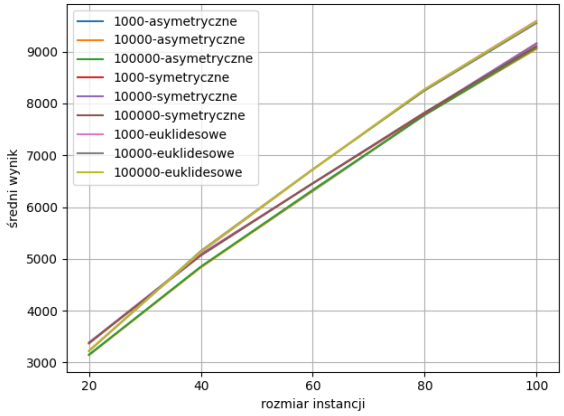
Przewidywania potwierdziły się. Dla czytelności przedstawione zostały wyniki tylko dla instancji asymetrycznych, ale dla pozostałych typów wnioski są identyczne. Różnice w wynikach maleją wraz ze wzrostem badanych statystyk. Nie zostały zamieszczone wykresy czasu oraz zużycia pamięci, ale również rosną one wraz ze wzrostem wartości tych dwóch parametrów.

5.6. ALGORYTM SYMULOWANEGO WYŻARZANIA

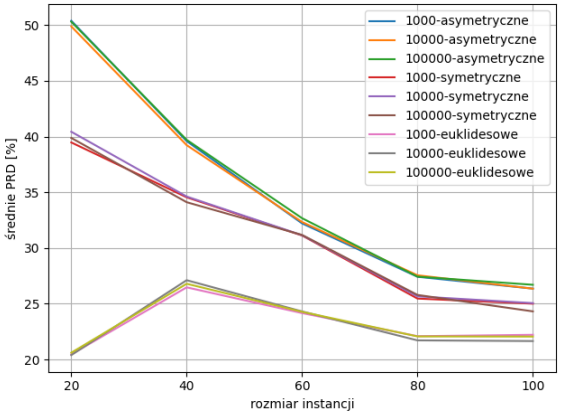
Dla tego algorytmu zbadane zostały dwa parametry. Pierwszy z nich, początkowa temperatura, został porównany dla wartości $1000, 10000, 100000$, a drugi, czyli współczynnik schładzania dla $20, 80, 90$ oraz 99 procent. Liczba iteracji była zawsze równa 100000. W każdej iteracji rozwiązanie było modyfikowane tylko jeden raz.

5.6.1. Początkowa temperatura

5.6.1.1. Średni wynik oraz PRD



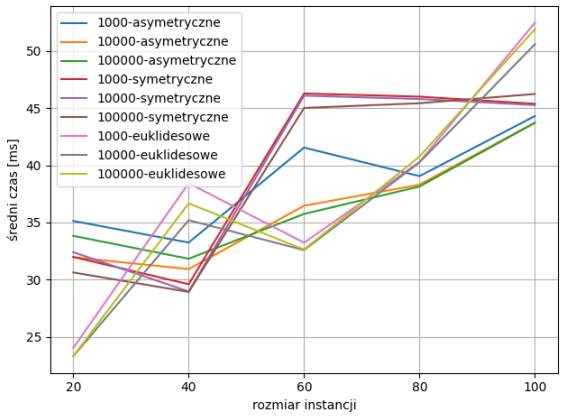
Rys. 5.29. Początkowa temperatura - wyniki



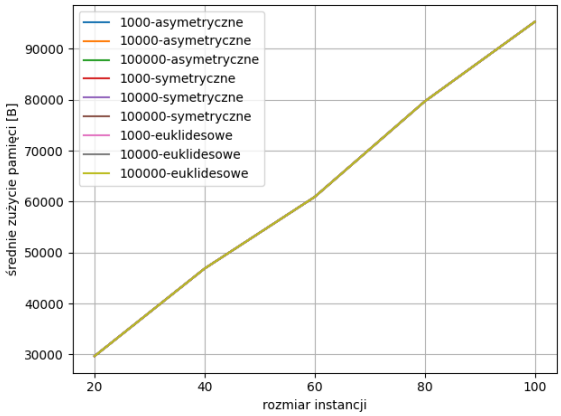
Rys. 5.30. Początkowa temperatura - PRD

Na wykresach nie widać żadnej zależności początkowej temperatury od jakości wyników. Dla przejrzystości wygenerowane zostały osobne wykresy dla każdego z typów instancji, ale również nie pozwoliło to wyciągnąć żadnych wniosków, więc nie zostały zamieszczone w pracy.

5.6.1.2. Średni czas i zużycie pamięci



Rys. 5.31. Początkowa temperatura - czas

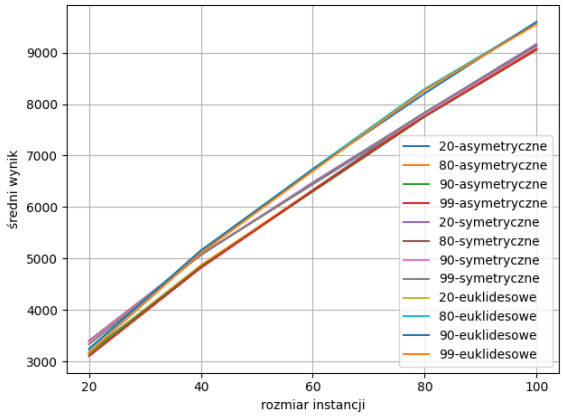


Rys. 5.32. Początkowa temperatura - pamięć

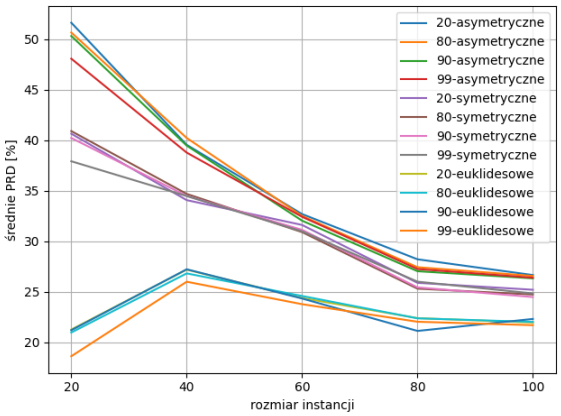
Zużycie pamięci jest takie samo dla każdego typu instancji oraz początkowej temperatury. Czas nie ma stałej wartości, ale nie widać żadnej zależności od temperatury początkowej.

5.6.2. Współczynnik schładzania

5.6.2.1. Średni wynik oraz PRD

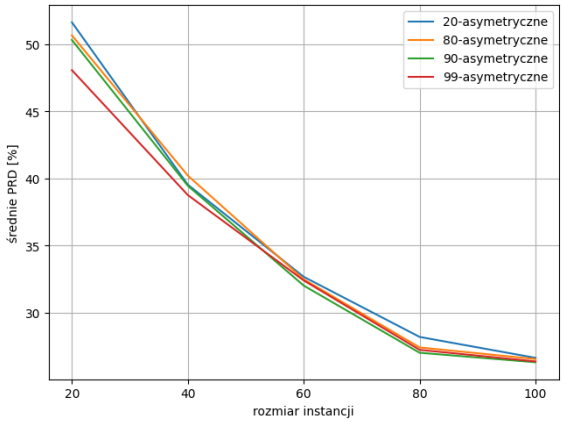


Rys. 5.33. Wsp. schładzania - wyniki

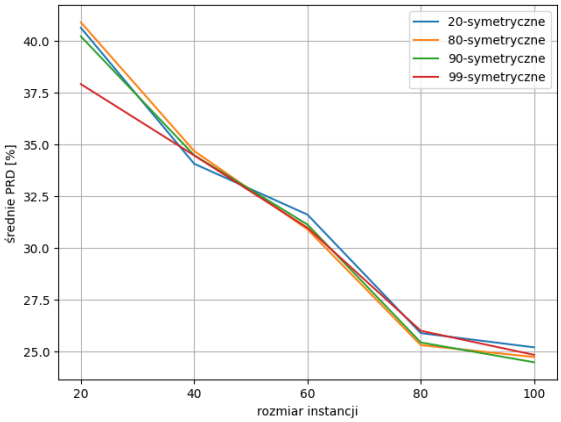


Rys. 5.34. Wsp. schładzania - PRD

Algorytm dla małych instancji zwraca najlepsze wyniki dla współczynnika 99. Dla większych lepsze wyniki zwykle zwraca dla wartości 90. Dla przejrzystości wygenerowane zostały dodatkowo osobne wykresy PRD dla instancji asymetrycznych i symetrycznych.



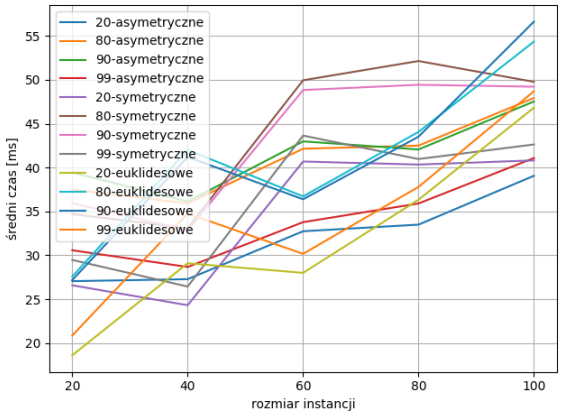
Rys. 5.35. Instancje asymetryczne - PRD



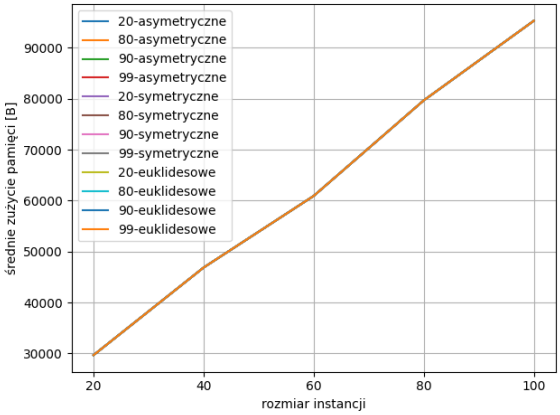
Rys. 5.36. Instancje symetryczne - PRD

Dla większych rozmiarów instancji symetrycznych otrzymujemy lepsze wyniki także dla współczynnika równego 80. Dla wartości 20 algorytm wydaje się działać średnio najgorzej.

5.6.2.2. Średni czas i zużycie pamięci



Rys. 5.37. Wsp. schładzania - czas



Rys. 5.38. Wsp. schładzania - pamięć

Zużycie pamięci nie zmienia się. Dla współczynników 80 i 90 algorytm działa ewidentnie najdłużej, a dla 20 najszybciej (dla każdego z trzech typów instancji).

5.6.3. Dobrane parametry

Typ	Rozmiar	P. temperatura	W. schładzania
symetryczny	20	10000	99
symetryczny	40	100000	20
symetryczny	60	10000	99
symetryczny	80	1000	90
symetryczny	100	100000	99
asymetryczny	20	10000	99
asymetryczny	40	100000	99
asymetryczny	60	1000	90
asymetryczny	80	100000	90
asymetryczny	100	1000	99
euklidesowy	20	10000	99
euklidesowy	40	100000	99
euklidesowy	60	1000	99
euklidesowy	80	100000	90
euklidesowy	100	1000	80

Tabela 5.4. Algorytm symulowanego wyżarzania - dobrane parametry

Dobrane parametry pokrywają się w większości z wyciągniętymi wnioskami. Najczęściej dobierany współczynnik schładzania to 99, a dla niektórych większych instancji dobrana

została wartość 90. Jedyne wyjątki to dwa rodzaje instancji, dla których dobrane zostały współczynniki 20 i 80. Jeżeli chodzi o temperaturę początkową to nie widać żadnego schematu tak jak oczekiwano.

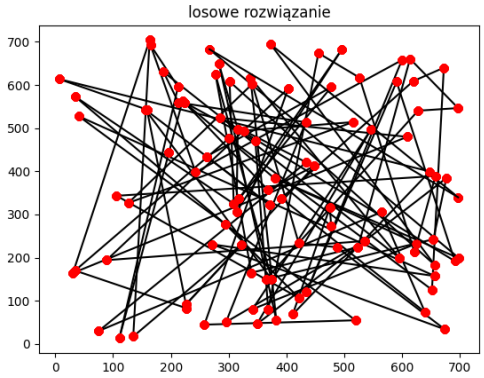
6. PORÓWNANIE ALGORYTMÓW

6.1. WSTĘP

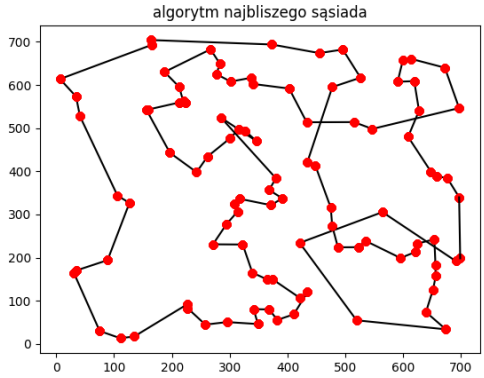
W tym rozdziale porównane zostały między sobą omówione w pracy algorytmy. Parametry zostały dobrane na podstawie badań z poprzedniego rozdziału. Parametry były dobierane pod kątem najlepszego średnie PRD. Wszystkie algorytmy były porównywane na jednym wspólnym zestawie instancji. Zbadane zostały instancje symetryczne, asymetryczne oraz euklidesowe o rozmiarach 20,40,60,80,100. Zakres wartości wynosił zawsze 1000. Dla każdej takiej konfiguracji wygenerowane zostało 20 instancji, co łącznie daje 300 instancji w całym zestawie. Dla każdej z nich wykonane zostało 20 powtórzeń algorytmu, a następnie wyciągnięto z nich średnią. Oprócz wyników badań dodatkowo zostały przedstawione przykładowe wizualizacje rozwiązań każdego z algorytmów dla instancji euklidesowych.

6.2. WIZUALIZACJA ROZWIĄZAŃ

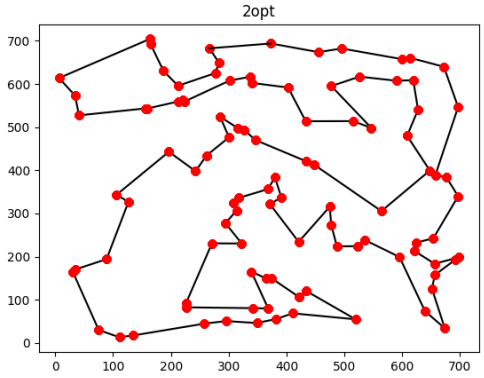
Na początek przedstawione zostały przykładowe wizualizacje rozwiązań dla instancji euklidesowych. Dla pozostałych typów instancji nie było to możliwe ze względu na to, że nie da się ich przedstawić na płaszczyźnie. Instancja, na której przedstawione zostały rozwiązania ma 100 wierzchołków i została wygenerowana losowo. Współrzędne punktów były losowane tak aby długość krawędzi nie przekraczała 1000. Oznacza to, że maksymalna wartość obu współrzędnych punktu wyniosła $1000/\sqrt{2}$, czyli około 707. W ten sam sposób generowane były wszystkie instancje euklidesowe, które zostały zbadane w pracy. Ze względu na to, że są to pojedyncze przykładowe rozwiązania mają one jedynie charakter poglądowy. Czerwone punkty reprezentują wierzchołki, a czarne odcinki łączące je krawędzie. Przykładowa instancja oraz rozwiązania zostały wygenerowane z wykorzystaniem programu, który służył do przeprowadzenia badań. Wymagało to jego lekkiej modyfikacji, ponieważ oprócz ocen rozwiązań musiał on dodatkowo zwrócić same trasy. Podobnie poza samą macierzą odległości konieczne było także zapisanie współrzędnych każdego wierzchołka. Wizualizacje zostały stworzone za pomocą biblioteki $matplotlib$ w języku Python.



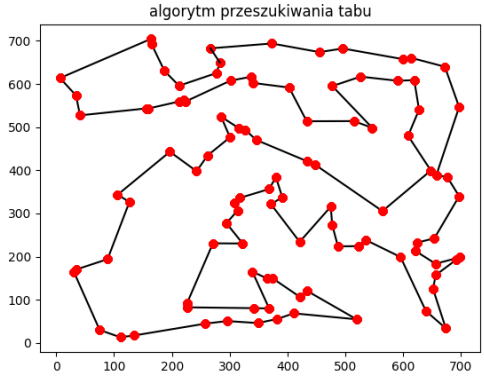
Rys. 6.1. Wizualizacja - losowa trasa



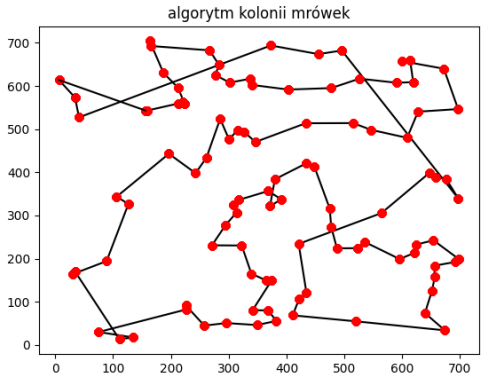
Rys. 6.2. Wizualizacja - najbliższy sąsiad



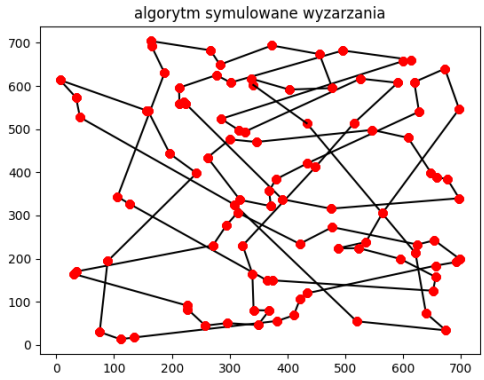
Rys. 6.3. Wizualizacja - 2OPT



Rys. 6.4. Wizualizacja - tabu



Rys. 6.5. Wizualizacja - kolonia mrówek

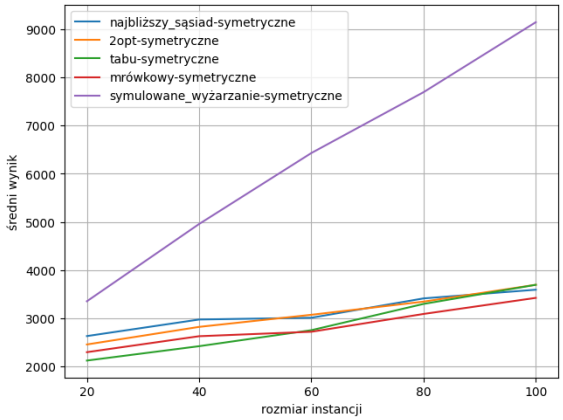


Rys. 6.6. Wizualizacja - symulowane wyżarzanie

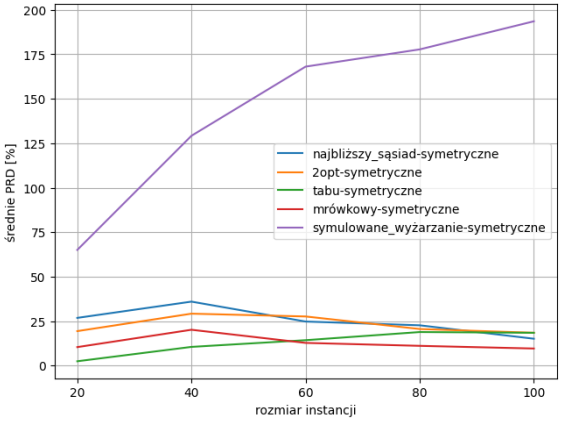
Im mniej chaotyczne trasy tym mniejszy jest ich łączny dystans. Najbardziej optymalne trasy zwrócił 2OPT oraz algorytm przeszukiwania tabu. Najmniej optymalna jest oczywiście trasa losowa, która mocno różni się od pozostałych. Dosyć chaotyczna trasa względem pozostałych została także wygenerowana przez algorytm symulowanego wyżarzania.

6.3. INSTANCJE SYMETRYCZNE

6.3.1. Średni wynik i PRD



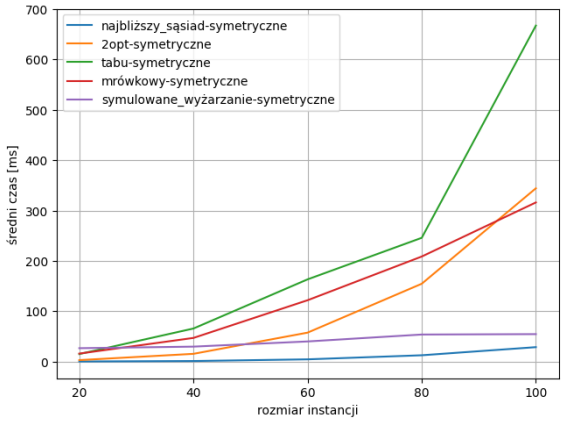
Rys. 6.7. Instancje symetryczne - wyniki



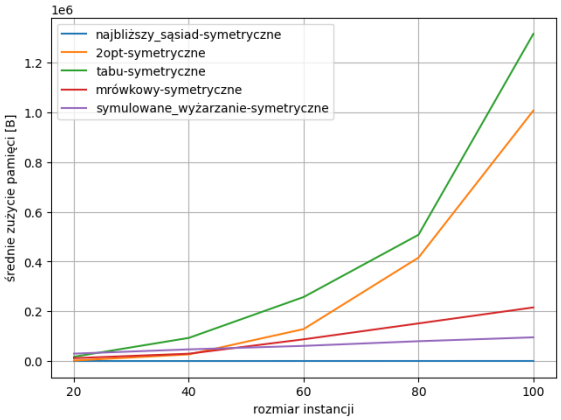
Rys. 6.8. Instancje symetryczne - PRD

Zdecydowanie gorsze wyniki od pozostałych algorytmów zwraca algorytm symulowanego wyżarzania. Najlepsze wyniki dla mniejszych instancji daje algorytm przeszukiwania tabu, a dla większych algorytm kolonii mrówek.

6.3.2. Średni czas i zużycie pamięci



Rys. 6.9. Instancje symetryczne - czas

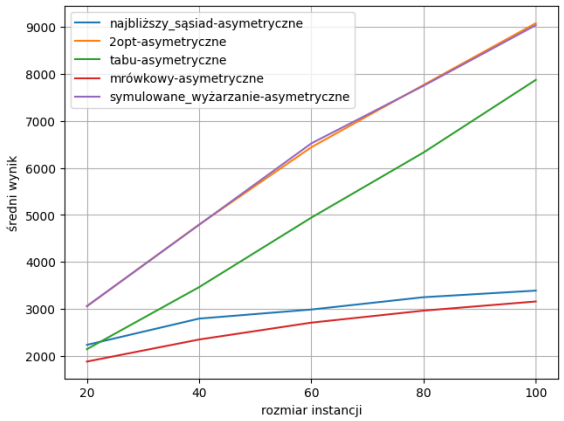


Rys. 6.10. Instancje symetryczne - pamięć

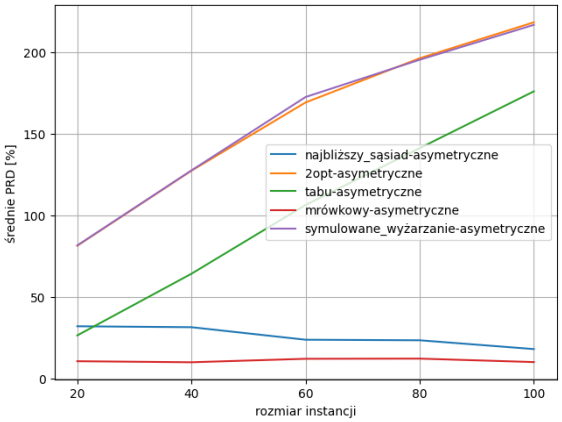
Najdłużej działa algorytm przeszukiwania tabu, następnie algorytm kolonii mrówek oraz 2OPT. Najkrócej działa rozszerzony algorytm najbliższego sąsiada, ale niewiele dłużej od niego działa też algorytm symulowanego wyżarzania. Jeśli chodzi o zużycie pamięci sytuacja wygląda podobnie, z wyjątkiem tego że algorytm kolonii mrówek zużywa zdecydowanie mniej pamięci niż 2OPT, ale nadal więcej niż algorytm symulowanego wyżarzania.

6.4. INSTANCJE ASYMETRYCZNE

6.4.1. Średni wynik i PRD



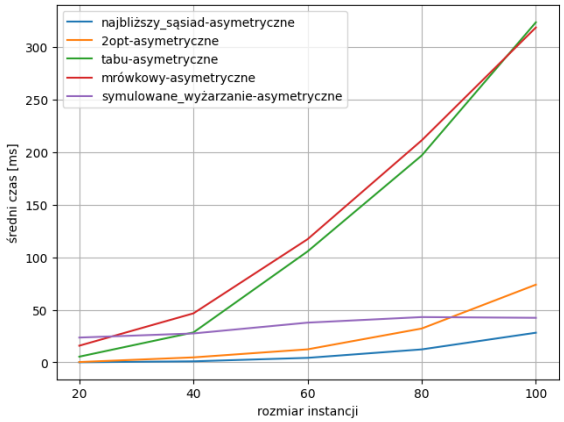
Rys. 6.11. Instancje asymetryczne - wyniki



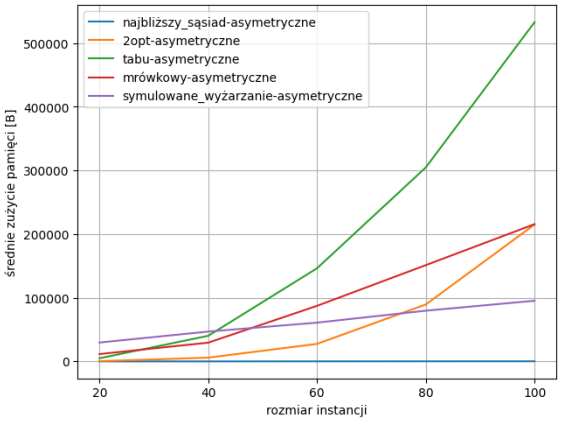
Rys. 6.12. Instancje asymetryczne - PRD

Najlepiej działa algorytm kolonii mrówek, ale niedużo gorzej także algorytm najbliższego sąsiada. Zdecydowanie najgorsze wyniki zwraca algorytm symulowanego wyżarzania oraz 2OPT.

6.4.2. Średni czas i zużycie pamięci



Rys. 6.13. Instancje asymetryczne - czas

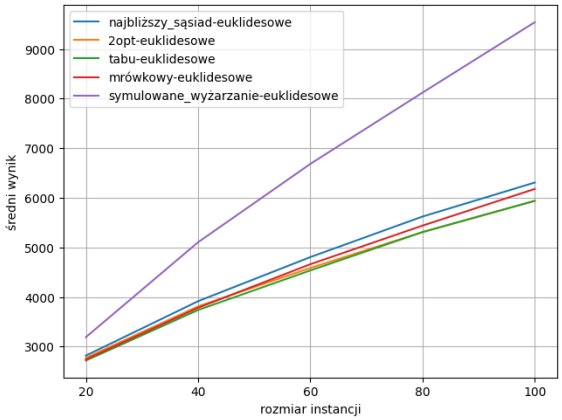


Rys. 6.14. Instancje asymetryczne - pamięć

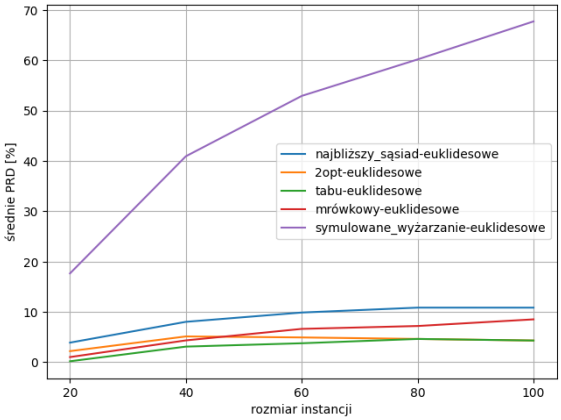
Najdłużej działa algorytm przeszukiwania tabu oraz algorytm kolonii mrówek. Najkrócej ponownie algorytm najbliższego sąsiada. Algorytm 2OPT działa krócej od algorytmu symulowanego wyżarzania (z wyjątkiem instancji o rozmiarze 100). Najwięcej pamięci zużywa zdecydowanie algorytm przeszukiwania tabu, a najmniej rozszerzony algorytm najbliższego sąsiada.

6.5. INSTANCJE EUKLIDESOWE

6.5.1. Średni wynik i PRD



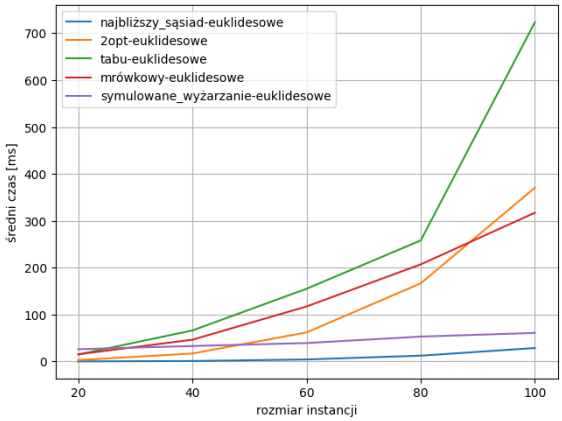
Rys. 6.15. Instancje euklidesowe - wyniki



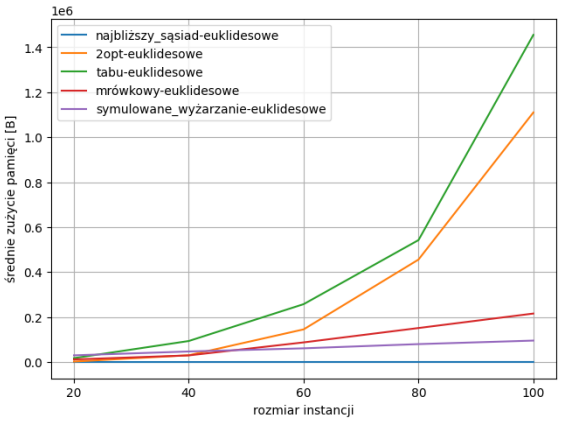
Rys. 6.16. Instancje euklidesowe - PRD

Dla instancji euklidesowych najlepiej działa algorytm przeszukiwania tabu. Dla większych instancji algorytm 2OPT osiąga tak samo dobre wyniki jak on. Ponownie najgorsze wyniki zwraca algorytm symulowanego wyżarzania. Algorytm kolonii mrówek zwraca lepsze wyniki niż algorytm najbliższego sąsiada.

6.5.2. Średni czas i zużycie pamięci



Rys. 6.17. Instancje euklidesowe - czas



Rys. 6.18. Instancje euklidesowe - pamięć

Najdłużej działa algorytm przeszukiwania tabu, następnie algorytm kolonii mrówek oraz 2OPT. Najkrócej działa rozszerzony algorytm najbliższego sąsiada, ale niewiele dłużej od niego działa też algorytm symulowanego wyżarzania. Jeśli chodzi o zużycie pamięci sytuacja wygląda podobnie, z wyjątkiem tego że algorytm kolonii zużywa mniej pamięci niż 2OPT, ale nadal więcej niż algorytm symulowanego wyżarzania.

6.6. DOBRANE ALGORYTMY

Na podstawie średniego PRD stworzona została tabelka z dobranymi algorytmami dla konkretnych rodzajów instancji.

Typ	Rozmiar	Zakres wartości	Algorytm
symetryczny	20	1000	tabu
symetryczny	40	1000	tabu
symetryczny	60	1000	mrówkowy
symetryczny	80	1000	mrówkowy
symetryczny	100	1000	mrówkowy
asymetryczny	20	1000	mrówkowy
asymetryczny	40	1000	mrówkowy
asymetryczny	60	1000	mrówkowy
asymetryczny	80	1000	mrówkowy
asymetryczny	100	1000	mrówkowy
euklidesowy	20	1000	tabu
euklidesowy	40	1000	tabu
euklidesowy	60	1000	tabu
euklidesowy	80	1000	2opt
euklidesowy	100	1000	2opt

Tabela 6.1. Dobrane algorytmy

Dobrane algorytmy pokrywają się z wyciągniętymi wnioskami. Dla największych instancji euklidesowych w tabeli znalazł się algorytm 2opt. Osiągnął on dla nich dokładnie takie same wyniki jak algorytm przeszukiwania tabu.

7. PODSUMOWANIE

7.1. WSTĘP

Przeprowadzone badania pozwoliły na wyciągnięcie wielu wniosków. W tym rozdziale podsumowane zostały najważniejsze z nich, dotyczące głównie jakości rozwiązań. Ze względu na eksperymentalną formę badań uzyskane wnioski w praktyce niekoniecznie sprawdzą się we wszystkich przypadkach, ale mimo to mogą być przydatne podczas dobierania algorytmów oraz ich parametrów dla problemu komiwojażera.

7.2. PARAMETRY ALGORYTMÓW

7.2.1. Algorytm 2-OPT

Algorytm 2OPT dla instancji asymetrycznych lepiej działa z operatorem $swap$, ale dla pozostałych typów lepsze wyniki zwraca dla operatora $invert$.

7.2.2. Algorytm przeszukiwania tabu

Dla algorytmu przeszukiwania tabu, wnioski dotyczące operatora modyfikacji są takie same jak dla 2OPT. Dla instancji asymetrycznych lepiej dobrać większą długość listy tabu w celu poprawienia wyników. Dla pozostałych typów wpływ ten jest mniej zauważalny, ale dla instancji o rozmiarze mniejszym niż 40 również warto dobrać większe długości listy. Algorytm dla najmniejszych długości listy (poniżej 5) zwraca najgorsze wyniki dla wszystkich typów instancji.

7.2.3. Algorytm kolonii mrówek

Dla algorytmu kolonii mrówek lepiej stosować równomierny początkowy rozkład feromonu niż losowy. Współczynnik β powinien być zdecydowanie większy od współczynnika alfa. Zwiększanie liczby iteracji oraz mrówek także poprawia wyniki, ale kosztem czasu i zużycia pamięci.

7.2.4. Algorytm symulowanego wyżarzania

Dla algorytmu symulowanego wyżarzania najlepiej stosować współczynnik schładzania bardzo bliski 1.0, szczególnie dla małych instancji. Dla większych rozmiarów instancji dobrze sprawdza się także współczynnik 0.9.

7.3. PORÓWNANIE ALGORYTMÓW

7.3.1. Instancje symetryczne

Dla instancji symetrycznych najlepiej stosować algorytm przeszukiwania tabu dla instancji o rozmiarze poniżej 50 oraz algorytm kolonii mrówek dla większych instancji (badania zostały przeprowadzone maksymalnie dla rozmiaru 100). Jeżeli zależy nam na czasie to dosyć dobre wyniki w dużo krótszym czasie zwraca także rozszerzony algorytm najbliższego sąsiada.

7.3.2. Instancje asymetryczne

Dla instancji asymetrycznych najlepiej stosować algorytm kolonii mrówek. Niedużo gorsze wyniki w dużo krótszym czasie zwraca także rozszerzony algorytm najbliższego sąsiada.

7.3.3. Instancje euklidesowe

Dla instancji euklidesowych najlepiej stosować algorytm przeszukiwania tabu. Algorytm kolonii mrówek działa trochę gorzej, ale za to szybciej. Jeszcze gorzej (ale dalej dobrze) i zdecydowanie szybciej od pozostałych działa algorytm najbliższego sąsiada.

7.4. UWAGA

Ze względu na krótki czas działania algorytmu symulowanego wyżarzania dla zadanych mu parametrów, sprawdzone zostało jak wpłynie na jego wyniki kilkukrotne zwiększenie liczby iteracji. Otrzymane wyniki nadal były zdecydowanie gorsze od pozostałych algorytmów. Nie oznacza to jednak, że nie warto z niego korzystać dla problemu komiwojażera. Łącząc go z innymi algorytmami, korzystając z bardziej skomplikowanych operatorów modyfikacji lub lepiej dostrajając parametry związane z temperaturą można prawdopodobnie osiągnąć równie dobre lub nawet lepsze wyniki niż pozostałe algorytmy omówione w tej pracy.

BIBLIOGRAFIA

[1] https://en.wikipedia.org/wiki/Julia_(programming_language).

[2] https://en.wikipedia.org/wiki/Python_(programming_language).

[3] A. Debudaj-Grabysz, S. Deorowicz, J.W., *Algorytmy i struktury danych. wybór zaawansowanych metod*. 2012.

[4] D.S. Johnson, L.M., *The traveling salesman problem: A case study in local optimization*. 1995.

[5] Gilbert, S., *Optimization algorithms: The traveling salesman*. 2015.

[6] Gospodarczyk, P., *Przeszukiwanie z tabu*. 2011.

[7] Gutin, G., Yeo, A., Zverovich, A., *Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp*, Discrete Applied Mathematics. 2002.

[8] Hannu Toivonen, Fang Zhou, A.H.A.H., *Compression of weighted graphs*. 2011.

[9] Jakubowska, A., Piechocka, K., *W poszukiwaniu optymalnej trasy - wybrane algorytmy w zastosowaniu do problemu komiwojażera*, Journal of Translogistics. 2015.

[10] Rembelski, P., *Algorytmy i struktury danych - wykład i wstęp do algorytmów*. 2009.

[11] Szlachcic, E., *Teoria i metody optymalizacji*.

[12] Thomas H. Cormen, Charles E. Leiserson, R.L.R.C.S., *Wprowadzenie do algorytmów* (1990).

[13] Wilson, R.J., *Introduction to graph theory* (1970).

[14] Łukasz, S., *Adaptacyjny algorytm optymalizacji stadnej cząsteczek dla dynamicznego problemu komiwojażera*. 2017.

SPIS RYSUNKÓW

SPIS LISTINGÓW

SPIS TABEL