AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA AUTOMATYKI I ROBOTYKI

Projekt dyplomowy

Zmodyfikowany algorytm ewolucji różnicowej dla problemu QAP
A modified differential evolution algorithm for the QAP problem

Autor: *Karolina Sroczyk*
Kierunek studiów: *Automatyka i Robotyka*
Opiekun pracy: *dr inż. Wojciech Chmiel*

Kraków, 2020

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Rodzicom, za wszystkie lata nauki, dedykuję i dziękuję.

Spis treści

1. Wstęp

Główny cel pracy stanowi opracowanie i przeanalizowanie poprawności działania zmodyfikowanego algorytmu ewolucji różnicowej przystosowanego do rozwiązywania problemu kwadratowego zagadnienia przydziału (Quadratic Assigment Problem). Ze względu na fakt, iż QAP należy do klasy problemów NP-trudnych, a więc problemów, dla których nie istnieje jednoznacznie określony algorytm dający optymalne wyniki, istnieje konieczność opracowania algorytmu przynoszącego satysfakcjonujące efekty w przypadku konkretnie postawionego zagadnienia. Dobrych wyników optymalizacji można spodziewać się wówczas gdy opracowany algorytm wykorzystuje specyficzne cechy oraz regularności rozwiązywanego problemu.

W celu uzyskania satysfakcjonujących wyników dla problemów NP-trudnych, można posłużyć się metodami metaheurystycznymi, a więc metodami umożliwiającymi iteracyjne poprawianie wyniku na podstawie przyjętych uprzednio funkcji oceny. Metody metaheurystyczne w dużej mierze inspirują się zjawiskami zachodzącymi w naturze, z tego względu omawiając je operuje się nomenklaturą zaczerpniętą ze świata biologii. Dają one jednak rozwiązanie przybliżone, co oznacza, iż nie ma pewności, że uzyskany wynik jest minimum globalnym. Jedną z metod często stosowaną w celu rozwiązywania problemów optymalizacji globalnej jest algorytm ewolucji różnicowej. Algorytm ten w dużej mierze oparty jest na losowości oraz rozwiązaniach o charakterze intuicyjnym, niemniej jednak dających oczekiwane rezultaty. Niejednokrotnie dokonuje się modyfikacji tego algorytmu w zależności od potrzeb konkretnie rozwiązywanego problemu. Modyfikacje klasycznej wersji algorytmu najczęściej wykazują swoją skuteczność w przypadku działania w kontekście jednego wybranego zagadnienia. Natomiast w przypadku zastosowania ich dla innych problemów skuteczność ta ewidentnie spada. W przedmiotowej pracy przeprowadzona zostanie analiza wpływu poszczególnych modyfikacji na wyniki osiągane przez algorytm, tak by móc wyodrębnić spośród nich strategie osiągające najkorzystniejsze wyniki.

Algorytmy wchodzące w skład metaheurystyk charakteryzują się posiadaniem rozwiązania będącego wektorem wartości binarnych. Ze względu na fakt, iż algorytm rozważany w niniejszej pracy ma zostać przystosowany do optymalizacji problemów QAP, którego rozwiązanie jest w postaci wektora będącego permutacją pewnego zbioru, należy dokonywać modyfikacji klasycznych metod również na tej płaszczyźnie.

Etapy realizacji celu pracy obejmują:

– Zdefiniowanie NP-trudnego problemu, jaki stanowi kwadratowe zagadnienie przydziału (Quadratic Assigment Problem) oraz określenie relacji, jakie zachodzą pomiędzy zmodyfikowanym algorytmem ewolucji różnicowej a problemem, który rozwiązuje. Opracowanie modelu matematycznego opisującego problem QAP i stanowiącego podstawę dalszych rozważań.

– Implementacja podstawowej wersji algorytmu ewolucji różnicowej wraz z wprowadzeniem modyfikacji w algorytmie w celu dostosowania go do rozwiązywania problemu QAP. Modyfikacje dotyczą kwestii opracowania reprezentacji pojedynczego osobnika jako permutacji pewnego zbioru.

– Opracowanie algorytmu ewolucji różnicowej wraz z uwzględnieniem modyfikacjami w obrębie metod reprodukcji, mutacji oraz krzyżowania. Zmiany dotyczą głównie operacji mutacji oraz krzyżowania. Dodatkowo w tych operacjach genetycznych badaniu podlegają odpowiednie współczynniki.

– Badanie skuteczności zaproponowanych wersji zmodyfikowanego algorytmu ewolucji różnicowej oraz porównanie z działaniem podstawowej wersji. Oprócz konfrontacji wyników wersji klasycznej i zmodyfikowanej ważnym czynnikiem jest liczba iteracji, w których algorytm osiąga wartość uznawaną przez bibliotekę [9] za najmniejszą wartość, jaką udało się uzyskać dla konkretnej instancji danych wejściowych.

– Przeprowadzanie testów algorytmu dla bardziej oraz mniej złożonych problemów. Rozważane dane wejściowe posiadają inny rozmiar oraz osiągają inne średnie wartości funkcji celu.

Niniejsza praca składa się z 7 rozdziałów, z których 5 początkowych zawiera informacje teoretyczne. Opisana została w nich zarówno klasyczna, jak i zmodyfikowana wersja algorytmu ewolucji różnicowej. Rozdział 6 stanowi część badawczą, gdyż został przeznaczony na testy uprzednio opisanych modyfikacji. Wnioski i spostrzeżenia, a także informacje o najlepiej działających wersjach algorytmu zostały umieszczone w rozdziale 7.
Rozdział 2 zawiera informacje teoretyczne dotyczące problemów optymalizacji zarówno lokalnej, jak i globalnej. Został w nim opisany oraz przedstawiony schemat algorytmu ewolucji różnicowej. Zawarto także opis nomenklatury dotyczącej algorytmów ewolucyjnych. Drugą część rozdziału stanowi opis oraz definicja kwadratowego zagadnienia przydziału QAP.
Na rozdział 3 składają się opisy zasady działania poszczególnych metod reprodukcji rozważanych w pracy. Zawarte zostały także implementacje poszczególnych metod.
Rozdział 4 został poświęcony metodom mutacji w wersji klasycznej, jak i zmodyfikowanej. Zawarte zostały także opisy możliwości modyfikacji współczynnika mutacji.
W rozdziale 5 zaprezentowano metody krzyżowania stosowane w podstawowych implementacjach algorytmów oraz te mające bezpośrednie zastosowanie w przypadku problemów, których podstawową

jednostką jest osobnik będący permutacją pewnego zbioru. Zostały również opisane modyfikacje współczynnika krzyżowania.
W rozdziale 6 przeprowadzono testy wszystkich opisanych metod oraz modyfikacji. Na podstawie uzyskanych wyników ustalona została wersja algorytmu uzyskująca najlepsze efekty. Następnie z wykorzystaniem tej wersji algorytmu przeprowadzone zostały testy z wykorzystaniem innych instancji danych wejściowych.
Rozdział 7 stanowi zakończenie i podsumowanie pracy. Znajdują się tam informacje dotyczące wersji algorytmu uznanych za najskuteczniejsze. Dodatkowo można znaleźć informacje na temat dalszych kierunków badań.

2. Wstęp teoretyczny

2.1. Opis algorytmu ewolucji różnicowej

Algorytm ewolucji różnicowej jest rodzajem heurystyki, a więc rodzajem algorytmu, dla którego znalezione rozwiązanie nie jest objęte gwarancją bycia rozwiązaniem optymalnym. Metod heurystycznych używa się w sytuacji, gdy przestrzeń potencjalnych rozwiązań jest na tyle duża, iż nie jest możliwe w dostatecznie krótkim czasie przebadanie wszystkich potencjalnych rozwiązań w celu znalezienia tego najlepszego. Algorytmy ewolucyjne, których to pewnego rodzaju odmianą są algorytmy ewolucji różnicowej [11], są jednym z narzędzi służących do ukierunkowania procesu poszukiwań poprzez zastosowanie technik probabilistycznych, a co za tym idzie służących do usprawnienia rozwiązywania wspomnianej wyżej sytuacji. W większości przypadków, przy odpowiednim doborze parametrów rozwiązanie będące wynikiem działania algorytmu ewolucyjnego jest zarazem rozwiązaniem optymalnym, nie mniej jednak zależność ta nie występuje we wszystkich przypadkach.

Swoim działaniem algorytmy ewolucyjne przypominają zjawisko ewolucji biologicznej, a kolejne etapy algorytmu stanowią operacje będące odpowiednikami zjawisk typowych dla świata przyrody, a więc zjawiska reprodukcji, mutacji oraz krzyżowania. Inspiracja zasadą doboru naturalnego sprowadza się do kierowania się ideą, iż tylko najlepiej przystosowane osobniki mają szanse na przeżycie i to one powinny zapoczątkowywać istnienie nowych co raz to lepszych osobników potomnych, które to z kolei tworzą nowe mocniejsze populacje. Dzięki temu algorytm ewolucyjny tworzy stopniowo coraz lepsze rozwiązania i może służyć on do rozwiązywania problemów optymalizacyjnych. Więcej informacji na ten temat można znaleźć w [7].

Obecnie w skład algorytmów ewolucyjnych wchodzą między innymi algorytmy genetyczne, programowanie genetyczne czy też programowanie ewolucyjne. Jednak jeszcze w latach sześćdziesiątych istniały i rozwijały się one niezależnie od siebie jako oddzielne wersje wcielające w życie darwinowską zasadę doboru naturalnego. Dopiero od początku lat 20 XX wieku strategie ewolucyjne, algorytm genetyczny, programowanie ewolucyjne i programowanie genetyczne zaczęły być traktowane jako różne formy tej samej techniki określane wspólnym mianem algorytmów ewolucyjnych [10]. Algorytm ewolucji różnicowej został zaproponowany w 1995 roku przez Kene'a Price. Główna cecha odróżniająca algorytm ewolucji różnicowej od klasycznego algorytmu ewolucyjnego dotyczy dokonywania operacji odejmowania od siebie wektorów w operacji mutacji, a więc tworzenia wektorów różnicowych.

2.1.1. Optymalizacja lokalna i globalna

Zgodnie z [1] problem optymalizacji może zostać w ogólności przedstawiony jako problem mający na celu znalezienie takiej wartości x w danym zbiorze X, dla której funkcja f(x) będąca funkcją celu, a więc funkcją mierzącą cel, który ma zostać osiągnięty, przyjmuje najkorzystniejszą wartość. Wartość najkorzystniejsza może być interpretowana w zależności od intencji jako minimum bądź też maksimum danej funkcji f(x). Minima i maksima są zbiorowo nazywane ekstremami odpowiednio globalnymi bądź lokalnymi.

Niech R oznacza zbiór wszystkich rozwiązań obranego problemu. Punkt x* w przestrzeni R jest minimum globalnym, jeśli dla każdego x należącego do R zachodzi zależność taka, że:

$$f(x*) \leq f(x) \tag{2.1}$$

Co można zapisać :

$$x = x* \Leftrightarrow \forall x \in R, f(x*) \leq f(x) \tag{2.2}$$

W takiej sytuacji x* jest optimum globalnym określającym najmniejszą wartość w całej dziedzinie. Nie mniej jednak łatwiejszym do znalezienia, rozwiązania i częściej formułowanym problemem jest zadanie określające minimum lokalne. Problem sprowadza się do znalezienia takiego x_1 dla którego

$$f(x_1) \leq f(x_1 + t), t > 0 \tag{2.3}$$

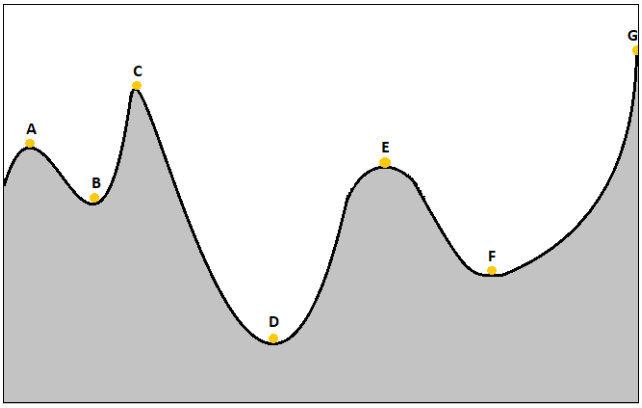
gdzie $x_1 + t$ są punktami otaczającymi x_1. Można to zdefiniować w następujący sposób:

$$x = x_1 \Leftrightarrow f(x_1) \leq f(x_1 + t), t \neq 0 \tag{2.4}$$

Na rysunku 2.1 zaznaczone są ekstrema funkcji spośród których możemy wyróżnić ekstrema lokalne oraz globalne. Odpowiednio punkty:

- A, C, E są maksimami lokalnymi,
- G jest maksimum globalnym,
- B, F są minimami lokalnymi,
- D jest minimum globalnym

Większość metod optymalizacyjnych ma zasięg lokalny, co znaczenie utrudnia poszukiwanie rozwiązania problemu, jako że algorytmy kończą swoje działanie w chwili, gdy wartość funkcji celu osiągnie wartość ekstremum lokalnego. Powodem tych problemów jest zależność metod optymalizacyjnych od pochodnych funkcji, które to z kolei nie są wystarczająco odporne na nieciągłości czy też rozległe wielomodalności.



Rys. 2.1. Przebieg funkcji wraz z zaznaczonymi ekstremami.

2.1.2. Podstawowe pojęcia

W terminologii algorytmów genetycznych używa się następujących pojęć zapożyczonych z genetyki naturalnej [1], [6]:

Osobnik - podstawowa jednostka charakteryzująca się pewnym przystosowaniem do środowiska w którym żyje czego miarą jest wartość funkcji celu obliczona dla tego osobnika. Dany osobnik, reprezentowany poprzez wektor liczb rzeczywistych, jest potencjalnym rozwiązaniem całego problemu optymalizacji.

Populacja - zbiór osobników ulegający zmianie w kolejnych iteracjach algorytmu jako, że słabsze osobniki są zastępowane przez nowe, lepiej przystosowane.

Genotyp - zbiór informacji przypisany do każdego osobnika indywidualnie, opisujący proponowane rozwiązanie problemu optymalizacyjnego.

Fenotyp - zbiór cech osobnika podlegających ocenie funkcji przystosowania.

Chromosom - jest to uporządkowany ciąg genów, który inaczej nazywany jest łańcuchem. W chromosomie zakodowany jest fenotyp i ewentualnie pewne informacje pomocnicze dla algorytmu genetycznego.

Gen - jest to pojedynczy element chromosomu.

Grupa rozrodcza - zbiór osobników służących jako podstawa do utworzenia osobnika potomnego.

Funkcja celu - funkcja pozwalająca określić miarę przystosowania i jakość konkretnego osobnika z punktu widzenia rozwiązywanego problemu. Jako argument przyjmuje ona wektor będący odpowiednikiem osobnika.

Funkcja dopasowania - jest to funkcja stanowiąca modyfikacje funkcji celu tak aby zachować zależność iż im większa wartość funkcji dopasowania dla danego osobnika tym osobnik jest lepszy.

Selekcja - technika mająca na celu wybór osobników, które przedostaną się do grupy rozrodczej tworzącej potomstwo.

Mutacja - technika mająca na celu utworzenie nowego osobnika na podstawie osobników znajdujących się w grupie rozrodczej .

Krzyżowanie - technika mająca na celu utworzenie nowego osobnika na podstawie osobnika wynikowego mutacji oraz osobnika z aktualnej populacji.

Operacje genetyczne - zbiór operacji takich jak selekcja, mutacja, krzyżowanie.

2.1.3. Pseudokod

Większość algorytmów optymalizacyjnych stosuje procedurę postępowania, w której to z każdym krokiem działania algorytmu coraz bardziej zbliżamy się do rozwiązania optymalnego. Algorytmy tego typu zazwyczaj rozpoczynają poszukiwanie startując z konkretnego punktu, a następnie, po uwzględnieniu dodatkowych informacji, przemieszczają się dalej dochodząc po wielu iteracjach do rozwiązania końcowego.

Podobnie jak w algorytmach ewolucyjnych początkiem działania algorytmu ewolucji różnicowej jest utworzenie pierwszej populacji osobników będących pierwszymi potencjalnymi rozwiązaniami problemu. W każdej iteracji osobniki są poddawane ocenie zgodnie z przyjętą odgórnie funkcją celu. W kontekście algorytmu ewolucji różnicowej osobnik zdefiniowany jest jako n-elementowy wektor liczb rzeczywistych będący permutacją n-elementowego zbioru:

$$\overrightarrow{x_i} = \{x_{i,1}, x_{i,2}, ...x_{i,n}\}, j = 1, 2, ..., n \tag{2.5}$$

Natomiast populacje można określić jako:

$$X = \{\overrightarrow{x_1}, \overrightarrow{x_2}, ...\overrightarrow{x_i}\}, i \ = 1, 2, ..., n \tag{2.6}$$

Następnie przeprowadzana jest reprodukcja, w wyniku której otrzymujemy grupę rozrodczą. Grupa rozrodcza stanowi podstawę do przeprowadzenia operacji genetycznych, jakimi są mutacja oraz krzyżowanie. Operacja mutacji dokonuje przekształceń genetycznych na danej grupie rozrodczej, w wyniku których powstaje nowy osobnik. Poddawany jest on operacji krzyżowania. Dopiero wynik tych dwóch operacji daje rezultat w postaci osobnika potomnego, którego współczynnik przystosowania zostaje oceniony. W sytuacji gdy będzie on większy od współczynnika przystosowania osobnika z obecnej populacji, nastąpi ich podmiana, a więc osobnik potomny zajmie w populacji miejsce poprzednika. Warunkiem zakończenia algorytmu może być na przykład brak zmienności wyniku w kolejnych n iteracjach bądź też osiągnięcie zadowalającego poziomu przystosowania.

Opisane zależności można w ogólności zaprezentować w postaci pseudokodu algorytmu przedstawionego poniżej.

Przyjęcie określonych wartości n, F, C_r
Inicjalizacja pierwszej populacji $X \leftarrow \{x_1, x_2, ..., x_n\}$

```
while ( !warunek stopu) do
    for i ∈ {1, 2, ..., n} do
        #reprodukcja
            x_{i,1} ← reprodukcja (i, X)
            x_{i,2} ← reprodukcja (i, X)
            x_{i,3} ← reprodukcja (i, X)
        #mutacja różnicowa
            u_i ← mutacja różnicowa (x_{i,1}, x_{i,2}, x_{i,3}, F)
        #krzyżowanie
            o_i ← krzyżowanie (x_i, u_i, C_r)
        #funkcja oceny
            if f(o_i) ≤ f((x_i) then
                x_i ← o_i
            end if
    end for
end while
return x_{minf(x_i)}
```

Na schemacie zauważalna staje się odmienność algorytmu ewolucji różnicowej od klasycznych algorytmów ewolucyjnych głównie w kontekście sposobu dokonywania operacji mutacji. Dobór metod reprodukcji, mutacji czy też krzyżowania nie może zostać jednoznacznie określony i być uniwersalny dla wszystkich problemów, które ma on rozwiązywać. Poprawność i jakość działania metod jest ściśle zależna od charakteru rozwiązywanego problemu. Z tego również względu w niektórych przypadkach konieczne jest dokonanie zmian w klasycznym algorytmie ewolucyjnym, w wyniku czego otrzymujemy zmodyfikowany algorytm ewolucji różnicowej.

2.2. Opis kwadratowego zagadnienia przydziału

Kwadratowe zagadnienie przydziału jest jednym z fundamentalnych problemów optymalizacyjnych zajmujących się problemami rozlokowania obiektów. Problem należy do klasy problemów NP-trudnych co oznacza, że nie ma znanego i jednoznacznie określonego algorytmu rozwiązującego ten typ problemu w czasie wielomianowym. Sprawdzenie wszystkich opcji w celu znalezienia rozwiązania najlepszego nawet małego rozmiaru problemu okazuje się problematyczne i długotrwałe.

Model matematyczny kwadratowego zagadnienia przydziału

Dany jest zbiór n budynków oraz zbiór n lokalizacji. Pomiędzy każdymi dwoma lokalizacjami znana jest odległość oraz wartość przepływu. Wartość przepływu można interpretować jako koszt transportu dóbr pomiędzy dwiema lokalizacjami. Celem zadania jest takie przypisanie budynków do danych lokalizacji, by zminimalizować sumę odległości pomiędzy każdymi dwoma budynkami pomnożoną przez odpowiadającą danej trasie wartość przepływu.

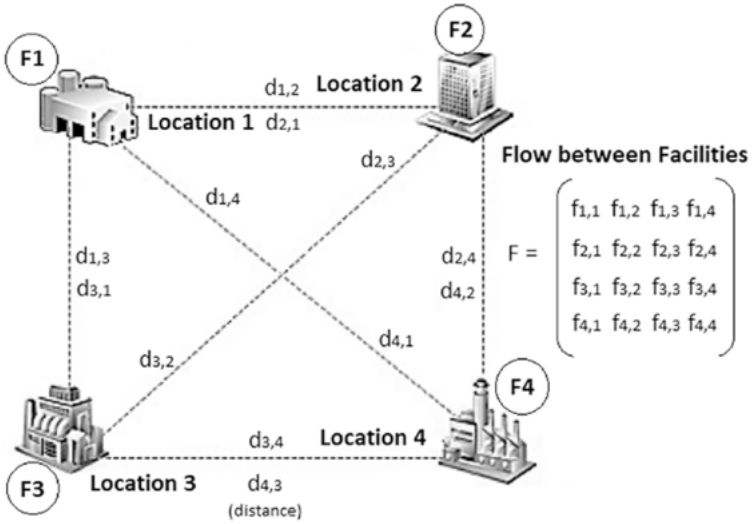
Formalna definicja kwadratowego zagadnienia przydziału jest następująca [4] :

Dane są dwie rzeczywiste macierze F = (f_{ij}) oraz D = (d_{ij}) o wymiarze $n \times n$ każda, gdzie odpowiednio:

f_{ij} - macierz przepływu pomiędzy punktami i oraz j,
d_{ij} - macierz odległości, określająca odległość pomiędzy punktami i oraz j Minimalizacja całkowitego kosztu działania systemu opisana jest następującą zależnością :

$$\sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} \cdot d_{\pi(i)\pi(j)} \longrightarrow min \tag{2.7}$$

Gdzie :
$\pi = \{\pi(1), ..., \pi(n)\}$ - permutacja zbioru N = {1, ... , N}
Należy zaznaczyć również fakt, iż jeden budynek może zostać przypisany do jednej lokalizacji oraz do jednej lokalizacji może zostać przypisany tylko jeden budynek. Poniżej zamieszczono rysunek obrazujący kwadratowe zagadnienie przydziału.



Rys. 2.2. Przedstawienie kwadratowego zagadnienia przydziału [3].

Opisana wzorem 2.7 zależność stanowi model matematyczny i funkcję celu będącą podstawą rozważań w dalszych rozdziałach. Permutacja będąca wynikiem działania algorytmu ewolucji różnicowej interpretowana jest jako pewien sposób rozlokowania *n* budynków w *n* lokalizacjach. Poszczególne wartości genów odpowiadają konkretnym przypisanym do nich na stałe budynkom. Natomiast pozycje o danych indeksach w wektorze wynikowym interpretowane są jako poszczególne lokalizacje.

3. Reprodukcja

Jednym z etapów algorytmu ewolucyjnego jest reprodukcja polegająca na wyborze osobników, które przedostaną się do grupy rozrodczej tworzącej potomstwo. W klasycznym algorytmie ewolucji różnicowej wybór wektorów bazowych oraz wektorów składających się na wektor różnicowy jest dokonywany losowo [12]. Nie mniej jednak w niniejszej pracy w celach doświadczalnych zostaną porównane i zestawione ze sobą dodatkowo metody doboru grupy rodzicielskiej zgodnie z dostępnymi metodami selekcji stosowanymi w algorytmach ewolucyjnych.

Znanych jest kilka metod selekcji, na podstawie których dokonuje się wyboru osobników z różnym prawdopodobieństwem. Metody te w większości zostały opracowane w taki sposób, by dawały większe szanse przetrwania osobnikom lepiej do tego przystosowanym, a więc osobnikom o większym współczynniku funkcji przystosowania. Metody selekcji zostały zaimplementowane w dwojaki sposób. Pierwszym z nich jest możliwość pojawienia się danego osobnika w grupie rodzicielskiej więcej niż jeden raz. Druga technika nakazuje różnorodność osobników w obrębie danej instancji grupy rozrodczej.

$$i_1 \neq i_2 \neq i_3 \neq \ldots \neq i_n \tag{3.1}$$

Gdzie:
i_n - n-ty osobnik

Poniżej przestawiono idee metod selekcji podlegających rozważaniom w ramach tej pracy. Implementacje metod powstały w oparciu o [6], [8]. Listingi poszczególnych metod reprodukcji można znaleźć w dodatku A.

3.1. Klasyczna reprodukcja : Losowy wybór

Z populacji wybierany jest osobnik przechodzący do grupy rozrodczej w oparciu o wylosowaną liczbę odpowiadającą indeksowi konkretnego osobnika. Metoda ta jest niezależna od wartości funkcji przystosowania, a więc daje równe szanse każdemu z osobników. Prawdopodobieństwo wylosowania danego osobnika, z populacji składającej się z n osobników, jest równe:

$$P = \frac{1}{n} \tag{3.2}$$

3.2. Modyfikacje reprodukcji

3.2.1. Metoda rankingowa

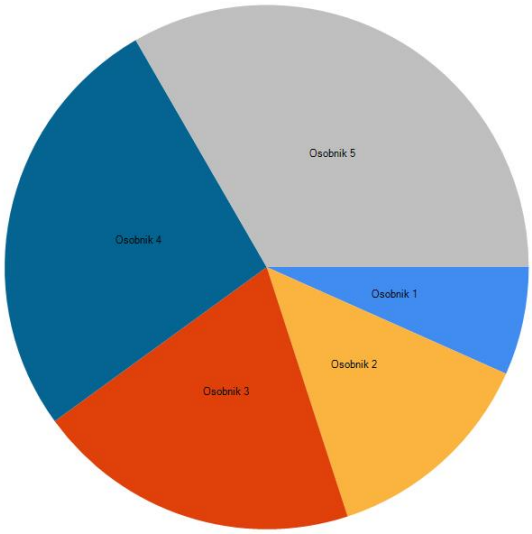
W metodzie selekcji rankingowej dla osobników składających się na populacje rodziców obliczana jest wartość funkcji celu, na której to podstawie tworzony jest ranking. Osobniki z najmniejszą wartością wyniku uzyskują najwyższe miejsca w rankingu [5]. Miejsce w rankingu jest argumentem funkcji, na podstawie której obliczane jest prawdopodobieństwo wyboru danego osobnika jako rodzica. Rodzic zostanie poddany kolejnym operacjom (mutacja, krzyżowanie).
Funkcja stosowana, na której to podstawie tworzony jest ranking, może być funkcją liniową. Wówczas osobniki zajmujące w rankingu będącym n elementowym wektorem konkretne miejsce określone rangą, gdzie n to ilość osobników w populacji otrzymują prawdopodobieństwo przejścia do grupy rodzicielskiej określone zależnością 3.3:

$$P = \frac{ranga}{\sum_{j=1}^{n} j} \tag{3.3}$$

Tabela 3.1. Wartości parametrów na kolejnych osobników w metodzie rankingowej.

Osobnik	Funkcja celu	Ranga	Prawdopodobieństwo
Osobnik 1	373	1	6,(6) %
Osobnik 2	254	2	13,(3) %
Osobnik 3	202	3	20 %
Osobnik 4	136	4	26,(6) %
Osobnik 5	10	5	33,(3) %

Na podstawie 3.1 zauważalny staje się fakt, iż rozkład prawdopodobieństwa nie jest bezpośrednio zależny od wartości funkcji celu. W metodzie tej zwiększone są więc szanse przejścia do grupy rodzicielskiej osobników z małą wartością funkcji dopasowania, które to miałyby małe szanse przejścia dalej w metodzie selekcji zależnej od wartości funkcji celu. Metoda może wydawać się niekorzystna z punktu widzenia osobników z dużą wartością funkcji dopasowania, jako że mogą otrzymać one niewiele większe prawdopodobieństwo przejścia do grupy rodzicielskiej od osobników znajdujących się rangę niżej, pomimo iż ich wartości funkcji celu mogą różnić się znacząco.



Rys. 3.1. Rozkład prawdopodobieństwa w metodzie rankingowej.

3.2.2. Metoda ruletki

W metodzie ruletki każdemu chromosomowi zostaje przypisany wycinek koła [5]. Im silniejszy osobnik, tym większy wycinek koła dostaje, gdyż zależność wartości funkcji przystosowania i prawdopodobieństwo wylosowania osobnika jest wprost proporcjonalna. Każdy osobnik tworzący populacje poddawany jest ocenie funkcji celu. Maksymalna wartość funkcji przystosowania oznacza najmniejszą wartość funkcji celu. Uzyskana wartość funkcji przystosowania w stosunku do sum wartości tej funkcji wszystkich osobników populacji wyznacza prawdopodobieństwo wylosowania danego osobnika na rodzica. Prawdopodobieństwo wylosowania danego osobnika dane jest następującą zależnością:

$$P_i = \frac{f(i)}{\sum_{j=1}^{n} f(j)} \tag{3.4}$$

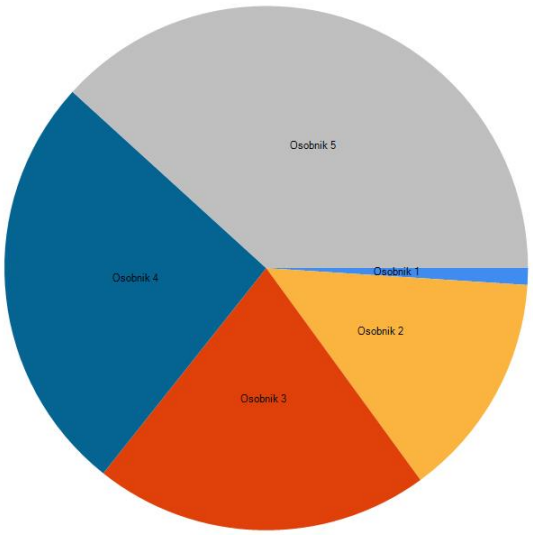
Gdzie f(x) - funkcja przystosowania

Funkcją przystosowania w rozważanym poniżej przypadku jest funkcja przyporządkowującą największej wartości funkcji celu wartość najmniejszą, a wartości najmniejszej wartość największą.

W metodzie ruletki wkład każdego osobnika w rozkład prawdopodobieństwa jest wprost proporcjonalny do wartości jego funkcji dopasowania. Wady tej metody stają się widoczne, w sytuacji, gdy dany osobnik zajmuje większość powierzchni koła, gdyż osobniki o małej wartości funkcji dopasowania są przedwcześnie eliminowane, a więc nie mają wielkiej szansy, by zostać wylosowane. Prowadzi to do braku różnorodności w kolejnych grupach rozrodczych, co utrudnia dojście do lepszego wyniku. Z problemem tym dobrze sobie radzi wspomniana wyżej metoda rankingowa, gdzie eliminowana jest możliwość dużej przewagi jednego z nich nad resztą.

Tabela 3.2. Wartości parametrów na kolejnych osobników w metodzie ruletki.

Osobnik	Funkcja celu	Funkcja dopasowania	Prawdopodobieństwo
Osobnik 1	373	10	1,02 %
Osobnik 2	254	136	13,95 %
Osobnik 3	202	202	20,72 %
Osobnik 4	136	254	26,05 %
Osobnik 5	10	373	38,25 %



Rys. 3.2. Rozkład prawdopodobieństwa w metodzie ruletki.

3.2.3. Metoda turniejowa

Metoda selekcji turniejowej polega na rozgrywaniu turnieju pomiędzy osobnikami tworzącymi populacje. W wyniku jednorazowej rozgrywki wygrywa osobnik o większej wartości funkcji przystosowania, przechodząc tym samym do kolejnej rundy. W każdym turnieju zwycięża jeden osobnik, którego to genotyp jest wykorzystywany przy kolejnych operacjach genetycznych. Populacja dzielona jest na podgrupy złożone z dwóch osobników, w których to dokonywane są rozgrywki. W przypadku nieparzystego rozmiaru populacji osobnik ostatni przechodzi do kolejnej rundy bezkonkurencyjnie.

Metoda turniejowa nie wymaga znajomości optymalizowanej funkcji, konieczny jest jedynie informacja o relacji zachodzącej pomiędzy dwoma kolejnymi osobnikami.

3.2.4. Metoda elitarna

W metodzie tej nacisk położony jest na to, aby do grupy rozrodczej przedostały się z prawdopodobieństwem równym p=100% osobniki najlepiej przystosowane. W poprzednich metodach (ruletki, turniejowa, rankingowa) prawdopodobieństwo przejścia do grupy rozrodczej dla osobników z większym wskaźnikiem funkcji przystosowania jest jedynie większe niż dla pozostałych osobników. Nie ma jednak gwarancji przedostania się dalej osobników najlepiej przystosowanych. W strategii elitarnej do grupy, na której zostanie dokonana operacja mutacji, a następnie krzyżowania przedostaje się n osobników z populacji odznaczających się najwyższym wskaźnikiem funkcji przystosowania [1].

Zastosowanie elitaryzmu jako metody selekcji daje pewność, iż do grupy rodzicielskiej przedostaną się najlepiej do tego przystosowane osobniki. Dzięki temu unika się sytuacji, w której stosunkowo mocny osobnik w wyniku operacji genetycznych zostanie osłabiony, a co za tym idzie, nie przechodzi on do grupy rodzicielskiej.

4. Operacja mutacji

Mutacja stanowi najważniejszą część algorytmu ewolucyjnego, z tego względu stosuje się wiele modyfikacji w celu rozwoju i poprawy działania tego operatora. Mutacja stosowana w algorytmie ewolucji różnicowej zdefiniowana jest w odmienny sposób w stosunku do algorytmu ewolucyjnego, w którym to genotyp stanowią wartości binarne, a operator mutacji ma za zadanie wprowadzenie zmian na zasadzie negacji obecnej wartości genu. W mutacji różnicowej zastosowana jest operacja odejmowania od siebie dwóch wektorów, tworząc tym samym wektor różnicowy, czemu również algorytm zawdzięcza swoją nazwę z ang. *difference* [2]. Wektor różnicowy jest zatem miarą określającą odległość pomiędzy dwoma osobnikami. Podobnie jak ma to miejsce w innych algorytmach należących do grona algorytmów ewolucyjnych, tak i ewolucja różnicowa posiada wiele alternatywnych metod mutacji. Metody mutacji różnicowej różnią się od siebie głównie elementami takimi jak:

1. Sposób wyboru osobnika będącego wektorem bazowym, a więc wektorem znajdującym się w grupie rozrodczej, lecz nie wchodzącym w skład wektora różnicowego X,
2. Liczba wektorów różnicowych Y,

W związku z powyższym, metody mutacji mogą przyjąć następującą nomenklaturę *DE/X/Y/Z*. Z oznacza rodzaj zastosowanego operatora krzyżowania w związku z czym parametr ten nie będzie uwzględniony w poniższym opisie metod mutacji . Wybór danej metody jest zależny od specyfiki rozwiązywanego problemu i dla różnych jego instancji może dawać odmienne rezultaty. Ogólny wzór opisujący sposób powstawania genu osobnika potomnego można wyrazić następująco:

$$\forall U_i = S_{r_1 i} + \sum_{j=1}^{n} F_j \cdot (S_{r_{2j} i} - S_{r_{3j} i}) \tag{4.1}$$

Gdzie:
j - liczba wektorów różnicowych,
U_i - i-ty gen osobnika potomnego,
r_1,r_2,r_3 - osobniki wchodzące w skład populacji,
$S_{r_1 i}$ - i-ty gen osobnika r_1
F - współczynnik mutacji, $F \in (0, 1)$,
S -populacja,

Wprowadzono zależność, iż osobnikami biorącymi udział w procesie mutacji są osobniki uprzednio wybrane poprzez metodę reprodukcji [12]. W implementacji metod mutacji dodatkowo należało zastosować pewnego rodzaju modyfikacje, wynikającą z faktu, iż pojedynczy osobnik jest ciągiem będącym permutacją, a więc nie dopuszcza istnienia wartości np. zmiennoprzecinkowych. Jako że wektor różnicowy mnożony jest przez współczynnik mutacji, będący wartością zmiennoprzecinkową, wektor końcowy posiada geny będące wartościami zmiennoprzecinkowymi. W celu eliminacji takiej sytuacji należało dokonać stosownego rodzaju skalowanie uwzględniające relacje mniejszości i większości pomiędzy poszczególnymi genami, a jednocześnie zamieniające wartości zmiennoprzecinkowe na całkowite. Pseudokod wspomnianej modyfikacji zamieszczono poniżej:

cnt = 1
while (cnt != $RozmiarOsobnika$ + 1) do
    1. $index \leftarrow$ indeks najmniejszego elementu w wektorze wynikowym mutacji
    2. W wektorze wynikowym mutacji, pod $index$, wstaw $MaxSizeOf$ ($double$)
    3. W wektorze wynikowym operacji, pod $index$, wstaw wartość cnt
    4. Zwiększ cnt
end while

Poniżej zestawione zostały strategie najczęściej spotykane w literaturze i opisane szczegółowo w [10]. Listingi poszczególnych metod mutacji można znaleźć w dodatku A.

4.1. Klasyczna mutacja: DE/rand/1

Podstawową strategią stosowaną jako operator mutacji jest strategia zakładająca uznanie za wektor bazowy pierwszego osobnika znajdującego się w grupie rozrodczej [12]. Jako że grupa rozrodcza składa się z trzech osobników, dwa pozostałe są odpowiedzialne za utworzenie wektora różnicowego, który jest następnie skalowany współczynnikiem $F = 0,8$. Przyjęta wartość jest najczęściej spotykaną w literaturze wartością czynnika skalującego. Analiza wpływu tego parametru na działanie metod mutacji zostanie przedstawiona w dalszej części pracy. Powyżej wspomniane zależności można opisać wzorem:

$$\forall U_i = S_{r_1 i} + F \cdot (S_{r_2 i} - S_{r_3 i}) \tag{4.2}$$

4.2. Modyfikacje metod mutacji

Klasyczna metoda mutacji pomimo częstego jej stosowania, nie zawsze daje oczekiwane rezultaty. W pewnych instancjach problemów efektywniejszym działaniem mogą wykazać się metody mutacji, do których zostały wprowadzone pewnego rodzaju modyfikacje. Poniżej zestawionych zostało kilka możliwych modyfikacji opisanych bardziej szczegółowo [12].

4.2.1. Strategia II: DE/best/1

W strategii II miejsce wektora bazowego zajmuje wektor odznaczający się największą wartością funkcji celu. W celu znalezienia najlepszego osobnika konieczne jest przeszukanie całej populacji, co

może okazać się niekorzystne w kontekście złożoności obliczeniowej i eksploatacji algorytmu. Wektor różnicowy jest różnicą dwóch losowo wybranych osobników przeskalowanych współczynnikiem mutacji. Zależność strategii DE/best/1 można opisać wzorem:

$$\forall U_i = S_{best,i} + F \cdot (S_{r_1 i} - S_{r_2 i}) \tag{4.3}$$

4.2.2. Strategia III: DE/rand/n_v

Strategię III od poprzednich metod wyróżnia fakt, iż nowy osobnik powstaje w oparciu o sumę wektorów różnicowych, a nie na podstawie jednego wektora różnicowego tak jak miało to miejsce w przypadku poprzednich metod. Istnieje więc dodatkowy parametr n_v określający ilość wektorów różnicowych. W sytuacji gdy $2 * n_v + 1$ jest większe od rozmiaru populacji n, do n_v zostaje przypisana odgórna wartość równa największej liczbie wektorów różnicowych, które można utworzyć przy danym rozmiarze populacji n.

$$n_v = (n-1)/2 \tag{4.4}$$

Wartość będąca sumą wektorów różnicowych jest następnie skalowana współczynnikiem mutacji F. Opisane powyżej zależności można zapisać w postaci wzoru:

$$\forall U_i = S_{r_1 i} + F \cdot \sum_{k=1}^{n_v} (S_{r_2 i} - S_{r_3 i}) \tag{4.5}$$

4.2.3. Strategia IV: DE/current to best/$n_v + 1$

Strategia ta zawiera w sobie pewne elementy zawarte we wcześniej opisanych metodach mutacji różnicowej. Na osobnika potomnego składa się suma n_v wektorów różnicowych, będących różnicą dwóch losowo wybranych osobników z populacji, pomnożonych razy współczynnik mutacji (strategia III). Sumuje się także wartość, przemnożonego przez współczynnik mutacji, odrębnego wektora różnicowego, w skład którego wchodzi najlepszy (strategia II) oraz losowy osobnik. Dodatkowo należy również uwzględnić wartość osobnika o indeksie odpowiadającym iteracji, w której znajduje się obecnie algorytm. Zależności te opisuje wzór 4.6:

$$\forall U_i = S_{r_{itert} i} + F \cdot (S_{r_{best} i} - S_{r_1 i}) + F \cdot \sum_{k=1}^{n_v} (S_{r_2 i} - S_{r_3 i}) \tag{4.6}$$

4.2.4. Mutacja z wprowadzeniem parametru $\diagdown$

W literaturze [14] można także znaleźć wersje metod mutacji, w których wprowadza się modyfikacje powyżej zaprezentowanych metod poprzez wprowadzenie czynnika skalującego $\diagdown$. Parametr ten

wprowadza w algorytmie dodatkową losowość, co może wpłynąć korzystnie na szybkość dochodzenia algorytmu do rozwiązania końcowego. Zmiany, w stosunku do poprzednich wersji, dotyczą pomnożenia wektora bazowego przez czynnik skalujący $\nwarrow$, gdzie $\nwarrow \in (0, 1)$. W poprzednich strategiach niezależnie od ich rodzajów wektor bazowy był zawsze wartością nieprzeskalowaną. Można to opisać poniższymi zależnościami:

1. Strategia I : $\forall U_i = \nwarrow \cdot S_{r_1 i} + F \cdot (S_{r_2 i} - S_{r_3 i})$
2. Strategia II : $\forall U_i = \nwarrow \cdot S_{best,i} + F \cdot (S_{r_1 i} - S_{r_2 i})$
3. Strategia III : $\forall U_i = \nwarrow \cdot S_{r_1 i} + F \cdot \sum_{k=1}^{n_v} (S_{r_2 i} - S_{r_3 i})$
4. Strategia IV : $\forall U_i = \nwarrow \cdot S_{r_{itert} i} + F \cdot (S_{r_{best} i} - S_{r_1 i}) + F \cdot \sum_{k=1}^{n_v} (S_{r_2 i} - S_{r_3 i})$

4.3. Dostrajanie parametru mutacji

W literaturze [2], [14], za wartość parametru mutacji F, który to odpowiada za przeskalowanie wektora różnicowego, najczęściej przyjmuje się wartość $F = 0{,}8$. Nie mniej jednak jest to wartość uzyskana doświadczalnie, co oznacza, iż daje ona najlepsze efekty w większości przypadków testowych. Należy jednak pamiętać o fakcie, iż przypadkami testowymi są zazwyczaj standardowe funkcje, a więc w sytuacji, gdy zagadnienia są bardziej złożone, wartość $F = 0{,}8$ może okazać się niezadowalająca. Stała wartość parametru F może być więc pewnego rodzaju ograniczeniem. Z tego względu wprowadzenie dynamicznej zmiany tego parametru może okazać się rozwiązaniem poprawiającym działanie algorytmu. Poniżej zaprezentowano dwie metody modyfikacji czynnika skalującego F.

4.3.1. Modyfikacja parametru F zgodnie z daną funkcją

Zgodnie ze sposobem opisanym w pracy [17], zmiana wartości współczynnika mutacji F w kolejnych iteracjach algorytmu sprowadza się do modyfikacji współczynnika zgodnie ze wzorem:

$$F_{t+1} = F_t + \frac{(0.5 + F_0)}{g} \tag{4.7}$$

Gdzie:

F_{t+1} - współczynnik skalującyw kolejnej iteracji,

F_0 - współczynnik skalujący w pierwszej iteracji,

g - liczba iteracji algorytmu,

Początkowo parametr F_0 przyjmuje wartość $F_0 = 0{,}3$. W kolejnych iteracjach algorytmu wielkość ta jest sukcesywnie zwiększana o wartość zależną od założonej początkowo liczby iteracji algorytmu.

4.3.2. Modyfikacja parametru F zgodnie z rozkładem normalnym

Istnieje również podejście, w którym rozkład prawdopodobieństwa wylosowania danej wartości parametru określony jest rozkładem normalnym. W podejściu tym zmiana czynnika skalującego

dokonywana jest w każdej iteracji algorytmu i nie jest ona zależna od poprzedniej iteracji, tak jak miało to miejsce w działaniach opisanych w podpunkcie 4.3.1.
Jednym ze sposobów opisu rozkładu normalnego jest funkcja gęstości prawdopodobieństwa dana wzorem:

$$f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{4.8}$$

Gdzie:
μ - wartość średnia,
σ - odchylenie standardowe,

Konieczne staje się zatem przypisanie pewnych wartości do parametrów μ oraz σ. Wielkość μ powinna być wartością współczynnika F, który wykazuje się, największą skutecznościa spośród wszystkich badanych. W sytuacji gdy nie jest możliwe wyodrębienie najlepszej wartości parametru, gdyż zależność wartości współczynnika oraz funkcji celu jest losowa, wspomniane w niniejszym podrozdziale zależności nie zdajdują zastosowania. Z kolei parametr σ powinien być ustalony w taki sposób by w przedziale (0,1) znajdowała się większość obszaru prawdopodobieństwa.

5. Operacja krzyżowania

Operacja krzyżowania jest drugą, obok mutacji, operacją należącą do grupy operatorów genetycznych. Głównym celem istnienia krzyżowania w algorytmie ewolucji różnicowej jest wprowadzenie dodatkowego zróżnicowania wśród osobników. Potomek będący wynikiem operacji krzyżowania o_i zgodnie z pseudokodem zmieszczonym w podrozdziale 2.1.3 powstaje poprzez wymianę kodu genetycznego osobnika wynikowego operatora mutacji u_i oraz osobnika x_i należącego do populacji rodziców. Operacji tej towarzyszy również parametr Cr określany mianem współczynnika krzyżowania. Wartości tego współczynnika może zostać przypisana stała wartość bądź też może ona ulegać zmianie wraz z biegiem algorytmu.

W algorytmach ewolucyjnych stosuje się między innymi metody krzyżowania takie jak krzyżowanie dwupunktowe, wielopunktowe czy też równomierne [12], natomiast w przypadku algorytmów ewolucji różnicowej spotykane warianty to krzyżowanie dwumianowe czy też wykładnicze (odpowiednik krzyżowania dwupunktowego w algorytmie ewolucyjnym). Wspomniane wyżej metody nie znajdują jednak bezpośredniego zastosowania w kontekście problemów optymalizacji, w których to poszczególne osobniki są permutacjami, jako że osobniki w metodach tych są ciągami bitowymi. Istnieje wówczas ryzyko duplikacji genu w osobniku wynikowym. W celu dostosowania algorytmu ewolucji różnicowej tak by rozwiązywał on problem NP-trudny, jakim jest kwadratowe zagadnienie przydziału (QAP) należy dokonać modyfikacji standardowej implementacji metod krzyżowania, w taki sposób by powtórzenia w genach nie występowały. Niemniej jednak wspomniane wyżej modyfikacje wprowadzają do algorytmu dodatkową losowość, co niekoniecznie może wpłynąć korzystnie na całość działania algorytmu.

W literaturze [15] można znaleźć metody krzyżowania dostosowane do problemów, w których to osobniki są reprezentowane przez permutacje, a nie poprzez wartości binarne jak ma to miejsce w klasycznym algorytmie ewolucyjnym. Do grona tych metod zaliczamy między innymi OX(Order Crossover), CX(Cycle Crossover) czy też PMX(Parcial Mapped Crossover). Implementacje i omówienie wspomnianych metod krzyżowania zostanie zaprezentowane w poniższych podrozdziałach. Listingi poszczególnych metod krzyżowania można znaleźć w dodatku A.

5.1. Klasyczne krzyżowanie : krzyżowanie dwumianowe

Zgodnie z [10], [12] w klasycznym algorytmie ewolucji różnicowej najczęściej wykorzystywaną metodą odpowiedzialną za operacje krzyżowania jest krzyżowanie dwumianowe (ang. *binomial*). Polega ono na przypisaniu do genu osobnika potomnego, wartości genu osobnika zmutowanego bądź też osobnika należącego do populacji rodzicielskiej. O wyborze, z którego osobnika ma pochodzić gen wynikowy, decyduje zależność wartości wylosowanej liczby zmiennoprzecinkowej z zakresu (0,1) i parametru krzyżowania C_r. Opisane wyżej zależności można przedstawić w formie zapisu.

$$o_i = \begin{cases} u_i & \text{,gdy } rand(0,1) \leq C_r \\ x_i & \text{,gdy } rand(0,1) > C_r \end{cases} \tag{5.1}$$

gdzie $i = 1, ..., n$

W celu wykorzystania metody krzyżowania w taki sposób, by wektor wynikowy był permutacją danego zbioru, należy wprowadzić pewne modyfikacje w algorytmie działania tej metody. Modyfikacje te dotyczą sytuacji w której do osobnika wynikowego ma zostać wpisany gen, pochodzący od osobnika zmutowanego czy też od rodzica, który występuje już w osobniku wynikowym. Wówczas z pozostałych, niewykorzystanych jeszcze genów osobnika zmutowanego lub rodzica, zostaje losowany kolejny gen. Czynność ta jest powtarzana aż do momentu wylosowania genu, który nie zawiera się w osobniku wynikowym.

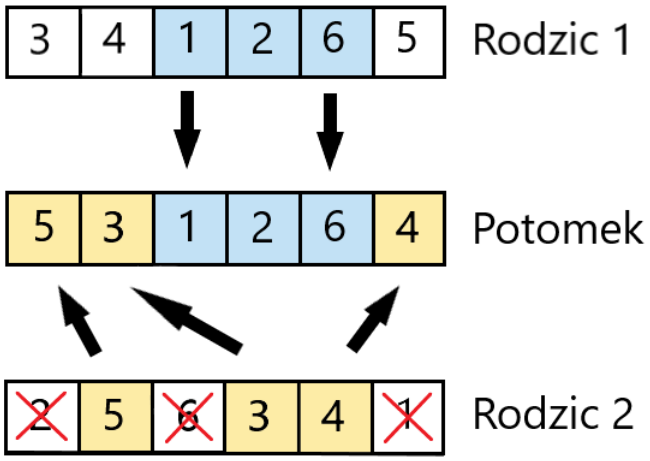
5.2. Pozostałe metody krzyżowania

W literaturze [16] istnieją metody krzyżowania mające bezpośrednie zastosowanie w przypadku osobników będących permutacjami pewnego zbioru. Poniżej zostały opisane zasady działania oraz implementacje poszczególnych metod krzyżowania.

5.2.1. Krzyżowanie OX

Strategia OX ang.*Order Crossover* zakłada utworzenie osobnika potomnego poprzez wybór podciągu w jednym z osobników będącym rodzicem [15]. Następnie podciąg ustawiany jest w potomku na miejscach o tych samych indeksach, na których znajdował się on w osobniku będącym rodzicem. Pozostałe luki w osobniku potomnym wypełnianie są wartościami genów drugiego z rodziców, przy jednoczesnym sprawdzeniu, czy osobnik potomny nie zawiera już danej wartości. Gen potomny jest dopełniany od lewej do prawej strony. Opisane zależności przedstawione zostały na poniższym schemacie.

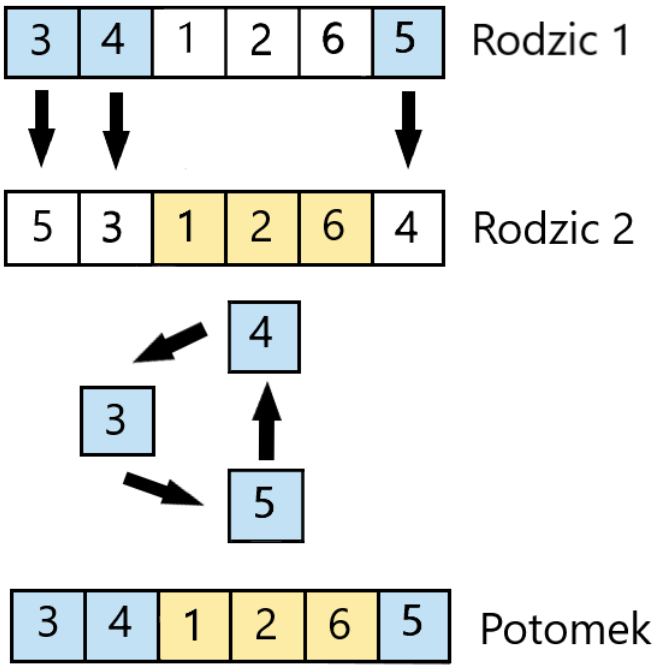
Długość podciągu oraz indeks jego początku również są losowane programowo.



Rys. 5.1. Schemat działania operatora krzyżowania OX.

5.2.2. Krzyżowanie CX

Operacja krzyżowania CX ang. *Cycle Crossover* opiera się na utworzeniu domkniętego obiegu zawierającego w sobie elementy każdego z rodziców [16]. Obieg rozpoczyna się od pierwszego elementu osobnika zmutowanego, kolejny fragment stanowi zaś element znajdujący się na tej samej pozycji, lecz w osobniku wywodzącym się z populacji rodzicielskiej. Następnie na podstawie wartości tego elementu odszukiwany jest odpowiadający mu gen w osobniku zmutowanym. Czynności te powtarzane są do chwili, gdy kandydatem na dołączenie do obiegu jest gen odpowiadający wartości pierwszego elementu w obiegu. Sytuacja ta oznacza zakończenie pętli.
Geny osobnika zmutowanego, odznaczające się wartościami zawartymi w obiegu, zostają wpisane na odpowiadających im pozycjach w genie wynikowym operacji. Dopełnienie luk zostaje realizowane poprzez inicjalizacje danej pozycji wartością pochodzącą od osobnika z populacji rodzicielskiej, w kolejności od lewej do prawej strony. Opisane wyżej zależności przedstawione zostały na poniższym schemacie.

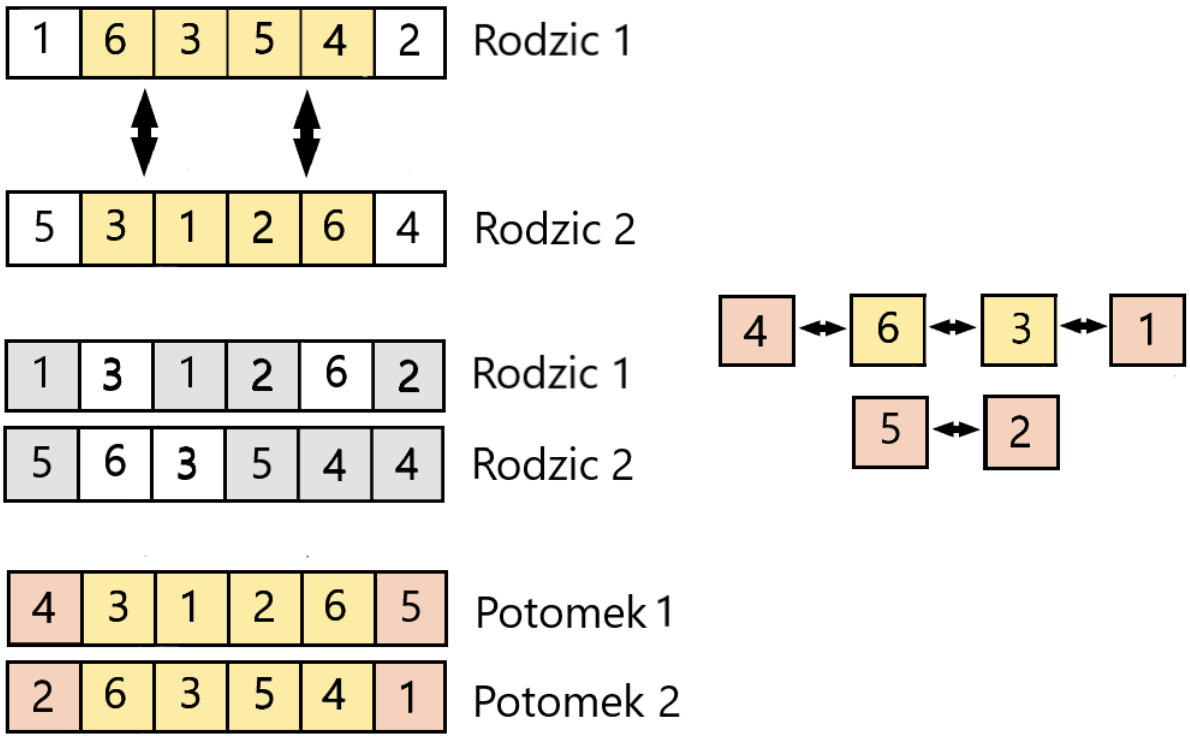


Rys. 5.2. Schemat działania operatora krzyżowania CX.

5.2.3. Krzyżowanie PMX

Metoda PMX ang. *Partial-Mapped Crossover* jest jedną z bardziej skomplikowanych strategii krzyżowania. Jej działanie, podobnie jak uprzednio opisanych metod, jest zależne od osobników będących rodzicami. Zarówno z pierwszego, jak i z drugiego rodzica wybierany jest podciąg rozpoczynający się w miejscu o losowym indeksie i posiadający losową długość. Wartości te są takie same dla obydwu rodziców. Następnie dokonuje się wymiany podciągów pomiędzy dwoma rodzicami, w wyniku czego w osobnikach mogą wystąpić powtórzenia. Rozwiązaniem tego konfliktu jest stworzenie relacji mapowania, rysunek. Geny tworzą miedzy sobą relacje, wtedy gdy jeden z nich stanowi początek, a drugi koniec zamkniętego obiegu. Obieg tworzony jest na tej samej zasadzie co odpowiadający mu i opisany w podrozdziale 5.2.2, z uwzględnieniem faktu, iż zbiorem danych jest wylosowany podciąg. Opisane zależności przedstawione są na poniższym schemacie.

Metodę krzyżowania PMX od poprzednich metod odróżnia fakt, iż w wyniku jej działania uzyskujemy dwa osobniki potomne.



Rys. 5.3. Schemat działania operatora krzyżowania PMX.

5.3. Dostrajanie parametru krzyżowania

Dostrajanie parametru krzyżowania w niniejszej pracy zastosowanie znajduje jedynie w kontekście metody krzyżowania dwumianowego, gdyż tylko w tej metodzie stosowany jest współczynnik C_r. Zmiana wartości tego parametru dokonywana jest na podobnych zasadach co parametru mutacji, opisanego w podrozdziale 4.3, z uwzględnieniem jednak innej postaci funkcji, na podstawie której dokonywane jest dostrajanie [10]. Szczegółowy opis metod modyfikacji znajduje się poniżej.

5.3.1. Modyfikacja parametru C_r zgodnie z daną funkcją

W oparciu o prace [17] dostrajanie współczynnika krzyżowania powinno być dokonywane zgodnie z funkcją opisaną zależnością 5.2:

$$C_{r,t+1} = C_{r,t} - \frac{(C_{r,0} - 0.7)}{g} \tag{5.2}$$

Gdzie :

$C_{r,t+1}$ - współczynnik krzyżowania w kolejnej iteracji,

$C_{r,0}$ - współczynnik krzyżowania w pierwszej iteracji,

g - liczba iteracji algorytmu,

Początkowo parametrowi $C_{r,0}$ zostaje przypisywana wartość $C_{r,0} = 0{,}3$. Następnie w miarę działania algorytmu i występowania nowych iteracji wartość ta ulega aktualizacji, tworząc tym samym cały proces dynamicznym.

5.3.2. Modyfikacja parametru C_r zgodnie z rozkładem normalnym

W podstawowej wersji algorytmu ewolucyjnego współczynnik C_r jest wartością losową z przedziału (0,1), tak samo jak współczynnik mutacji. W związku z powyższym metoda modyfikacji współczynnika krzyżowania zgodnie z rozkładem normalnym jest identyczna z parametrem F. Metoda ta została szczegółowo opisana w podrozdziale 4.3.

6. Testy

6.1. Dane wejściowe instancja I

Dane wejściowe zostały zaczerpnięte z biblioteki [9]. Znajdowały się tam zarówno macierze trasy, kosztu, jak i informacje o najmniejszej wartości funkcji celu, jaką udało się uzyskać dla danej instancji danych wejściowych. Dane rozważane w tym rozdziale są rozmiaru równego 12. Macierz testowa jest macierzą symetryczną względem przekątnej.

$$\textbf{Macierz\_trasy} = \begin{pmatrix} 0 & 1 & 2 & 2 & 3 & 4 & 4 & 5 & 3 & 5 & 6 & 7 \\ 1 & 0 & 1 & 1 & 2 & 3 & 3 & 4 & 2 & 4 & 5 & 6 \\ 2 & 1 & 0 & 2 & 1 & 2 & 2 & 3 & 1 & 3 & 4 & 5 \\ 2 & 1 & 2 & 0 & 1 & 2 & 2 & 3 & 3 & 3 & 4 & 5 \\ 3 & 2 & 1 & 1 & 0 & 1 & 1 & 2 & 2 & 2 & 3 & 4 \\ 4 & 3 & 2 & 2 & 1 & 0 & 2 & 3 & 3 & 1 & 2 & 3 \\ 4 & 3 & 2 & 2 & 1 & 2 & 0 & 1 & 3 & 1 & 2 & 3 \\ 5 & 4 & 3 & 3 & 2 & 3 & 1 & 0 & 4 & 2 & 1 & 2 \\ 3 & 2 & 1 & 3 & 2 & 3 & 3 & 4 & 0 & 4 & 5 & 6 \\ 5 & 4 & 3 & 3 & 2 & 1 & 1 & 2 & 4 & 0 & 1 & 2 \\ 6 & 5 & 4 & 4 & 3 & 2 & 2 & 1 & 5 & 1 & 0 & 1 \\ 7 & 6 & 5 & 5 & 4 & 3 & 3 & 2 & 6 & 2 & 1 & 0 \end{pmatrix}$$

$$\textbf{Macierz\_kosztu} = \begin{pmatrix} 0 & 3 & 4 & 6 & 8 & 5 & 6 & 6 & 5 & 1 & 4 & 6 \\ 3 & 0 & 6 & 3 & 7 & 9 & 9 & 2 & 2 & 7 & 4 & 7 \\ 4 & 6 & 0 & 2 & 6 & 4 & 4 & 4 & 2 & 6 & 3 & 6 \\ 6 & 3 & 2 & 0 & 5 & 5 & 3 & 3 & 9 & 4 & 3 & 6 \\ 8 & 7 & 6 & 5 & 0 & 4 & 3 & 4 & 5 & 7 & 6 & 7 \\ 5 & 9 & 4 & 5 & 4 & 0 & 8 & 5 & 5 & 5 & 7 & 5 \\ 6 & 9 & 4 & 3 & 3 & 8 & 0 & 6 & 8 & 4 & 6 & 7 \\ 6 & 2 & 4 & 3 & 4 & 5 & 6 & 0 & 1 & 5 & 5 & 3 \\ 5 & 2 & 2 & 9 & 5 & 5 & 8 & 1 & 0 & 4 & 5 & 2 \\ 1 & 7 & 6 & 4 & 7 & 5 & 4 & 5 & 4 & 0 & 7 & 7 \\ 4 & 4 & 3 & 3 & 6 & 7 & 6 & 5 & 5 & 7 & 0 & 9 \\ 6 & 7 & 6 & 6 & 7 & 5 & 7 & 3 & 2 & 7 & 9 & 0 \end{pmatrix}$$

6.1.1. Metody reprodukcji

Analiza metod reprodukcji w obrębie jednej instancji została przeprowadzona przy założeniu stałych takich jak:

- macierz odległości,
- macierz przepływu,
- metoda reprodukcji,
- metoda krzyżowania,
- wielkość populacji

Testy zostały przeprowadzone dla instancji danych wejściowych, zaczerpniętych z [9], gdzie również znajduje się informacja o najlepszym otrzymanym wyniku funkcji celu dla danej instancji danych, wynosił on 1652. Dzięki temu możliwe jest określenie jak blisko rozwiązania znajduje się wynik działania algorytmu dla każdej z metod. W celu uzyskania ostatecznego rozwiązania, a więc rozwiązania niezmieniającego się dalej pod wpływem kolejnych iteracji algorytmu, analiza została przeprowadzona dla odpowiednio 20 000 oraz 50 000 iteracji.

6.1.1.1. 20 000 iteracji

W 6.1 zostały zestawione wartości błędu względnego funkcji celu w stosunku do wartości, która zgodnie z [9] jest najmniejszą znaną wartością dla danej instancji danych. Testy zostały przeprowadzone dwudziestokrotnie dla poszczególnych metod reprodukcji. Metody te zostały szczegółowo opisane w rozdziale 3. W metodach takich jak metoda Losowa 2, Rankingowa 2 oraz Turniejowa 2 nie istnieje warunek co do niepowtarzalności się osobników w obrębie jednej grupy rozrodczej. Dla każdej z metod został policzony średni błąd względny wartości funkcji celu, a także odchylenie

standardowe. Za pomocą tych narzędzi statystycznych można określić zachowanie poszczególnych metod oraz wyodrębnić metodę dającą w tym kontekście najbardziej satysfakcjonujący wynik.

Tabela 6.1. Wartości błędu względnego funkcji celu dla poszczególnych metod reprodukcji, 20 000 iteracji.

Iteracja	Losowa	Rankingowa	Ruletka	Turniejowa	Elitarna	Losowa 2	Rankingowa 2	Turniejowa 2
1	1,45 %	0,60%	2,9%	0,84 %	2,05%	2,17%	0,96%	6,53%
2	1,69%	0,60%	0,72%	0,84%	1,45%	1,81%	1,81%	6,29%
3	0,96%	0,72%	3,51%	1,08%	1,69%	1,81%	1,21%	8,71%
4	0,84%	0,72%	2,54%	2,66%	1,81%	1,69%	3,02%	5,44%
5	0,84%	1,21%	2,42%	3,26%	2,42%	1,21%	1,08%	10,16%
6	1,08%	0,48%	3,26%	1,69%	1,08%	1,81%	1,08%	8,83%
7	0,84%	0,84%	3,38%	2,3%	1,08%	1,69%	3,02%	6,9%
8	1,08%	0,24%	4,23%	1,45%	0,96%	2,17%	2,05%	2,9%
9	2,42%	084%	0,60%	0,24%	1,69%	3,38%	1,57%	3,26%
10	0,96%	0,24%	1,08%	1,45%	1,33%	3,87%	3,26%	10,53%
11	1,81%	0,48%	0,96%	1,08%	2,3%	3,63%	1,81%	7,62%
12	2,66%	0,48%	3,87%	2,17%	1,57%	4,6%	0,48%	7,14%
13	0,96%	0,12%	2,42%	1,69%	1,93%	3,02%	1,21%	7,38%
14	0,96%	0,48%	0,96%	1,21%	3,26%	2,66%	2,78%	4,6%
15	2,66%	0,72%	1,21%	2,17%	1,93%	1,93%	1,93%	9,32%
16	3,75%	0,48%	2,05%	1,45%	1,33%	1,45%	2,3%	7,38%
17	0,60%	0,24%	2,9%	1,33%	1,45%	1,93%	1,93%	6,41%
18	1,57%	0,60%	1,21%	1,08%	2,3%	2,42%	1,69%	5.69%
19	1,09%	0,84%	3,99%	1,21%	1,69%	3,63%	0,60%	6,65%
20	1,93%	0,48%	1,81%	0,84%	1,33%	0,84%	1,08%	4,11%
ŚREDNIA	1,51%	0,58%	2,31%	1,51%	1,74%	2,39%	1,75%	6,8%
ODCHYLENIE STANDARDOWE	0,8%	0,25%	1,15%	0,7%	0,54%	0,97%	0,79%	6,8%

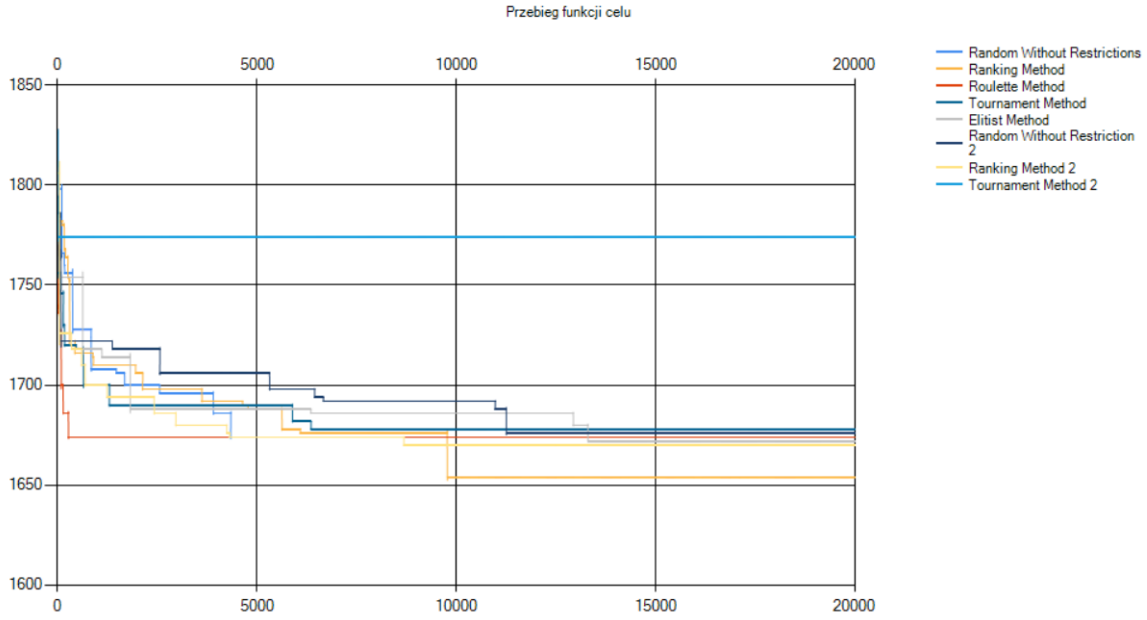
Na podstawie powyższej tabeli zauważalny staje się fakt, iż najbardziej satysfakcjonujące wyniki w przeciągu 20 000 iteracji otrzymuje metoda rankingowa. W 17 na 20 przypadków uzyskała ona najmniejsze wartości funkcji celu. W pozostałych 3 przypadkach znajduje się na drugim bądź też trzecim miejscu. Posiada ona również najmniejszy współczynnik odchylenia standardowego oraz współczynnik zmienności co świadczy o stosunkowo najbliższym spośród wszystkich metod położeniu wyników wokół wartości średniej. Poniżej zestawiony został ranking metod selekcji utworzony w zależności od uzyskanej wartości średniej funkcji celu.

Widoczny staje się więc fakt, iż lepiej działają metody, w których stosujemy ograniczenia co do konieczności niepowtarzania się danego rodzica w grupie rodzicielskiej. A więc metody Rankingowa, Turniejowa, Losowa oraz Elitarna. W gronie metod niestosujących się do zasady indywidualności osobnika najwyższe miejsce w rankingu otrzymuje również metoda Rankingowa. Przebieg działania algorytmu dla poszczególnych metod można przenalizować na podstawie wykresu 6.1. Na wykresie tym został zaprezentowany przebieg, dla którego to wartość końcowa uzyskana przez metodę rankingową jest jednocześnie najmniejszą wartością funkcji celu otrzymaną w we wszystkich 20 iteracjach.

Tabela 6.2. Ranking metod reprodukcji na podstawie średniej wartości błędu względnego funkcji celu, 20 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	Rankingowa	0,58%	0
2	Turniejowa	1,51 %	0
3	Losowa	1,51 %	0
4	Elitarna	1,74 %	0
5	Rankingowa 2	1,75%	0
6	Ruletki	2,31%	0
7	Losowa 2	2,39%	0
8	Turniejowa 2	6,8%	0

Na podstawie wykresu 6.1 zauważalne staje się zmniejszanie wartości funkcji celu w kolejnych iteracjach algorytmu. W przypadku metod takich jak metoda ruletki czy też metoda turniejowa, która dopuszcza powtórzeń wśród osobników w grupie rodzicielskiej, stabilizacja poziomu następuje w początkowych iteracjach i utrzymuje się tak do końca. Powodem tak szybkiej zbieżności tych metod jest brak różnorodności w kolejnych grupach rodzicielskich. W metodzie ruletki spowodowane jest to dużym prawdopodobieństwem wylosowania osobnika najlepszego, co prowadzi do jego dominacji nad innymi. Natomiast w metodzie turniejowej turniej jest w większości przypadków wygrywany przez tego samego osobnika, co przy dodatkowym założeniu, iż w jednej grupie rodzicielskiej mogą znajdować się te same osobniki prowadzi do sytuacji w której na grupę rodzicielską składają się 3 te same osobniki. Pomimo wrażenia, iż rozwiązania osiągnęły stan stabilny już w 20 000 iteracji, należy sprawdzić, czy nie nastąpi jeszcze spadek wartości funkcji celu, gdy liczba iteracji zostanie zwiększona. W tym celu ponowione zostają badania metod dla 50 000 iteracji.



Rys. 6.1. Przebieg działania algorytmu dla poszczególnych metod reprodukcji, 20 000 iteracji.

6.1.1.2. 50 000 iteracji

W tabeli 6.3 zostały zestawione wartości błędu względnego funkcji celu w kolejnych iteracjach dla poszczególnych metod. W metodach takich jak metoda Losowa 2, Rankingowa 2 oraz Turniejowa 2 nie istnieje warunek co do niepowtarzalności się osobników w obrębie jednej grupy rozrodczej. W tym przypadku zwiększona została liczba iteracji działania algorytmu dla każdej z metod z 20 000 na 50 000. Kolorem zielonym zostały zaznaczone wyniki działania algorytmu osiągające wartość błędu względnego równą 0%.

W przypadku 50 000 iteracji zauważalne jest uzyskiwanie lepszych średnich wyników przez metody takie jak: metoda rankingowa, metoda elitarna oraz turniejowa dopuszczające istnienie powtórzeń w grupie rodzicielskiej. Dzieje się tak, ponieważ metody te nie kończą stabilizacji na poziomie 20 000, a spadek wartości funkcji celu następuje również w dalszych iteracjach. Pozostałe metody utrzymały średnią na tym samym bądź też nieco wyższym poziomie w stosunku to próby testowej zakładającej 20 000 iteracji. Poniżej w tabeli 6.4 zestawiony został ranking metod uporządkowany od metody działającej najlepiej, na pierwszym miejscu, do metody działającej najmniej korzystnie, na ostatnim.

Na podstawie tabeli 6.4 można wysunąć wniosek, iż po raz kolejny najlepsze wyniki uzyskuje metoda rankingowa, a zaraz po niej powtórnie najkorzystniej zachowuje się metoda turniejowa. Niemniej jednak przodowanie tych metod może mieć miejsce w przypadku założonej instancji danych wejściowych, dlatego należy przeprowadzić testy przy użyciu innej macierzy odległości oraz przepływu. Przebieg funkcji celu we wszystkich 50 000 iteracjach widoczny jest na wykresie 6.2, na którym to zauważyć można

Tabela 6.3. Wartości błędu względnego funkcji celu dla poszczególnych metod reprodukcji, 50 000 iteracji.

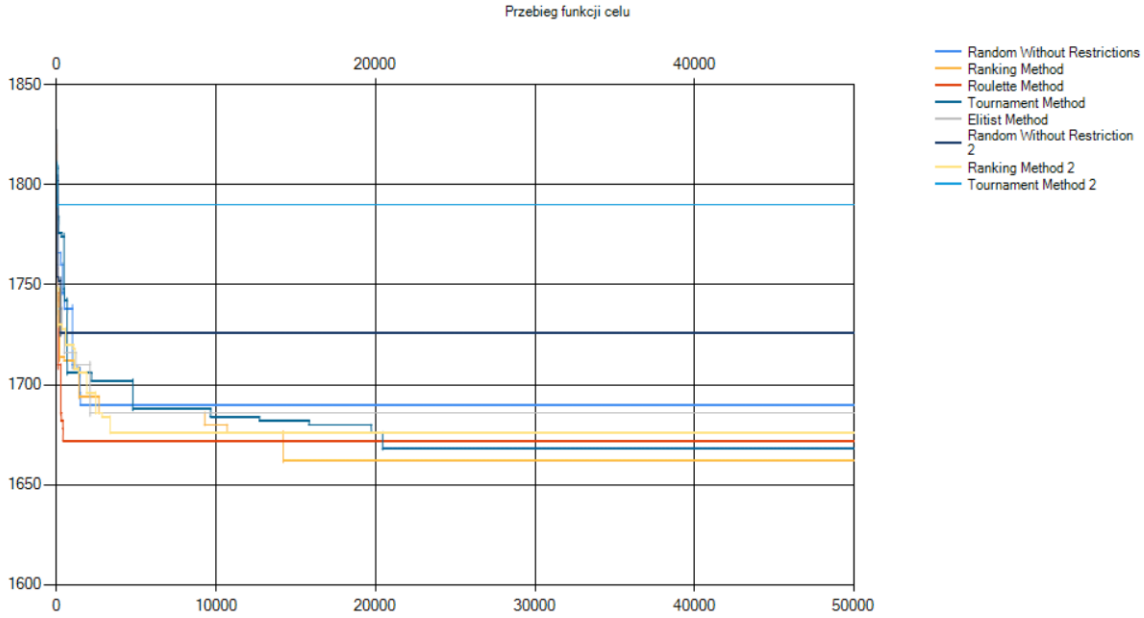
Iteracja	Losowa	Rankingowa	Ruletka	Turniejowa	Elitarna	Losowa 2	Rankingowa 2	Turniejowa 2
1	2,3 %	0,6%	4,47%	0,24%	1,21%	2,66%	1,93%	4,6%
2	1,93%	0,6%	2,05%	0,6%	1,33%	1,45%	2,42%	4,47%
3	0,24%	1,81%	1,93%	1,33%	1,93%	1,21%	1,21%	6,29%
4	3,99%	0,24%	1,81%	1,21%	1,45%	2,42%	0,48%	7,99%
5	2,3%	0,6%	1,21%	0,96%	2,05%	4,47%	1,45%	8,35%
6	1,69%	0,48%	4,47%	2,3%	1,45%	2,54%	1,21%	9,2%
7	1,33%	0,24%	1,93%	0,6%	1,45%	1,21%	1,08%	7,26%
8	2,42%	0,84%	1,81%	0,96%	0,72%	1,39%	2,05%	6,77%
9	0,84%	**0%**	0,6%	0,72%	1,08%	2,05%	2,42%	8,83%
10	2,54%	0,48%	2,78%	0,72%	2,78%	1,93%	2,66%	5,69%
11	2,54%	0,24%	1,69%	1,33%	1,45%	3,14%	3,14%	3,87%
12	0,6 %	0,48%	2,05%	0,48%	0,6%	2,66%	0,96%	10,53%
13	1,33%	0,84%	3,14%	1,08%	1,21%	2,66%	1,45%	8,59%
14	0,84%	0,24%	2,3%	1,45%	1,21%	4,47%	1,21%	5,69%
15	2,54%	1,45%	1,33%	0,6%	2,3%	3,63%	1,81%	6,29%
16	1,08%	0,6%	2,66%	0,96%	1,81%	1,45%	1,08%	10,53%
17	1,81%	**0%**	1,33%	0,72%	1,21%	3,99%	0,48%	9,2%
18	1,57%	0,48%	3,26%	1,93%	1,93%	2,17%	1,45%	7,14%
19	1,08%	0,48%	1,57%	0,84%	2,3%	3,99%	0,96%	6,53%
20	1,93%	0,6%	3,14%	2,42%	2,05%	1,45%	1,93%	10,53%
ŚREDNIA	1,75%	0,57%	2,4%	1,08%	1,47%	2,55%	1,57%	7,42%
ODCHYLENIE STANDARDOWE	0,85%	0,42%	0,93%	0,57%	0,51%	1,06%	0,7%	1,99%

Tabela 6.4. Ranking metod reprodukcji na podstawie średniej wartości błędu względnego funkcji celu, 50 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	Rankingowa	0,57%	2
2	Turniejowa	1,08%	0
3	Elitarna	1,47%	0
4	Rankingowa 2	1,57%	0
5	Losowa	1,75%	0
6	Ruletki	2,4%	0
7	Losowa 2	2,55%	0
8	Turniejowa 2	7,42%	0

również spadek dla metody turniejowej, w której istnieje zastrzeżenie co do niepowtarzalności się osobników w grupie rozrodczej. Wyjątkowo niekorzystnym działaniem odznacza się metoda Turniejowa 2, gdyż spadek następuje jedynie w początkowych iteracjach i na tym poziomie utrzymuje się już do końca.

Możliwe jest zatem ustalenie optymalnej ilości iteracji na liczbę 30 000, wówczas wartości funkcji celu w poszczególnych metodach osiągną już stan stabilny, a dodatkowe obliczenia dla liczby równej aż 50 000 iteracji nie będą musiały być wykonywane.



Rys. 6.2. Przebieg działania algorytmu dla poszczególnych metod reprodukcji, 50 000 iteracji.

6.1.2. Metody mutacji

Metody mutacji o których mowa w podrozdziale zostały szczegółowo opisane w rozdziale 4. Testy metod mutacji w obrębie jednej instancji danych zostały przeprowadzone przy uprzednim określeniu:

- macierzy odległości,
- macierzy przepływu,
- metod reprodukcji,
- metoda krzyżowania,
- wielkość populacji,

W klasycznych metodach mutacji stosowany jest wybór osobników do grupy rozrodczej poprzez metodę losową. Niemniej jednak ze względu na wyjątkowo dobre wyniki uzyskiwane przez metodę rankingową w testach przeprowadzanych w rozdziale 6.1.1, postanowiono sprawdzić jej działanie w połączeniu z różnymi wariantami metod mutacji. Liczba iteracji algorytmu była równa, podobnie jak w powyższych testach, 20 000 iteracji. Następnie, w celu zbadania czy dalej niż granica 20 000 następuje spadek wartości funkcji celu, również dla 50 000 iteracji. Zgodnie z podrozdziałem 4.2.4 metody mutacji zostaną dodatkowo przetestowane pod kątem wpływu na ich działanie parametru $\diagdown$. Również dynamiczna zmiana parametru mutacji poddawana jest testom.

6.1.2.1. Reprodukcja losowa

W rozdziale tym testy metod mutacji zostały przeprowadzone z użyciem metody reprodukcji wykorzystywanej w klasycznej wersji algorytmu ewolucji różnicowej. Metodą decydującą o osobnikach

wchodzących w skład grupy rozrodczej jest więc metoda losowa.

– **20 000 iteracji**

Wyniki zostały zaprezentowane w formie tabeli zawierającej wartości błędu względnego funkcji celu. Testy przeprowadzono dla 20 powtórzeń algorytmu. W wyniku tego działania możliwe było policzenie wartości średniej oraz odchylenia standardowego z kolejnych 20 prób.

Tabela 6.5. Wartości błędu względnego funkcji celu dla poszczególnych metod mutacji, metoda losowa, 20 000 iteracji.

Iteracja	DE/rand/1	DE/rand/1+⤡	DE/best/1	DE/best/1+⤡	DE/rand/n_v	DE/rand/n_v+⤡	DE/current to best/n_v + 1	DE/current to best/n_v + 1+⤡
1	1,21%	0%	3,39%	1,94%	0,85%	0,48%	0,73%	0,61%
2	0,97%	0,61%	4,84%	0,85%	1,21%	1,33%	0,61%	0,73%
3	2,91%	0,61%	2,66%	1,21%	1,33%	0,85%	0,12%	0,12%
4	2,18%	1,33%	1,33%	1,45%	1,82%	0,61%	0,61%	1,09%
5	2,06%	2,42%	2,42%	3,51%	1,57%	1,69%	1,09%	0,24%
6	1,57%	1,94%	2,54%	1,82%	0,85%	0,24%	0,85%	0,24%
7	3,15%	1,69%	3,75%	0,85%	2,54%	1,09%	1,09%	0,48%
8	1,57%	1,21%	3,39%	0,85%	1,45%	1,69%	1,21%	0,48%
9	1,82%	4,84%	5,08%	1,82%	0,97%	0,48%	0,61%	0,61%
10	1,33%	0,61%	4,24%	1,69%	1,33%	2,18%	0,48%	0%
11	2,42%	0,48%	3,51%	0,73%	1,21%	0,24%	0,48%	0,24%
12	3,15%	1,69%	1,82%	1,69%	2,18%	1,82%	0,48%	0,61%
13	1,21%	0,48%	1,33%	0,85%	4,60%	0,48%	0,48%	0,48%
14	1,21%	0%	3,03%	1,21%	1,33%	0,85%	0,97%	0,24%
15	1,21%	0,48%	1,45%	0,97%	1,45%	0,48%	1,21%	0,61%
16	0,61%	1,33%	2,30%	1,21%	0,48%	2,91%	0,85%	0,12%
17	1,94%	0,48%	4,24%	3,39%	0,61%	0%	1,21	0%
18	3,15%	0,48%	2,42%	2,06%	2,18%	2,42%	0,48%	0,85%
19	4,24%	1,94%	6,17%	1,45%	0,85%	3,75%	0,61	0%
20	0,48%	0,24%	1,45%	0,73%	1,33%	2,78%	0,48%	0%
ŚREDNIA	1,92%	1,14%	3,07%	1,51%	1,51%	1,32%	0,73%	0,39%
ODCHYLENIE	0,96%	1,09%	1,33%	0,77%	0,88%	1,02%	0,30%	0,31%

Sumaryczna liczba osiągnięcia przez algorytm wartości błędu względnego równego 0% jest równa 7, z czego 4 są osiągane w wyniku zastosowania w algorytmie metody DE/current to best/n_v + 1.

Tabela 6.6. Ranking metod mutacji na podstawie średniej wartości błędu względnego funkcji celu, metoda losowa, 20 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	DE/current to best/n_v + 1+⤡	0,39%	4
2	DE/current to best/n_v + 1	0,73%	0
3	DE/rand/n_v+⤡	1,14%	1
4	DE/rand/1+⤡	1,32%	2
5	DE/rand/n_v	1,51%	0
6	DE/best/1+⤡1	1,51%	0
7	DE/rand/1	1,92%	0
8	DE/best/1	3,07%	0

Na podstawie tabeli 6.6 widoczny staje się fakt, iż najmniejszą średnią wartością błędu względnego, o wartości 0,39%, charakteryzuje się metoda DE/current to best/n_v + 1+$\diagup$. W zestawieniu z pozostałymi metodami mutacji przyjmuje ona około dwukrotnie lepsze rezultaty, jako, że pozostałe metody uzyskują wartości błędu względnego wahające się w zakresie 0,73% - 3,07%. Wartość odchylenia standardowego dla tej metody wynosi 0,31% co w porównaniu z pozostałymi testowanymi metodami mutacji jest wartością stosunkowo niewielką. Metoda DE/current to best/n_v + 1+$\diagup$ uzyskuje korzystniejsze rezultaty w porównaniu do jej odpowiednika bez modyfikatora DE/current to best/n_v + 1. W przypadku pozostałych metod, DE/rand/n_v, DE/rand/1 oraz DE/best/1, modyfikacja współczynnikiem $\diagup$ również pozwala na uzyskiwanie przez te metody bardziej zadowalających efektów.

– **50 000 iteracji**

Poniższe testy w stosunku do poprzednich zestawionych w tabelach 6.5 oraz 6.6 różnią się od siebie liczbą iteracji algorytmu, zwiększoną z 20 000 do 50 000. Działanie to wprowadzone jest w celu zbadania czy nie następuje spadek wartości funkcji celu w iteracjach dalszych niż 20 000. Dodatkowym atutem jest możliwość konfrontacji wyników, a więc sprawdzenia, czy uzyskane w poprzednich testach wartości utrzymują się na tym samy bądź też lepszym poziomie.

Tabela 6.7. Wartości błędu względnego funkcji celu dla poszczególnych metod mutacji, metoda losowa, 50 000 iteracji.

Iteracja	DE/rand/1	DE/rand/1+$\diagup$	DE/best/1	DE/best/1+$\diagup$	DE/rand/n_v	DE/rand/n_v+$\diagup$	DE/current to best/n_v + 1	DE/current to best/n_v + 1+$\diagup$
1	1,09%	0,85%	2,06%	2,91%	0,85%	1,57%	0,12%	0%
2	1,82%	0,48%	2,91%	2,06%	0,73%	1,09%	0,48%	0%
3	1,57%	1,45%	1,82%	2,54%	1,57%	0,48%	0,12%	0,48%
4	0,97%	0,85%	2,54%	2,54%	1,45%	0,61%	0,24%	0%
5	2,91%	0,73%	2,18%	0,73%	0,85%	0,73%	0%	0,48%
6	1,09%	1,82%	2,54%	0,48%	1,45%	1,69%	0,61%	0,97%
7	0,73%	2,06%	3,27%	1,94%	1,21%	2,42%	0,48%	0%
8	2,66%	8,23%	2,06%	2,30%	2,06%	0,61%	0,48%	0,12%
9	1,57%	1,82%	1,94%	1,33%	0,61%	2,54%	0,61%	0%
10	1,94%	2,42%	1,21%	1,09%	1,45%	0,97%	0,12%	0%
11	2,66%	1,82%	2,06%	2,30%	0,85%	0,24%	0,61%	0,24%
12	0%	3,51%	4,36%	2,54%	2,30%	0%	0,12%	0,12%
13	0,48%	0,12%	1,82%	2,18%	1,45%	0%	0,48%	0%
14	0,97%	0,12%	1,45%	2,42%	1,94%	1,94%	0,24%	0,12%
15	0,85%	0,61%	2,78%	2,91%	1,33%	0,61%	0,24%	0,73%
16	1,09%	0,48%	5,08%	0,85%	0,61%	3,27%	0,48%	0,24%
17	2,06%	0%	2,42%	1,57%	0,85%	1,21%	0,48%	0,12%
18	0,48%	0%	2,30%	2,66%	0,61%	1,57%	0,48%	0,48%
19	5,57%	0%	2,78%	2,30%	6,90%	0%	0,48%	0,24%
20	1,94%	0,12%	3,27%	1,69%	0,61%	1,09%	0,24%	0%
ŚREDNIA	1,62%	1,37%	2,54%	1,97%	1,48%	1,13%	0,36%	0,22%
ODCHYLENIE	1,19%	1,83%	0,90%	0,72%	1,34%	0,88%	0,19%	0,27%

Liczba próbek, w których algorytm uzyskiwał wartość błędu względnego równą 0% wzrosła o 9 w stosunku do wersji algorytmu z liczbą iteracji równą 20 000. Świadczy to o zmniejszaniu

się wartość funkcji celu w iteracjach dalszych niż 20 000. W przypadku większości z metod następuje spadek średniej wartości błędu bezwzględnego funkcji celu. Niemniej jednak w przypadku metod takich jak DE/rand/1+$\diagdown$ oraz DE/best/1+$\diagdown$ następuje wzrost tej wartości. Zjawisko to spowodowane jest zadziałaniem czynnika losowego.

Tabela 6.8. Ranking metod mutacji na podstawie średniej wartości błędu względnego funkcji celu, metoda losowa, 50 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	DE/current to best/n_v + 1 +$\diagdown$	0,22%	8
2	DE/current to best/n_v + 1	0,36%	1
3	DE/rand/n_v+$\diagdown$	1,13%	3
4	DE/rand/1+$\diagdown$	1,37%	3
5	DE/rand/n_v	1,48%	0
6	DE/rand/1	1,62%	1
7	DE/best/1+$\diagdown$	1,97%	0
8	DE/best/1	2,54%	0

Na podstawie tabeli będącej zestawieniem wszystkich testowanych metod mutacji, tabela 6.8, potwierdza się zauważona w poprzednich testach zależność, iż najlepiej działającą metodą jest metoda DE/current to best/n_v +1+$\diagdown$. Średni błąd względny jest równy 0,22%, co jest wartością około dwukrotnie mniejszą od wyników uzyskiwanych przez inne metody mutacji.

6.1.2.2. Reprodukcja rankingowa

W poniżej przeprowadzonej analizie za metodę reprodukcji przyjmuje się metodę rankingową. Testy zostaną przeprowadzone zarówno dla 20 000, jak i dla 50 000 iteracji.

– **20 000 iteracji**

Analiza wyników przeprowadzona zostanie w oparciu o sprowadzenie uzyskanych wartości funkcji celu do postaci ich błędu względnego. Liczba iteracji algorytmu jest równa 20 000.
Analizując tabele 6.9 widoczny staje się fakt, iż najlepsze wyniki uzyskiwane są dla strategii DE-/current to best/n_v + 1+$\diagdown$, jako że aż 4 razy algorytm z udziałem tej metody mutacji osiągał błąd względy równy 0%. W pozostałych iteracjach wartości błędu względnego w przeważającej części również były niewielkie, wartości te wahają się w zakresie (0,1,48%). Współczynnik odchylenia standardowego dla wspomnianej powyżej metody jest równy 0,3 %, co świadczy o bliskim skupieniu wartości w poszczególnych iteracjach wokół średniej. Jest to również jedna z mniejszych wartości odchylenia standardowego, biorąc pod uwagę wartości uzyskiwane przez inne metody mutacji.
Strategia DE/current to best/n_v + 1 nie uwzględniająca zależności opisanych w podrozdziale 4.2.4,

Tabela 6.9. Wartości błędu względnego funkcji celu dla poszczególnych metod mutacji, metoda rankingowa, 20 000 iteracji.

Iteracja	DE/rand/1	DE/rand/1+⤫	DE/best/1	DE/best/1+⤫	DE/rand/n_v	DE/rand/n_v+⤫	DE/current to best/n_v + 1	DE/current to best/n_v + 1+⤫
1	0,85%	0,73%	1,21%	1,33%	0,61%	0,24%	0,24%	0,48%
2	1,09%	0,24%	1,82%	1,09%	0,73%	0,48%	0,24%	0,61%
3	0,48%	0,24%	2,06%	2,54%	0,73%	0,48%	0,85%	0,24%
4	0,48%	0,24%	1,33%	0,48%	0,24%	0,48%	0,48%	0,48%
5	0,73%	0,73%	0,48%	0,48%	0,61%	0,24%	0,24%	0,48%
6	1,09%	0,85%	2,78%	1,21%	0,48%	0,61%	0,24%	0,48%
7	0,85%	0,48%	1,33%	1,21%	0,85%	0,48%	0,24%	0,12%
8	1,21%	0,48%	0,48%	1,57%	1,82%	0,24%	0,24%	0,48%
9	0,24%	0,61%	1,21%	1,57%	0,85%	1,09%	0,48%	0,85%
10	1,33%	0,73%	1,33%	1,21%	1,09%	0,48%	0,61%	0%
11	0,85%	0%	0,85%	1,33%	0,12%	0,12%	1,09%	0%
12	0,24%	0,48%	1,69%	1,57%	0,97%	0,48%	0%	0,48%
13	0,73%	0,85%	0,61%	1,57%	0,61%	0,48%	0,24%	1,09%
14	0,48%	0,12%	0,97%	1,33%	0,48%	0,24%	1,21%	0,85%
15	0%	0,24%	0,48%	0,61%	1,21%	0,24%	0%	0%
16	0,48%	0,61%	2,30%	1,69%	0,61%	0,73%	0,48%	0%
17	0,61%	0,12%	2,54%	2,18%	0,48%	0,61%	0,24%	0,61%
18	1,82%	0,61%	2,18%	1,33%	0,48%	0,48%	0,48%	0,61%
19	1,09%	0,24%	1,21%	1,21%	1,57%	0,48%	0,61%	0,48%
20	0,73%	0,61%	2,66%	0,85%	0,48%	0,48%	0,73%	0,48%
ŚREDNIA	0,77%	0,46%	1,48%	1,32%	0,75%	0,46%	0,45%	0,44%
ODCHYLENIE	0,41%	0,25%	0,73%	0,49%	0,41%	0,21%	0,32%	0,30%

a więc bez wprowadzenia dodatkowego współczynnika ⤫, także przynosi satysfakcjonujące wyniki. Średni błąd względny wynosi 0,45%, a algorytm 2 razy doprowadza funkcję celu do wartości uznawanej przez [9] za najmniejszą znaną wartość. W tabeli 6.8 został przedstawiony ranking metod mutacji ułożony w zależności od uzyskanej przez metody te średniej wartości błędu względnego.

Tabela 6.10. Ranking metod mutacji na podstawie średniej wartości błędu względnego funkcji celu, metoda rankingowa, 20 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	DE/current to best/n_v + 1+⤫	0,44%	4
2	DE/current to best/n_v + 1	0,45%	2
3	DE/rand/n_v+⤫	0,46%	0
4	DE/rand/1+⤫	0,46%	1
5	DE/rand/n_v	0,75%	0
6	DE/rand/1	0,77%	1
7	DE/best/1+⤫	1,32%	0
8	DE/best/1	1,48%	0

Na podstawie tabeli 6.8 można wysnuć wniosek, iż metody takie jak DE/current to best/n_v + 1+⤫, DE/rand/n_v+⤫, DE/rand/1+⤫ oraz DE/best/1+⤫, a wiec metody, które zostały zmodyfikowane współczynnikiem ⤫ uzyskują korzystniejsze wyniki w porównaniu do ich odpowiedników bez modyfikatora. Widoczna staje się zatem skuteczność działania czynnika skalującego ⤫.

– **50 000 iteracji**

Poniższe testy zostały przeprowadzone w celu zbadania czy uzyskane wyniki utrzymają się na poziomie, który osiągają na etapie 20 000 iteracji, czy też następuje poprawa w sytuacji zwiększenia liczby iteracji. Wyniki testów zestawiono w tabeli 6.11.

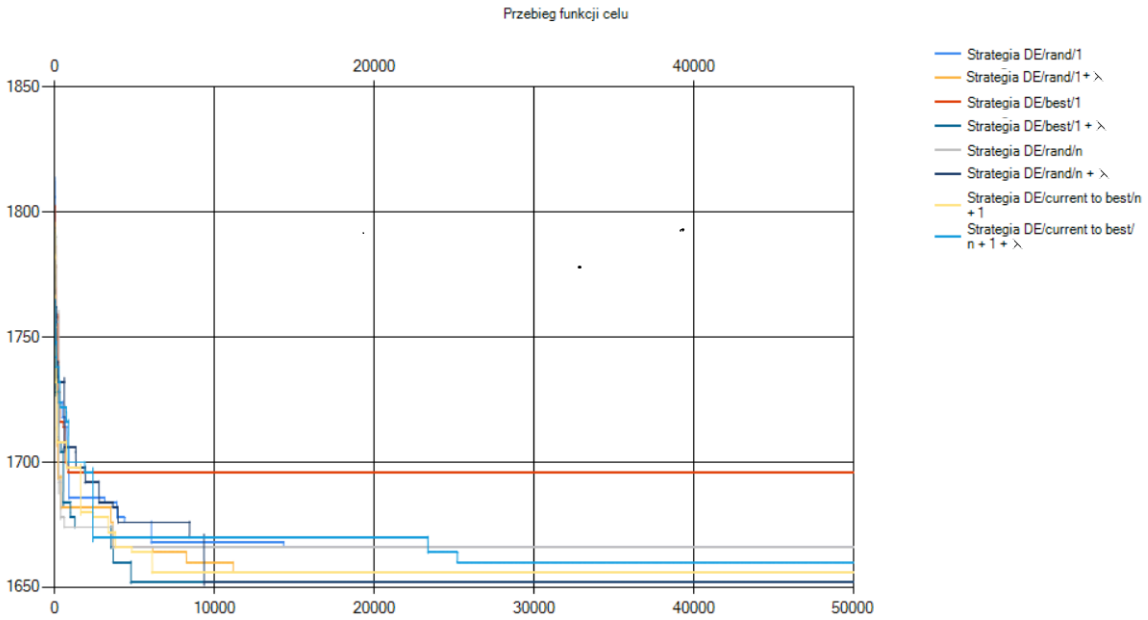
Tabela 6.11. Wartości błędu względnego funkcji celu dla poszczególnych metod mutacji, metoda rankingowa, 50 000 iteracji.

Iteracja	DE/rand/1	DE/rand/1+⤡	DE/best/1	DE/best/1+⤡	DE/rand/n_v	DE/rand/n_v+⤡	DE/current to best/n_v+1	DE/current to best/n_v+1+⤡
1	1,33%	0,85%	1,33%	1,33%	0,24%	0,48%	0,12%	0,48%
2	0,85%	0%	1,21%	0,97%	0,61%	0,12%	0,61%	0,48%
3	0,73%	1,57%	1,57%	0,61%	0,48%	0,24%	0,24%	0,12%
4	0,97%	0,48%	1,33%	1,45%	0%	0,48%	0,97%	0,12%
5	0,48%	0,24%	0,61%	1,45%	0,61%	0,24%	0,24%	0%
6	0,73%	0,48%	1,09%	2,54%	0,12%	0%	0%	0,48%
7	0,24%	0,24%	2,78%	0,85%	0,48%	0,48%	0,85%	0,12%
8	0,48%	0,48%	1,09%	2,18%	0,48%	0,24%	0%	0,24%
9	0,24%	0,48%	2,06%	0,61%	0,48%	0,48%	0,24%	0,48%
10	0,85%	0,48%	1,57%	1,33%	0,48%	0,48%	0,24%	1,09%
11	2,18%	0,48%	1,94%	2,18%	0,61%	0,24%	0,24%	0,12%
12	0%	0,61%	2,78%	1,69%	0,48%	0%	0,24%	0,24%
13	0,73%	0,73%	1,21%	1,45%	0,48%	0,48%	0,24%	0,48%
14	0%	0,24%	3,03%	1,09%	0,48%	0,73%	0,24%	0,24%
15	0,48%	0%	1,33%	1,09%	0,85%	0%	0,12%	0,24%
16	0,48%	0,61%	1,09%	1,45%	0%	0,24%	0,48%	0,24%
17	0,73%	0,24%	1,94%	1,69%	0,61%	0,48%	0,48%	0%
18	0,61%	1,09%	1,82%	1,57%	0,48%	0,61%	0,61%	0,24%
19	0,73%	0,48%	1,09%	1,82%	0,12%	0,12%	0%	0,24%
20	0,61%	0,73%	2,66%	1,69%	0%	0%	0,48%	0,24%
ŚREDNIA	0,67%	0,53%	1,68%	1,45%	0,41%	0,31%	0,33%	0,30%
ODCHYLENIE	0,46%	0,35%	0,67%	0,49%	0,23%	0,22%	0,26%	0,24%

Wartości błędu względnego dla danych metod uległy poprawie w stosunku do ograniczenia liczby iteracji do 20 000. Wynika to z faktu, iż po przekroczeniu tej granicy, w grupach rodzicielskich nadal występują osobniki o różnym od siebie genotypie. Zjawisko to prowadzi do polepszania się wartości funkcji dopasowania. Analizując przebiegi 6.3 można zauważyć, iż stan stabilny utrzymuje się dopiero na poziomie 30 000 iteracji, co widoczne jest na poniższym wykresie.

Poniżej zamieszczono zestawienie funkcji uporządkowanych w kolejności zależnej od uzyskanej przez metody te wartości błędu względnego funkcji celu.

Ranking dla 50 000 iteracji zmienił się w stosunku do rankingu 20 000 iteracji nie tylko w kontekście mniejszych wartości błędu względnego dla niektórych z funkcji, ale również w kontekście pozycji zajmowanych przez metody w rankingu. Nadal najlepsze wyniki uzyskiwane są dla metody DE/current to best/n_v + 1+⤡. Błąd względny tej metody zmniejszył się o 0,14 % w stosunku do jej działania z użyciem algorytmu, w którym liczba iteracji była ustalona na wartość 20 000. Podobne wyniki uzyskują również metody DE/rand/n_v+⤡ orazDE/current to best/n_v +1, zajmując tym samym odpowiednio drugie oraz trzecie miejsce w rankingu. Niezależnie od zastosowanej metody reprodukcji metody DE/best/1+⤡ oraz DE/best/1 odznaczają się wyjątkowo niekorzystnymi wynikami w porównaniu z pozostałymi metodami mutacji. Uzyskiwane przez metody



Rys. 6.3. Przebieg działania algorytmu dla poszczególnych metod mutacji.

Tabela 6.12. Ranking metod mutacji na podstawie średniej wartości błędu względnego funkcji celu, metoda rankingowa, 50 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	DE/current to best/n_v + 1+⋋	0,3%	2
2	DE/rand/n_v+⋋	0,31%	4
3	DE/current to best/n_v + 1	0,33%	3
4	DE/rand/n_v	0,41%	3
5	DE/rand/1+⋋	0,53%	2
6	DE/rand/1	0,67%	2
7	DE/best/1+⋋	1,45%	0
8	DE/best/1	1,68%	0

te wyniki są około dwukrotnie gorsze.

6.1.3. Metody krzyżowania

Metody krzyżowania, o których mowa w podrozdziale zostały szczegółowo opisane w rozdziale 5. Metody krzyżowania zostały przetestowane w każdym z podrozdziałów przy założeniu stałych takich jak:

– macierz odległości,
– macierz przepływu,
– metoda reprodukcji,
– metoda mutacji,

– wielkość populacji

Testy zostały podzielone na trzy główne bloki. Pierwszy z nich zakłada uznanie za metodę reprodukcji klasyczną metodę losową, natomiast za metodę mutacji strategie DE/rand/1. W kolejnych dwóch blokach testowych zostały uwzględnione wyniki otrzymane w wyżej przeprowadzonych testach. Poprzez zestawienie z metodami krzyżowania, możliwe było dokonanie sprawdzenia jaka kombinacja metod z poszczególnych sekcji algorytmu pozwoli na osiągnięcie najbardziej satysfakcjonujących wyników. Dane początkowe, dla których przeprowadzono testy, znajdują się w rozdziale 7.1.1. Dodatkowo zgodnie z informacjami zawartymi w podrozdziale 5.3 zostanie przebadany wpływ współczynnika krzyżowania C_r na działanie poszczególnych metod.

6.1.3.1. Reprodukcja losowa, mutacja DE/rand/1

W przypadku metod krzyżowania testy zostaną przeprowadzone, zgodnie z przyjętym sposobem prowadzenia testów dla metod selekcji oraz metod mutacji, dla 20 000 iteracji, a następnie dla 50 000 iteracji. Metoda losowa oraz strategia DE/rand/1 stanowi klasyczną wersję algorytmu ewolucji różnicowej, testy z użyciem tych metod są przeprowadzane w celu konfrontacji wyników działania wersji klasycznej z wynikami działania metod zmutowanych.

– **20 000 iteracji**
  Testy dla każdej z metod krzyżowania zostały przeprowadzone 20 razy, a następnie na podstawie uzyskanych wartości została obliczona wartość podstawowych miar statystycznych takich jak średnia i odchylenie standardowe. Na podstawie tych wartości oceniona została skuteczność działania poszczególnych metod. Wyniki uzyskiwane w kolejnych iteracjach przez poszczególne metody zostały zestawione w poniższej tabeli. Zapisane są one w formie wartości błędu względnego z uzyskanego wyniku.

  Zauważalny staje się fakt, iż w przypadku wykonywania się programu przez 20 000 iteracji, nie udało się żadnej ze strategii osiągnąć wartości błędu względnego równej 0%. Spośród dostępnych metod najlepsze rezultaty uzyskuje metoda OX, jako że wartość błędu jest równa 0,97 %. Wynik działania strategii krzyżowania dwumianowego znajduje się w niewielkim otoczeniu od wyniku uzyskanego przez krzyżowanie dwumianowe. Z koleii metody CX oraz PMX odznaczają się wyjątkowo dużą wartością błędu względnego, rzędu (3,9 %). Uporządkowany układ od najmniejszej do największej wartości średniego błędu względnego został przedstawiony poniżej.
– **50 000 iteracji** Niezadowalające efekty działania algorytmu uwzględniającego w swoim działaniu metody krzyżowania CX czy też PMX mogą poniekąd wynikać z niewystarczająco dużej liczby iteracji algorytmu. Wraz ze zwiększeniem ilości iteracji istnieją większe szanse na osiąganie przez metody te lepszych wyników.

Tabela 6.13. Wartości błędu względnego funkcji celu dla poszczególnych metod krzyżowania, reprodukcja losowa, mutacja DE/rand/1, 20 000 iteracji.

Iteracja	Dwumianowe	OX	CX	PMX
1	2,30%	1,09%	4,36%	10,53%
2	1,94%	1,33%	2,91%	4,96%
3	0,24%	0,61%	2,78%	11,86%
4	1,33%	1,21%	2,18%	11,50%
5	1,82%	0,48%	2,30%	7,02%
6	3,15%	1,09%	2,42%	9,32%
7	1,21%	1,45%	2,66%	8,11%
8	3,27%	0,48%	3,39%	8,47%
9	0,48%	1,45%	1,82%	11,14%
10	2,91%	0,12%	2,91%	8,72%
11	0,73%	0,85%	2,18%	9,44%
12	0,61%	0,48%	4,84%	10,05%
13	0,97%	0,24%	5,21%	8,84%
14	1,57%	1,09%	3,27%	9,93%
15	0,85%	1,57%	3,51%	9,56%
16	1,21%	2,18%	2,54%	5,21%
17	1,57%	0,97%	1,21%	9,69%
18	0,24%	1,09%	2,91%	7,14%
19	0,97%	0,73%	2,42%	8,96%
20	0,97%	0,85%	4,36%	8,47%
ŚREDNIA	1,42%	0,97%	3,01%	8,95%
ODCHYLENIE	0,89%	0,49%	1,00%	1,79%

Tabela 6.14. Ranking metod krzyżowania na podstawie średniej wartości błędu względnego funkcji celu, reprodukcja losowa, mutacja DE/rand/1, 20 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	OX	0,97%	0
2	Dwumianowe	1,42%	0
3	CX	3,01%	0
4	PMX	8,95%	0

Widoczne staje się osiąganie przez metodę OX wartości błędu względnego równą 0% w 5 z 20 iteracji. W pozostałych 15 iteracjach uzyskiwane wartości są w większości przypadków również niewielkie. Strategia ta osiąga ponad dwukrotnie lepsze wyniki niż jej odpowiednik w postaci metody dwumianowej i ponad sześciu i szesnastokrotnie lepsze w stosunku do metod, odpowiednio CX i PMX. Najmniejsze odchylenie wartości od średniej zauważalne jest dla metody krzyżowania dwumianowego, jest ono równe 0,62%. Nie mniej jednak strategia OX osiąga niewiele większą wartość odchylenia standardowego, 0,71 %.

Tabela 6.15. Wartości błędu względnego funkcji celu dla poszczególnych metod krzyżowania, reprodukcja losowa, mutacja DE/rand/1, 50 000 iteracji.

Iteracja	Dwumianowe	OX	CX	PMX
1	0,85%	2,91%	2,18%	5,57%
2	1,33%	0,48%	4,24%	9,20%
3	1,94%	0,24%	3,63%	8,96%
4	0,61%	0,24%	4,12%	5,45%
5	1,21%	**0%**	1,33%	10,05%
6	0,73%	0,48%	1,69%	11,74%
7	0,24%	1,57%	4,84%	6,30%
8	2,66%	0,24%	1,94%	6,90%
9	0,97%	1,45%	3,15%	9,81%
10	1,57%	**0%**	0,61%	10,90%
11	0,85%	**0%**	3,39%	11,99%
12	1,57%	**0%**	2,30%	8,96%
13	1,09%	0,12%	2,30%	9,44%
14	2,18%	**0%**	4,84%	5,21%
15	0,97%	0,24%	1,82%	8,96%
16	1,09%	0,85%	3,51%	10,77%
17	0,85%	0,12%	2,42%	2,18%
18	1,45%	0,48%	6,05%	8,72%
19	**0%**	0,24%	2,91%	5,81%
20	0,61%	0,12%	2,66%	5,69%
ŚREDNIA	1,14%	0,49%	3,00%	8,13%
ODCHYLENIE	0,62%	0,71%	1,32%	2,53%

Tabela 6.16. Ranking metod krzyżowania na podstawie średniej wartości błędu względnego funkcji celu, reprodukcja losowa, mutacja DE/rand/1, 50 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	OX	0,49%	5
2	Dwumianowe	1,14%	1
3	CX	3%	0
4	PMX	8,13%	0

6.1.3.2. Reprodukcja losowa, DE/current to best/n_v + 1+$\diagup$

Kolejny zestaw testów zakłada użycie metody losowej, jako metody reprodukcji, natomiast za metodę mutacji przyjmuje się strategię DE/current to best/n_v + 1+$\diagup$. Kombinacja dobrana w ten sposób wynika z faktu, iż właśnie dla takiego połączenia metod, algorytm uzyskiwał najlepsze wyniki. Przy założeniu takiego połączenia zostaną przetestowane rozważane w ramach niniejszej pracy metody krzyżowania.

– **20 000 iteracji**

W poniższej tabeli 6.17 znajdują się wartości błędu względnego funkcji celu dla algorytmów zakładających w swoim działaniu użycie poszczególnych metod krzyżowania. Liczba iteracji algorytmu dla pojedynczej instancji testowej jest równa 20 000.

Tabela 6.17 odznacza się wielokrotnym powtarzaniem się wartości błędu względnego równego 0%. Szczególnie wiele razy wartości uznawane przez [9] za najmniejsze jakie udało się uzyskać dla obranych danych wejściowych, są otrzymywane przez metodę OX. Wartość błędu wynosi 0,19

Tabela 6.17. Wartości błędu względnego funkcji celu dla poszczególnych metod krzyżowania, reprodukcja losowa, mutacja DE/current to best/n_v + 1+$\diagup$, 20 000 iteracji.

Iteracja	Dwumianowe	OX	CX	PMX
1	0,61%	**0%**	1,33%	7,99%
2	0,48%	**0%**	0,73%	7,87%
3	0,48%	**0%**	1,94%	7,26%
4	0,24%	1,33%	0,61%	8,84%
5	0,48%	**0%**	1,57%	8,35%
6	0,85%	**0%**	1,09%	8,47%
7	0,24%	0,48%	1,45%	8,84%
8	0,12%	0,12%	**0%**	8,84%
9	0,48%	0,24%	1,45%	8,11%
10	0,24%	**0%**	0,48%	6,17%
11	0,48%	0,48%	1,33%	7,38%
12	0,48%	0,12%	0,61%	8,96%
13	0,24%	0,73%	1,57%	10,05%
14	**0%**	0,24%	2,18%	5,81%
15	0,48%	0,12%	**0%**	6,05%
16	0,85%	**0%**	0,12%	5,93%
17	0,61%	**0%**	0,85%	7,99%
18	0,24%	**0%**	1,82%	5,08%
19	0,24%	**0%**	1,45%	3,87%
20	0,48%	**0%**	1,09%	7,26%
ŚREDNIA	0,42%	0,19%	1,08%	7,46%
ODCHYLENIE	0,21%	0,33%	0,62%	1,50%

%. Wynik ten znacznie odbiega od wartości uzyskiwanych przez pozostałe metody, szczególnie przez metody CX oraz PMX.

Tabela 6.18. Ranking metod krzyżowania na podstawie średniej wartości błędu względnego funkcji celu, reprodukcja losowa, mutacja DE/current to best/n_v + 1+$\diagup$, 20 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	OX	0,19%	11
2	Dwumianowe	0,42%	1
3	CX	1,08%	2
4	PMX	7,46%	0

– **50 000 iteracji**

Istnieje możliwość, iż przy zwiększeniu ilości iteracji, wartości uzyskiwane przez poszczególne metody ulegną zmniejszeniu. W tym celu testy zostają powtórzone z liczbą iteracji równą 50 000. Na podstawie tabeli 6.23 zauważalne staje się wyjątkowo korzystne działanie metody krzyżowania OX, jako że średnia wartość błędu względnego jest równa 0,06 %. W 15 na 20 prób wartość błędu względnego jest równa 0%, w pozostałych 5 próbach nie przekracza ona wartości 0,48 %. Również w kontekście odchylenia standardowego uzyskana wartość jest niewielka i wynosi 0,12%, co świadczy o dużym skupieniu wyników wokół wartości średniej.

Działanie metody krzyżowania dwumianowego oraz krzyżowania CX również okazało się przynosić lepsze efekty przy zwiększeniu liczby iteracji do 50 000. Natomiast metoda krzyżowania

Tabela 6.19. Wartości błędu względnego funkcji celu dla poszczególnych metod krzyżowania, reprodukcja losowa, mutacja DE/current to best/n_v + 1+⋋, 50 000 iteracji.

Iteracja	Dwumianowe	OX	CX	PMX
1	0,48%	0%	1,33%	8,60%
2	0,24%	0%	0,73%	4,96%
3	0%	0%	1,09%	9,93%
4	0,48%	0,12%	0,85%	8,23%
5	0,61%	0%	1,09%	6,54%
6	0%	0%	0,85%	9,81%
7	0,12%	0%	0,48%	9,32%
8	0,12%	0%	1,09%	6,05%
9	0,48%	0,24%	0,97%	6,54%
10	0,24%	0%	1,09%	9,20%
11	0,12%	0%	0,85%	6,05%
12	0,24%	0%	0,12%	7,14%
13	0%	0%	0,85%	7,99%
14	0,48%	0,12%	1,69%	8,84%
15	0%	0%	0,73%	8,23%
16	0,12%	0,24%	0,12%	7,75%
17	0,48%	0%	0,61%	3,63%
18	0,48%	0%	1,21%	7,51%
19	0,48%	0%	1,21%	10,90%
20	0%	0,48%	0,85%	6,42%
ŚREDNIA	0,26%	0,06%	0,89%	7,68%
ODCHYLENIE	0,21%	0,12%	0,37%	1,75%

Tabela 6.20. Ranking metod krzyżowania na podstawie średniej wartości błędu względnego funkcji celu, reprodukcja losowa, mutacja DE/current to best/n_v + 1+⋋, 50 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	OX	0,06%	15
2	Dwumianowe	0,26%	5
3	CX	0,89%	0
4	PMX	7,68%	0

PMX utrzymuje wyniki na tym samym poziomie. Oznacza to, że algorytm z zastosowaniem tej metody jako metody krzyżowania na poziomie 20 000 iteracji zachowuje już stan stabilny, a więc funkcja celu nie ulega już dalszemu zmniejszaniu.

6.1.3.3. Reprodukcja rankingowa, DE/current to best/n_v + 1+⋋

W przeprowadzonych testach dobrymi wynikami wyróżniała się koncepcja algorytmu, w której metodą reprodukcji była metoda rankingowa, a metodą mutacji startegia DE/current to best/n_v + 1+⋋. Wyniki tej kombinacji były porównywalne z wynikami koncepcji opisanej w poprzednim podrozdziale, a mianowicie połączenia metody losowej z metodą DE/current to best/n_v + 1+⋋. Z tego względu w poniższym podrozdziale zostaną przeprowadzone testy z metodą rankingową jako metodą reprodukcji.

– **20 000 iteracji**

Tabela 6.21. Wartości błędu względnego funkcji celu dla poszczególnych metod krzyżowania, reprodukcja rankingowa, mutacja DE/current to best/n_v + 1+⋋, 20 000 iteracji.

Iteracja	Dwumianowe	OX	CX	PMX
1	0,48%	0,12%	1,33%	7,75%
2	0%	0%	1,45%	7,75%
3	0,24%	0,61%	0,73%	5,45%
4	0,48%	0,12%	1,45%	9,32%
5	0,12%	0,12%	1,21%	9,81%
6	0,12%	0,48%	0,85%	6,42%
7	1,09%	0,73%	0%	8,96%
8	0,12%	0,48%	1,45%	6,30%
9	0,24%	0%	0,48%	7,14%
10	0%	0%	1,69%	10,29%
11	0,12%	0,48%	1,45%	7,99%
12	0,12%	0%	1,09%	6,17%
13	0,48%	0,12%	0,24%	9,69%
14	0,85%	0%	0,48%	7,14%
15	0,24%	0%	1,09%	7,63%
16	0,24%	0,24%	0,73%	6,30%
17	0,24%	0%	1,21%	6,17%
18	0,24%	0%	1,09%	5,57%
19	0,61%	0,12%	0,97%	7,63%
20	0,48%	0%	1,33%	8,47%
ŚREDNIA	0,33%	0,18%	1,02%	7,60%
ODCHYLENIE	0,27%	0,23%	0,44%	1,42%

Uzyskane i zestawione w tabeli 6.21 wartości błędu względnego przyjmują wartości bliskie wartościom uzyskiwanym przez strategie, w której to metodą reprodukcji jest metoda losowa. Największa różnica w wartości błędu pomiędzy tymi kompozycjami metod wynosi 0,14 % i występuje ona w przypadku krzyżowania PMX. Krzyżowanie to odznacza się stosunkowo dużym odchyleniem standardowym o wartości 1,5 %. W przypadku innych metod różnica w działaniu jest znacznie mniejsza.

Tabela 6.22. Ranking metod krzyżowania na podstawie średniej wartości błędu względnego funkcji celu, reprodukcja rankingowa, mutacja DE/current to best/n_v + 1+⋋, 20 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	OX	0,18%	9
2	Dwumianowe	0,33%	2
3	CX	1,02%	1
4	PMX	7,6%	0

– **50 000 iteracji**

W przypadku gdy liczba iteracji jest równa 50 000 uzyskiwane wyniki dla strategii algorytmu, w której uwzględnimy metodę rankingową, są nieco gorsze niż te zestawione w tabeli 6.23. Różnice są jednak niewielkie, a średnia wartość błędu względnego uzyskana przy użyciu metody OX jest równa 0,1 % co jest drugą najlepszą wartością błędu względnego uzyskaną we wszystkich

Tabela 6.23. Wartości błędu względnego funkcji celu dla poszczególnych metod krzyżowania, reprodukcja rankingowa, mutacja DE/current to best/n_v + 1+⋋, 50 000 iteracji.

Iteracja	Dwumianowe	OX	CX	PMX
1	0,48%	0%	1,09%	6,17%
2	1,45%	0%	1,09%	9,20%
3	0%	0,48%	1,33%	10,05%
4	0,24%	0%	1,45%	8,60%
5	0,48%	0%	1,57%	7,75%
6	0,48%	0%	0,73%	8,84%
7	0,48%	0%	0,61%	10,29%
8	0,61%	0,12%	0,48%	5,33%
9	0,61%	0%	0,85%	4,96%
10	0,12%	0%	0,85%	7,02%
11	0,48%	0%	0,85%	7,51%
12	0%	0,12%	0,85%	7,99%
13	0,24%	0%	0,97%	7,14%
14	0,24%	0%	0,61%	7,87%
15	0,85%	0%	1,45%	8,84%
16	0,85%	0,12%	0,61%	9,93%
17	0%	0%	1,45%	6,54%
18	0%	0,48%	0,73%	10,17%
19	0%	0,61%	0%	8,23%
20	0,61%	0%	0,12%	7,63%
ŚREDNIA	0,41%	0,10%	0,88%	8,00%
ODCHYLENIE	0,36%	0,19%	0,42%	1,50%

przeprowadzanych testach. Podłożem różnic występujących pomiędzy dwoma strategiami, opisanymi odpowiednio w podrozdziale 6.1.3.2 oraz 6.1.3.3 może być działanie czynnika losowego.

Tabela 6.24. Ranking metod krzyżowania na podstawie średniej wartości błędu względnego funkcji celu, reprodukcja rankingowa, mutacja DE/current to best/n_v + 1+⋋, 50 000 iteracji.

Miejsce	Metoda	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	OX	0,1%	14
2	Dwumianowe	0,41%	5
3	CX	0,88%	1
4	PMX	8%	0

6.1.4. Modyfikacje parametrów operatorów genetycznych

Testy współczynników F oraz C_r zostały przeprowadzone z wykorzystaniem kilku wariantów zmodyfikowanego algorytmu ewolucji różnicowej, dostosowanego do rozwiązywania problemu kwadratowego zagadnienia przydziału. Wybrane zostały te strategie, które na podstawie uprzednio przeprowadzonych testów, opisanych szczegółowo w rozdziałach 6.1.1, 6.1.2 oraz 6.1.3, uzyskały najlepsze wyniki. Testy zostały wykonane dla 50 000 iteracji algorytmu, a następnie powtórzone 20 razy. W wyniku tego działania możliwe było policzenie średniego błędu względnego z 20 iteracji. Wartości te zostały umieszczone w odpowiednich tabelach. Dodatkowo tabele, w celu konfrontacji wyników, zawierają informacje

o średniej wartości błędu względnego dla wariantu algorytmu, w którym współczynnik F jest ustalony na stałym poziomie i wynosi $F = 0,8$.

6.1.4.1. Modyfikacje parametru F

Najmniejsza średnia wartość błędu względnego uzyskiwana była dla algorytmów wykorzystujących w swoim działaniu następujące metody:

– metoda reprodukcji losowej, metoda mutacji DE/current to best/n_v + 1+$\diagup$, metoda krzyżowania OX,

– metoda reprodukcji rankingowej, metoda mutacji DE/current to best/n_v + 1+$\diagup$, metoda krzyżowania OX,

Dynamiczna zmiana czynnika skalującego zostanie przetestowana w kontekście działania wszystkich metod mutacji, by zobaczyć wpływ działania tego współczynnika na każdą z nich. Jako że w rozdziale 4.3 zaprezentowane zostały dwie metody dynamicznej zmiany parametru F, poniższe testy również zostały podzielone w taki sposób.

– **Modyfikacja parametru F zgodnie z funkcją określoną w podrozdziale 4.3.1**

Tabela 6.25. Wartości średniego błędu względnego funkcji celu dla poszczególnych metod mutacji ze zmiennym parametrem F zgodnie z funkcją 4.3.1.

Wersja algorytmu	DE/rand/1	DE/rand/1+$\diagup$	DE/best/1	DE/best/1+$\diagup$	DE/rand/n_v	DE/rand/n_v+$\diagup$	DE/current to best/n_v + 1	DE/current to best/n_v + 1+$\diagup$
Reprodukcja losowa,krzyżowanie OX, F = 0,8	0,77%	0,33%	1,10%	0,49%	0,62%	0,43%	0,19%	0,10%
Reprodukcja losowa,krzyżowanie OX, F zgodne z 4.3.1	0,44%	0,39%	2,05%	0,81%	0,48%	0,33%	0,44%	0,09%
Reprodukcja rankingowa,krzyżowanie OX, F = 0,8	0,44%	0,24%	0,96%	0,29%	0,49%	0,21%	0,34%	0,10%
Reprodukcja rankingowa,krzyżowanie OX, F zgodne z 4.3.1	1,25%	0,38%	2,52%	0,64%	0,61%	0,51%	0,42%	0,09%

Wersja algorytmu, stosująca reprodukcje losową, krzyżowanie OX oraz dynamiczną zmianę parametru, dla metod takich jak DE/rand/1, DE/rand/n_v, DE/rand/n_v+$\diagup$ oraz DE/current to best/n_v+$\diagup$ okazała się przynosić lepsze efekty. Szczególną uwagę należy zwrócić na zastosowanie mechanizmu dostrajania parametru mutacji w kontekście metody DE/current to best/n_v+$\diagup$, ponieważ kombinacja ta pozwala dodatkowo na osiągnięcie najlepszego wyniku spośród wszystkich testowanych przypadków zamieszonych w tabeli 6.25. Biorąc pod uwagę algorytm, w którym metodą reprodukcji była metoda rankingowa, wartości funkcji celu dla każdego rodzaju mutacji były wyższe od wersji algorytmu z niezmiennym współczynnikiem F ustawionym na stałą wartość $F = 0,8$. Modyfikacja w tym przypadku nie wpływa zatem korzystnie na działanie algorytmu.

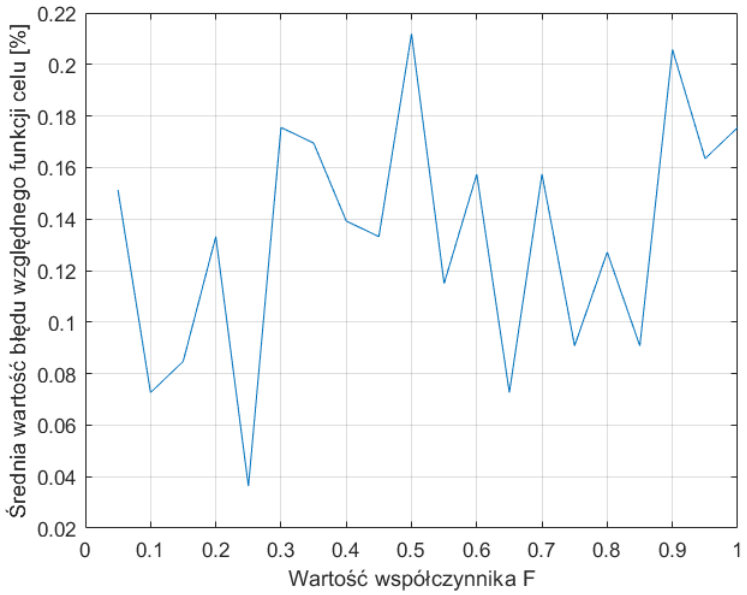
– **Modyfikacja parametru F w zakresie (0,1)**
W celu zbadania, dla jakiej wartości współczynnika F algorytm uzyska najlepsze wyniki, przeprowadzono analizę działania algorytmu dla różnych wartości parametru F. Wartość współczynnika

zwiększana była z każdym wywołaniem algorytmu o 0,05, w zakresie (0,1). W ten sposób przetestowanych zostało 20 różnych wartości czynnika skalującego. W poniższej tabeli 6.26 zaprezentowano średnie wartości błędu względnego obliczone na podstawie 20 iteracji algorytmu.

Tabela 6.26. Wartości średniego błędu względnego funkcji celu dla algorytmu ze zmiennym parametrem F w zakresie (0,1).

Wartość współczynnika F	Średni błąd względny funkcji celu	Wartość współczynnika F	Średni błąd względny funkcji celu
0,05	0,15%	**0,55**	0,12%
0,1	0,07%	**0,6**	0,16%
0,15	0,08%	**0,65**	0,07%
0,2	0,13%	**0,7**	0,16%
0,25	0,04%	**0,75**	0,09%
0,3	0,18%	**0,8**	0,10%
0,35	0,17%	**0,85**	0,09%
0,4	0,13%	**0,9**	0,21%
0,45	0,13%	**0,95**	0,16%
0,5	0,21%	**1**	0,18%

Zauważalne staje się osiąganie najniższych średnich wartości funkcji celu (0,04%) przez wersję algorytmu, w której współczynnik mutacji został ustalony na poziomie $F = 0,25$. Jest to jednocześnie najmniejsza wartość tej miary we wszystkich przeprowadzonych testach. Wyniki osiągane przez pozostałe koncepcje znajdują się w zakresie (0, 0,21%) co świadczy o skuteczności działania algorytmu, w którym metodą reprodukcji jest metoda losowa, metodą mutacji strategia DE/current to best/n_v + 1+⩒, natomiast metodę krzyżowania stanowi krzyżowanie OX, x każdą z wartości przyjętą dla parametru F. W celu uzyskania pełniejszego obrazu zachodzącego zjawiska, wartości zawarte w tabeli 6.26 zestawione zostały w postaci wykresu 6.4.



Rys. 6.4. Zależność współczynnika mutacji F od uzyskiwanych wartości funkcji celu.

– **Modyfikacja parametru F zgodnie z rozkładem normalnym, podrozdział 4.3.2**
Wśród uzyskiwanych wyników, zestawionych w powyżej, nie występuje trend pozwalający ustalić, na jakiej zasadzie czynnik skalujący wpływa na działanie algorytmu. Dostrajanie parametru F zgodnie z rozkładem normalnym nie znajduje więc w tym przypadku zastosowania.

6.1.4.2. Modyfikacje parametrów C_r

Niewiele gorsze wyniki od wspomnianych wyżej dwóch strategii uzyskiwały następujące warianty algorytmu:

– metoda reprodukcji losowej, metoda mutacji DE/current to best/n_v + 1+$\diagup$, metoda krzyżowania dwumianowego,

– metoda reprodukcji rankingowej, metoda mutacji DE/current to best/n_v + 1+$\diagup$, metoda krzyżowania dwumianowego,

Testy zostaną przeprowadzone z podziałem w zależności od sposobu aktualizacji współczynnika krzyżowania.

– **Modyfikacja parametru C_r zgodnie z funkcją określoną w podrozdziale 5.3.1**

Tabela 6.27. Wartości średniego błędu względnego funkcji celu dla metody krzyżowania dwumianowego ze zmiennym parametrem C_r zgodnie z funkcją 5.3.1.

Wersja algorytmu	Dwumianowe
Reprodukcja losowa,mutacja DE/current to best/n_v + 1+$\diagup$, C_r = rand(0,1)	0,26%
Reprodukcja losowa,mutacja DE/current to best/n_v + 1+$\diagup$, C_r zgodne z 5.3.1	0,1%
Reprodukcja rankingowa,mutacja DE/current to best/n_v + 1+$\diagup$, C_r = rand(0,1)	0,4%
Reprodukcja rankingowa,mutacja DE/current to best/n_v + 1+$\diagup$, C_r zgodne z 5.3.1	0,11%

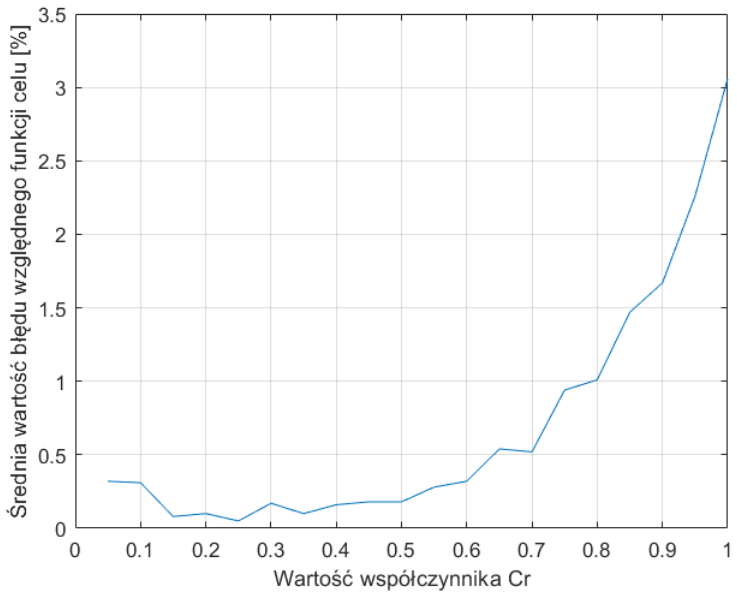
Zgodnie z tabelą 6.27 wprowadzenie dynamicznej zmiany parametru C_r wpływa korzystnie zarówno na wersję algorytmu, w której metodą reprodukcji jest metoda losowa, jak i rankingowa. W przypadku metody rankingowej modyfikacja parametru zgodnie z funkcją 4.3.1 przynosi około czterokrotnie lepsze efekty. Niemniej jednak w wyniku testów udało się uzyskać wersje algorytmu odznaczające się mniejszą średnią wartością błędu względnego funkcji celu.

– **Modyfikacja parametru C_r w zakresie (0,1)**
W celu zbadania dla jakiej wartości parametru C_r algorytm uzyskuje najmniejszą wartość funkcji celu, przeprowadzono testy dla 20 różnych jego wartości. Początkowo parametr miał wartość C_r=0,3 a następnie był sukcesywnie zwiększany o 0,05, aż do osiągnięcia ostatecznej wartości równej 1. Testy zostały przeprowadzone z użyciem algorytmu działającego w oparciu o metodę reprodukcji losowej, mutacji DE/current to best/n_v+1+$\diagup$ oraz krzyżowania dwumianowego. Poniżej znajduje się tabela podsumowująca uzyskane wyniki, przedstawione są one w postaci średnich wartości błędu względnego funkcji celu.

Tabela 6.28. Wartości średniego błędu względnego funkcji celu dla algorytmu ze zmiennym parametrem C_r w zakresie (0,1).

Wartość współczynnika C_r	Średni błąd względny funkcji celu	Wartość współczynnika C_r	Średni błąd względny funkcji celu
0,05	0,32%	**0,55**	0,28%
0,1	0,31%	**0,6**	0,32%
0,15	0,08%	**0,65**	0,54%
0,2	0,1%	**0,7**	0,52%
0,25	0,05%	**0,75**	0,94%
0,3	0,17%	**0,8**	1,01%
0,35	0,1%	**0,85**	1,47%
0,4	0,16%	**0,9**	1,67%
0,45	0,18%	**0,95**	2,26%
0,5	0,18%	**1**	3,06%

Zauważalne staje się osiąganie przez algorytm najmniejszej średniej wartości funkcji celu (0,05 %) przy ustaleniu współczynnika na poziomie C_r = 0,25. Uzyskany wynik jest niewątpliwie lepszy w porównaniu do wersji algorytmu, w której wartość C_r była ustalana losowo, gdyż wartość ta wówczas była równa 0,26%. Podejście to jest odmienne i wprowadza modyfikacje w stosunku do klasycznej wersji krzyżowania dwumianowego, gdyż wartość ta na cały czas trwania algorytmu ustalona jest na stałym poziomie. Wśród otrzymanych wyników istnieje również pewna tendencja ukazująca charakter pracy algorytmu w zależności od wartości współczynnika C_r, co można zaobserwować analizując wykres 6.5.



Rys. 6.5. Zależność współczynnika krzyżowania C_r od uzyskiwanych wartości funkcji celu.

– **Modyfikacja parametru C_r zgodnie z rozkładem normalnym, podrozdział 5.3.2**

Widoczny na wykresie 6.5 trend kształtuje się w sposób przybliżenie eksponencjalny. Istnieje zatem sens badania wpływu na algorytm dynamicznej zmiany parametru C_r zgodnie z rozkładem normalnym. Dla wartości współczynnika C_r większych niż 0,65 średnie wartości funkcji celu zaczynają przybierać znacznie większe wartości. Na tej podstawie można ustalić wartości współczynników μ oraz σ. Za wartość średnią μ została uznana μ=0,25, natomiast odchylenie standardowe σ było równe 0,25.

Tabela 6.29. Wartości średniego błędu względnego funkcji celu dla metody krzyżowania dwumianowego ze zmiennym parametrem C_r zgodnie z rozkładem normalnym.

Wersja algorytmu	Dwumianowe
Reprodukcja losowa,mutacja DE/current to best/n_v + 1+⋋, C_r = rand(0,1)	0,26%
Reprodukcja losowa,mutacja DE/current to best/n_v + 1+⋋, C_r zgodne z 5.3.2	0,14%
Reprodukcja rankingowa,mutacja DE/current to best/n_v + 1+⋋, C_r = rand(0,1)	0,4%
Reprodukcja rankingowa,mutacja DE/current to best/n_v + 1+⋋, C_r zgodne z 5.3.2	0,14%

Zastosowanie rozkładu normalnego w celu wyznaczania wartości współczynnika C_r wpływa korzystnie na działanie algorytmu. Niemniej jednak w przeprowadzonych uprzednio testach istniały wersje algorytmu odznaczające się skuteczniejszymi wynikami.

6.2. Dane wejściowe instancja II

Algorytm zostanie przetestowany z użyciem danych wejściowych, których rozmiar jest większy od poprzedniej instancji testowej, wynosi 18. Zarówno macierz przepływu, jak i macierz odległości jest symetryczna. Dane testowe zostały zaczerpnięte z biblioteki [9], gdzie również znajdowała się informacja o najmniejszej wartości funkcji celu, jaką udało się uzyskać dla danej instancji testowej. W tym przypadku wartość ta jest równa 5358. Wartość ta traktowana jest jako wartość dokładna i stanowi podstawę do obliczania wartości błędu względnego poszczególnych pomiarów. Poniżej znajdują się odpowiednio macierz przepływu oraz macierz odległości.

Macierz_trasy =

$$\begin{pmatrix}
0 & 1 & 2 & 2 & 3 & 4 & 4 & 5 & 3 & 5 & 6 & 7 & 8 & 9 & 7 & 8 & 4 & 5 \\
1 & 0 & 1 & 1 & 2 & 3 & 3 & 4 & 2 & 4 & 5 & 6 & 7 & 8 & 6 & 7 & 3 & 4 \\
2 & 1 & 0 & 2 & 1 & 2 & 2 & 3 & 1 & 3 & 4 & 5 & 6 & 7 & 5 & 6 & 2 & 3 \\
2 & 1 & 2 & 0 & 1 & 2 & 2 & 3 & 3 & 3 & 4 & 5 & 6 & 7 & 5 & 6 & 4 & 5 \\
3 & 2 & 1 & 1 & 0 & 1 & 1 & 2 & 2 & 2 & 3 & 4 & 5 & 6 & 4 & 5 & 3 & 4 \\
4 & 3 & 2 & 2 & 1 & 0 & 2 & 3 & 3 & 1 & 2 & 3 & 4 & 5 & 3 & 4 & 4 & 5 \\
4 & 3 & 2 & 2 & 1 & 2 & 0 & 1 & 3 & 1 & 2 & 3 & 4 & 5 & 3 & 4 & 4 & 5 \\
5 & 4 & 3 & 3 & 2 & 3 & 1 & 0 & 4 & 2 & 1 & 2 & 3 & 4 & 2 & 3 & 5 & 6 \\
3 & 2 & 1 & 3 & 2 & 3 & 3 & 4 & 0 & 4 & 5 & 6 & 7 & 8 & 6 & 7 & 1 & 2 \\
5 & 4 & 3 & 3 & 2 & 1 & 1 & 2 & 4 & 0 & 1 & 2 & 3 & 4 & 2 & 3 & 5 & 6 \\
6 & 5 & 4 & 4 & 3 & 2 & 2 & 1 & 5 & 1 & 0 & 1 & 2 & 3 & 1 & 2 & 6 & 7 \\
7 & 6 & 5 & 5 & 4 & 3 & 3 & 2 & 6 & 2 & 1 & 0 & 1 & 2 & 2 & 3 & 7 & 8 \\
8 & 7 & 6 & 6 & 5 & 4 & 4 & 3 & 7 & 3 & 2 & 1 & 0 & 1 & 1 & 2 & 8 & 9 \\
9 & 8 & 7 & 7 & 6 & 5 & 5 & 4 & 8 & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 9 & 10 \\
7 & 6 & 5 & 5 & 4 & 3 & 3 & 2 & 6 & 2 & 1 & 2 & 1 & 2 & 0 & 1 & 7 & 8 \\
8 & 7 & 6 & 6 & 5 & 4 & 4 & 3 & 7 & 3 & 2 & 3 & 2 & 1 & 1 & 0 & 8 & 9 \\
4 & 3 & 2 & 4 & 3 & 4 & 4 & 5 & 1 & 5 & 6 & 7 & 8 & 9 & 7 & 8 & 0 & 1 \\
5 & 4 & 3 & 5 & 4 & 5 & 5 & 6 & 2 & 6 & 7 & 8 & 9 & 10 & 8 & 9 & 1 & 0
\end{pmatrix}$$

Macierz_kosztu =

$$\begin{pmatrix}
0 & 3 & 4 & 6 & 8 & 5 & 6 & 6 & 5 & 1 & 4 & 6 & 1 & 5 & 4 & 5 & 6 & 8 \\
3 & 0 & 6 & 3 & 7 & 9 & 9 & 2 & 2 & 7 & 4 & 7 & 9 & 6 & 3 & 2 & 6 & 6 \\
4 & 6 & 0 & 2 & 6 & 4 & 4 & 4 & 2 & 6 & 3 & 6 & 5 & 6 & 2 & 6 & 5 & 7 \\
6 & 3 & 2 & 0 & 5 & 5 & 3 & 3 & 9 & 4 & 3 & 6 & 3 & 4 & 7 & 8 & 3 & 2 \\
8 & 7 & 6 & 5 & 0 & 4 & 3 & 4 & 5 & 7 & 6 & 7 & 7 & 3 & 3 & 3 & 4 & 4 \\
5 & 9 & 4 & 5 & 4 & 0 & 8 & 5 & 5 & 5 & 7 & 5 & 1 & 8 & 5 & 4 & 3 & 3 \\
6 & 9 & 4 & 3 & 3 & 8 & 0 & 6 & 8 & 4 & 6 & 7 & 1 & 8 & 5 & 6 & 7 & 6 \\
6 & 2 & 4 & 3 & 4 & 5 & 6 & 0 & 1 & 5 & 5 & 3 & 7 & 5 & 9 & 4 & 4 & 4 \\
5 & 2 & 2 & 9 & 5 & 5 & 8 & 1 & 0 & 4 & 5 & 2 & 4 & 5 & 4 & 5 & 4 & 7 \\
1 & 7 & 6 & 4 & 7 & 5 & 4 & 5 & 4 & 0 & 7 & 7 & 5 & 6 & 5 & 5 & 6 & 10 \\
4 & 4 & 3 & 3 & 6 & 7 & 6 & 5 & 5 & 7 & 0 & 9 & 6 & 5 & 1 & 8 & 5 & 3 \\
6 & 7 & 6 & 6 & 7 & 5 & 7 & 3 & 2 & 7 & 9 & 0 & 6 & 5 & 4 & 5 & 4 & 6 \\
1 & 9 & 5 & 3 & 7 & 1 & 1 & 7 & 4 & 5 & 6 & 6 & 0 & 5 & 7 & 4 & 5 & 2 \\
5 & 6 & 6 & 4 & 3 & 8 & 8 & 5 & 5 & 6 & 5 & 5 & 5 & 0 & 5 & 3 & 2 & 4 \\
4 & 3 & 2 & 7 & 3 & 5 & 5 & 9 & 4 & 5 & 1 & 4 & 7 & 5 & 0 & 8 & 5 & 6 \\
5 & 2 & 6 & 8 & 3 & 4 & 6 & 4 & 5 & 5 & 8 & 5 & 4 & 3 & 8 & 0 & 6 & 8 \\
6 & 6 & 5 & 3 & 4 & 3 & 7 & 4 & 4 & 6 & 5 & 4 & 5 & 2 & 5 & 6 & 0 & 3 \\
8 & 6 & 7 & 2 & 4 & 3 & 6 & 4 & 7 & 10 & 3 & 6 & 2 & 4 & 6 & 8 & 3 & 0
\end{pmatrix}$$

Liczba iteracji ze względu na wielkość danych testowych została zwiększona do 150000. Jako że nie istniała jednoznacznie określona najlepiej działająca wersja algorytmu, testy zostały przeprowadzone z wykorzystaniem kilku wersji algorytmu, które w uprzednio przeprowadzonych testach okazały się działać najkorzystniej. Były to strategie zakładające odpowiednio:

1. reprodukcja losowa, mutacja DE/current to best/n_v + 1+$\diagup$, krzyżowanie OX, F = 0,8,
2. reprodukcja losowa, mutacja DE/current to best/n_v + 1+$\diagup$, krzyżowanie OX, F = 0,25,
3. reprodukcja losowa, mutacja DE/current to best/n_v + 1+$\diagup$, krzyżowanie OX, F zgodnie z funkcją 4.3.1,
4. reprodukcja rankingowa, mutacja DE/current to best/n_v + 1+$\diagup$, krzyżowanie OX, F zgodnie z funkcją 4.3.1,
5. reprodukcja losowa, mutacja DE/current to best/n_v + 1+$\diagup$, krzyżowanie dwumianowe, C_r = 0,25, F = 0,8

W tabeli 6.30 zestawione zostały wartości błędu względnego osiągane przez poszczególne strategie. W tabeli przyjęto numeracje kolumn zgodnie w wyżej umieszczoną numeracją strategii.

Tabela 6.30. Wartości błędu względnego funkcji celu dla poszczególnych wersji algorytmu, dane wejściowe II.

Iteracja	1	2	3	4	5
1	0%	0,49%	0,15%	0%	0%
2	0,04%	0,30%	0,34%	0,22%	0%
3	0,60%	0,15%	0,67%	0,04%	0,04%
4	0,82%	0,34%	0,19%	0,26%	0%
5	0,07%	0,22%	0,60%	0%	0%
6	0,63%	0,15%	0,19%	0%	0%
7	0,60%	0,04%	0,34%	0%	0%
8	0%	0,34%	0,26%	0%	0%
9	0,60%	0,56%	0,71%	0,11%	0%
10	0,15%	0,15%	0,56%	0,15%	0,15%
11	0,22%	0,15%	0,49%	0,11%	0%
12	0,26%	0,34%	0,30%	0%	0%
13	0,34%	0%	0,22%	0,15%	0%
14	0,86%	0,37%	0,49%	0%	0%
15	0,15%	0,22%	0%	0,15%	0,04%
16	0,60%	0,19%	0,41%	0,04%	0%
17	0,60%	0,19%	0,37%	0,15%	0,15%
18	0,49%	0,22%	0,60%	0,04%	0,22%
19	0,22%	0,34%	0,26%	0%	0%
20	0,19%	0,15%	0,67%	0,04%	0,07%
ŚREDNIA	0,37%	0,24%	0,39%	0,07%	0,03%
ODCHYLENIE	0,27%	0,14%	0,20%	0,08%	0,06%

Na podstawie tabeli zauważyć można najkorzystniejsze działanie strategii 5, gdyż średnia wartość błędu względnego jest najmniejsza spośród wszystkich metod i wynosi jedynie 0,03%. Odchylenie standardowe jest również niewielkie (0,06%) co oznacza, iż algorytm ten charakteryzuje się dużym skupieniem wyników wokół średniej wartości. Strategia zakładająca zastosowanie reprodukcji losowej, mutacji DE/current to best/n_v + 1+$\diagup$ oraz krzyżowania dwumianowego z wartością współczynnika ustaloną na stałym poziomie równym $C_r = 0,25$, aż w 14 na 20 wykonanych iteracji algorytmu osiągała wartość

błędu względnego równą 0%. Satysfakcjonującym wynikiem odznacza się także strategia 4 osiągająca wartość średniego błędu względnego równą 0,07%. Od strategii 3 odróżnia ją jedynie zastosowanie metody rankingowej jako metody reprodukcji, niemniej jednak zabieg ten pozwala na osiągnięcie ponad czterokrotnie lepszych wyników. Zależności pomiędzy metodami zestawione zostały w formie poniższego rankingu.

Tabela 6.31. Ranking zastosowanych strategii dla danych wejściowych II.

Miejsce	Strategia	Średni błąd względny funkcji celu	Liczba osiągnietych wartośći 0%
1	5	0,03%	14
2	4	0,07%	8
3	2	0,24%	1
4	1	0,37%	2
4	3	0,39%	0

Strategia 1, zakładająca przypisanie do współczynnika mutacji F wartości równej 0,8, oraz 3 używająca w tym celu funkcji określonej w podrozdziale 4.3.1, znacząco odbiegają od wyników uzyskiwanych przez strategie 5 oraz 4.

6.3. Pozostałe testy danych wejściowych

W rozdziale tym zostały przeprowadzone testy z zastosowaniem wersji algorytmu, która okazała się działać najskuteczniej w testach opisanych w powyższych rozdziałach. W testach dla instancji II wersja ta osiągnęła pierwsze miejsce w rankingu, natomiast dla instancji I drugie miejsce. Za wersję tą został uznany algorytm wykorzystujący w swoim działaniu metodę reprodukcji losowej, metodę mutacji DE/current to best/n_v + 1+$\diagup$ oraz metodę krzyżowania dwumianowego z parametrem C_r równym 0,25. Instancje danych wejściowych wraz z odpowiednią dla nich nomenklaturą zostały zaczerpnięte z biblioteki QAPLIB [9]. Różnią się one między sobą przede wszystkim rozmiarem oraz najmniejszą wartością funkcji celu jaką udało się uzyskać dla konkretnej instancji danych. Liczba iteracji dla której wykonano testy była równa 150 000. W poniższej tabeli zostały zestawione uzyskane wyniki. Dla każdej instancji testowej została umieszczona wartość funkcji celu uznawana przez biblioteke [9] za najmniejszą. Względem tej wartości obliczany był średni błąd względny funkcji celu uzyskany na podstawie 20 powtórzeń algorytmu dla każdej z instancji danych wejściowych.

Na podstawie tabeli 6.32 można zauważyć istnienie relacji zachądzących w zależności od konkretnych instancji danych wejściowych. Macierze opisujące ten sam rodzaj rzeczywistego problemu osiągają podobne do siebie wyniki, tj. np Esc161, Esc16h oraz Esc16j. Macierze o większym rozmiarze charakteryzują się wiekszą wartością błędu względnego oraz odchylenia standardowego, co można zauważyć np. na podstawie instancji Had14 oraz Had20 czy też Chr15a, Chr18b oraz Chr22a. Analizując działanie przypadków testowych o rozmiarze większym od 20 można było dostrzec trudności w dochodzeniu algorytmu do rozwiązania najmniejszego zgodnie z biblioteką QAPLIB. Interesującym wydaje się być fakt,

Tabela 6.32. Zestawienie działania algorytmu dla poszczególnych instancji danych wejściowych.

Instancja testowa	QAPLIB	Średni błąd względny funkcji celu	Odchylenie standardowe błędu względnego funkcji celu
Els19	17212548	0,3%	0,41%
Had14	2724	0,007%	0,03%
Had20	6922	0,07%	0,08%
Chr15a	9896	0,29%	0,32%
Chr18b	1534	0,32%	0,37%
Chr22a	6156	5,13%	1,41%
Esc16a	68	0%	0%
Esc16h	996	0%	0%
Esc16j	8	0%	0%
Esc32d	200	2,18%	1,41%
Nug21	2438	0,99%	0,46%
Nug25	3744	2,21%	0,62%
Bur26a	5426670	Osiągnięto wartości mniejsze niż QAPLIB	Osiągnięto wartości mniejsze niż QAPLIB
Bur26f	3782044	Osiągnięto wartości mniejsze niż QAPLIB	Osiągnięto wartości mniejsze niż QAPLIB
Tai50a	4938796	5,25%	4,17%

że wyniki osiągnięte dla instancji Bur26a oraz Bur26f, okazały się być lepsze od wartości znajdującej się w bibliotece QAPLIB. Były on lepsze o około 2,76 %, co może świadczyć o zbieganiu testowanego algorytmu do innego minimum lokalnego niż algorytm na podstawie którego obliczono wartość zawartą w bibliotece QAPLIB.

7. Podsumowanie pracy i wnioski końcowe

Algorytm ewolucji różnicowej należący do grona algorytmów metaheurystycznych pozwala na znalezienie pewnego rodzaju rozwiązania dla problemów NP-trudnych. Nierzadko jednak lepszą skutecznością wykazują się jego zmodyfikowane wersje, uwzględniające pewne charakterystyczne cechy rozwiązywanego problemu, a więc mogące się dostosować do wymaganych potrzeb. W ten sposób tworzone są wersję algorytmów dostosowane do rozwiązywania konkretnie zdefiniowanego problemu, lecz niekoniecznie wykazujące dużą skuteczność w przypadku innego rodzaju rozważanego zagadnienia. Proces doboru odpowiednich parametrów czy też wersji poszczególnych etapów algorytmu jest zagadnieniem bardzo złożonym. Z tego względu dużą rolę odgrywają czynniki losowe czy też decyzje o charakterze intuicyjnym.

Zagadnieniem będącym problemem NP-trudnym w kontekście którego możliwe staje się zastosowanie oraz przetestowanie skuteczności działania algorytmu ewolucji różnicowej jest miedzy innymi kwadratowe zagadnienie przydziału (QAP). To w kontekście tego zagadnienia w niniejszej pracy został opracowany zmodyfikowany algorytm ewolucji różnicowej wykazujący się dużą skutecznością działania. W wyniku działania algorytmu uzyskiwane są informacje o całkowitym koszcie działania systemu oraz informacje na temat rozlokowania poszczególnych budynków w konkretnych lokalizacjach.

W pracy udało się zrealizować założone cele, a więc znaleźć wersję zmodyfikowanego algorytmu dostosowanego do rozwiązywania problemu kwadratowego zagadnienia przydziału. O skuteczności zmodyfikowanego algorytmu świadczą jego wyniki testów w konfrontacji z wynikami osiąganymi przez klasyczną postać algorytmu zakładającą metodę reprodukcji losowej, mutacje DE/rand/1 oraz krzyżowanie dwumianowe. Klasyczna wersja uzyskuje średnią wartość błędu względnego funkcji celu równą 1,14% (tabela 6.15), z kolei najmniejszą wartością uzyskiwaną przez zmodyfikowany algorytm, wykorzystujący w swoim działaniu reprodukcję losową, mutacje DE/current to best/n_v + 1+$\diagup$ oraz krzyżowanie dwumianowe, dla którego parametr C_r jest równy 0,25, jest wartość rzędu 0,05% (tabela 6.27). Różnica jest zatem znacząca i wynosi aż 1,09%.

Badania rozpoczęto od metod reprodukcji, opisanych szczegółowo w rozdziale 3, w celu wyłonienia spośród nich metody działającej najkorzystniej. Na tym etapie badań największą skutecznością wykazała się metoda reprodukcji rankingowej, a średnia wartość błędu względnego wynosiła 0,57%. Pomimo

przewagi działania tej metody nad podstawową wersją stosowaną w algorytmach ewolucji różnicowej, a więc reprodukcja losowa, uzyskany wyniki nie był w pełni satysfakcjonujący. Zauważalna stała się także przewaga działania algorytmów z użyciem metod reprodukcji, w których istniał warunek co do niepowtarzalności się osobników wchodzących w skład grupy rodzicielskiej. Pominięcie tego warunku powoduje, że położenie osobnika jest niezależne od położenia jego rodzica [12]. Metody które nie spełniały postawionego założenia osiągały o minimum 1% gorszy wynik średniej wartości błędu względnego. Założenie to nie jest jednoznacznie określone w analizach teoretycznych, zastosowanie go wprowadza, więc pewnego rodzaju modyfikacje w stosunku do klasycznej wersji algorytmu. Metoda ruletki oraz turniejowa odznaczały się przedwczesną zbieżnością algorytmu, co oznacza, że poprawa wartości funkcji celu następowała jedynie w początkowych iteracjach działania algorytmu i na stałym poziomie utrzymywała się już do końca. Takie zachowanie algorytmu pod wpływem użycia tych metod spowodowane było tym, że najlepsze, ale jeszcze nie optymalne osobniki dominowały w populacji, co wykluczało możliwość powstawania w kolejnych iteracjach różniących się od siebie osobników potomnych.

Kolejną część testów stanowiły eksperymenty wykonywane na metodach mutacji. Części tej została poświęcona szczególna uwaga, jako że operacja mutacji jest najistotniejszym i najbardziej wpływowym elementem algorytmu. Dodatkowo metody zostały przetestowane pod kątem wpływu na nie współczynnika $\diagup$. Testy wykazały ewidentną przewagę metod uwzględniających w swoim działaniu właśnie ten dodatkowy czynnik skalujący $\diagup$ nad metodami nie zakładającymi zastosowania tej modyfikacji (tabela 6.8 oraz 6.12). Badania zostały podzielone na dwie zasadnicze części. Pierwsza z nich miała na celu dokonanie badań podczas, gdy metodą reprodukcji była metoda losowa, druga część zaś zakładała użycie metody rankingowej. Zarówno w pierwszej, jak i w drugiej części testowej najskuteczniejszym działaniem odznacza się metoda mutacji DE/current to best/n_v + 1+$\diagup$, gdzie liczba wektorów różnicowych n_v jest równa 3. Z przewagą 0,8% na tym etapie prowadzenia testów lepsze wyniki uzyskała wersja algorytmu zakładająca użycie metody losowej.

Ostatnią testowaną operacją genetyczną była operacja krzyżowania. Na etapie prowadzenia testów krzyżowania istniały już pewne wersje algorytmu, które odznaczały się lepszym działaniem od pozostałych. Z tego względu oprócz przeprowadzenia badań klasycznej wersji algorytmu, zostały przeprowadzone również badania z uwzględnieniem modyfikacji metod reprodukcji oraz mutacji. Najskuteczniejszą wersją okazał się być algorytm stosujący metodę reprodukcji losowej, mutacji DE/current to best/n_v + 1+$\diagup$, a więc z uwzględnieniem działania mnożnika $\diagup$ oraz krzyżowania OX. Strategia ta osiągnęła średnią wartość błędu względnego o wartości 0.06 % (tabela 6.20). Niemniej jednak strategia zakładająca użycie metody rankingowej również osiąga dobre wyniki, tj. wartość 0.1%. Różnica pomiędzy uzyskanymi wynikami jest równa 0,04 %, co jest wartością niewielką i może wynikać jedynie z działania czynnika losowego, który w przypadku algorytmów metaheurystycznych

ma duże znaczenie.

Został poddany rozważaniom fakt, iż metody mutacji zawierają współczynnik F skalujący wektor różnicowy, który w klasycznej wersji algorytmu ewolucji różnicowej przyjmuje proponowaną w literaturze wartości $F = 0.8$. Wartości współczynnika były zmieniane zgodnie z wartością obliczaną na podstawie funkcji opisanej szczegółowo w podrozdziale 4.3.1, a także na podstawie wartości określanych w każdej iteracji przez rozkład normalny. Interesująca i uzyskująca lepsze wyniki od wersji, w której wartość współczynnika F była ustalona na stałym poziomie $F = 0.8$, okazała się być strategia stosująca dynamiczną zmianę parametrów zgodnie z funkcją określoną w podrozdziale 4.3.1 (tabela 6.25). Dodatkowo w pracy przeprowadzono testy w celu znalezienia stałej wartości współczynnika F, która dla rozważanego problemu będzie przynosiła bardziej satysfakcjonujące efekty od algorytmu z wartością tą ustaloną na poziomie $F = 0.8$. Testy wykazały, iż dużą skutecznością, tj. średnią wartością błędu względnego równą 0,04%, wykazała się strategia zakładająca przypisanie do czynnika skalującego F, wartości 0,25. W przypadku współczynnika mutacji nie została wykazana żadna zależność wartości tego współczynnika od wartości funkcji celu. Na tej podstawie można wysnuć wniosek, iż wartość ta jest zależna bezpośrednio od rozwiązywanego problemu i dla różnych instancji danych wejściowych może przynosić odmienne efekty. Podobne testy zostały przeprowadzone dla metod krzyżowania, a ściślej mówiąc dla metody krzyżowania dwumianowego, gdyż tylko ta metoda zawierała współczynnik C_r, który w klasycznej wersji jest wartością zmiennoprzecinkową losowaną w każdej iteracji algorytmu w przedziale (0,1). Pomimo iż dynamiczna zmiana wartości współczynnika zgodnie z funkcją 5.3.1, czy też 5.3.2 pozwoliła na osiągnięcie lepszych wyników (odpowiednio 0,1 % oraz 0,14%), wyniki te nadal nie były na tyle dobre, by móc dorównać strategiom algorytmu opisanym powyżej. Zaskakująco dobrym działaniem i jednym z lepszych wyników osiągniętych wśród wszystkich przeprowadzonych testów, wykazała się strategia, która zakładała użycie reprodukcji losowej, mutacji DE/current to best/n_v + 1+$\diagdown$ oraz krzyżowania dwumianowego ze stałą wartością współczynnika F ustaloną na poziomie równym 0,25, gdyż wynik średniej wartości błędu względnego funkcji celu był równy 0,05%. Dodatkowo zauważyć można było, iż zależność współczynnika C_r oraz wartości funkcji celu była w przybliżeniu eksponencjalna. Wprowadzenie dynamicznej zmiany parametrów, zarówno w operacji mutacji, jak i krzyżowania, pozwala uniknąć trudnego etapu strojenia algorytmu. W literaturze [10], [12] podane są optymalne wartości tych współczynników, należy jednak pamiętać, że najczęściej zbiorem testowym jest standardowy zestaw funkcji opisujących problemy optymalizacyjne. W przypadku bardziej złożonych zagadnień najczęściej wymagane jest ustalenie innych wartości parametrów.

Po przeprowadzeniu testów w obrębie metod reprodukcji, mutacji oraz krzyżowania możliwe było wyodrębnienie kilku wersji algorytmu, które wykazały się największą skutecznością. Są to odpowiednio:

1. reprodukcja losowa, mutacja DE/current to best/n_v + 1+$\diagup$, krzyżowanie dwumianowe, C_r = 0.25, F = 0.8
2. reprodukcja rankingowa, mutacja DE/current to best/n_v + 1+$\diagup$, krzyżowanie OX, F zgodnie z funkcją 4.3.1,
3. reprodukcja losowa, mutacja DE/current to best/n_v + 1+$\diagup$, krzyżowanie OX, F = 0.25,
4. reprodukcja losowa, mutacja DE/current to best/n_v + 1+$\diagup$, krzyżowanie OX, F = 0.8,
5. reprodukcja losowa, mutacja DE/current to best/n_v + 1+$\diagup$, krzyżowanie OX, F zgodnie z funkcją 4.3.1,

Dalsza część testów została przeprowadzona dla macierzy o różnych rozmiarach z wykorzystaniem 1 wersji algorytmu. Na podstawie uzyskanych wyników można było wysnuć wniosek, iż im większy rozmiar macierzy, tym trudniej o uzyskanie dobrego wyniku. Skuteczność działania algorytmu po przekroczeniu rozmiaru macierzy równego 20 znacznie spada. Odpowiedni dobór liczby iteracji jest również kwestią indywidualną każdej z rozważanych instancji testowych, a najlepszym ze sposobów jej doboru są metody doświadczalne. Wielkość funkcji celu do której dąży algorytm jest również czynnikiem w pewnym stopniu wpływającym na skuteczność działania algorytmu.

Uzyskanie różniących się od siebie wartości błędu względnego dla macierzy o tym samym rozmiarze wynika nie tylko ze specyfiki konkretnych danych wejściowych, ale jest spowodowane również faktem, iż algorytm ewolucji różnicowej jest algorytmem opartym w dużej mierze na czynniku losowym, którego działania nie da się zagwarantować czy też przewidzieć. Podsumowując, można stwierdzić, iż jakość wyników otrzymywanych za pomocą zmodyfikowanego algorytmu ewolucji różnicowej zależy od rozmiaru macierzy testowej, wartości funkcji celu, do której dąży algorytm, wielkości populacji, czasu przeznaczonego na poszukiwanie rozwiązania oraz doboru sposobu reprodukcji, a także zastosowanych operatorów mutacji i krzyżowania.

Z powodu mnogości dostępnych wersji metod i możliwości modyfikacji algorytmu, w niniejszej pracy badaniom poddano jedynie niektóre z nich. Przeprowadzona analiza w zakresie dostosowania algorytmu ewolucji różnicowej w celu rozwiązania kwadratowego zagadnienia przydziału stanowią dobry punkt wyjścia do dalszych rozważań w tej dziedzinie. Dodatkowe badania w kierunku adaptacji parametrów mogłyby skutkować ograniczeniem wpływu na działanie algorytmu czynnika ludzkiego.

A. Fragmenty implementacji

Metody reprodukcji

Klasyczna reprodukcja : losowy wybór

```
1   public List<int> RandomWitoutRestrictions()
2   {
3       List<int> WhichIndividuals = new List<int>();
4       int number = 3;
5       var rand = new System.Random();
6
7       while (number != 0)
8       {
9           WhichIndividuals.Add(Kit.GiveRandomNumber(WhichIndividuals, Population.Count
                - 1, rand));
10          number--;
11      }
12      return WhichIndividuals;
13  }
```

Metoda rankingowa

```
1   public List<int> RankingMethond()
2   {
3       List<int> WhichIndividuals = new List<int>();
4       List<int> WhichIndividualsIndex = new List<int>();
5       List<int> WhichIndividualBeforeSelection = new List<int>();
6       List<int> ResultsOfObjectiveFunction =
            inputdata.ObjectiveFunctionVector(Population);
7
8       var rand = new System.Random();
9       int size_tmp = size;
10      int number = 3;
11
12      while (size_tmp != 0)
13      {
14          var index =
                ResultsOfObjectiveFunction.IndexOf(ResultsOfObjectiveFunction.Min());
15          ResultsOfObjectiveFunction[index] = Int32.MaxValue;
16
17          int numberOfIter = size_tmp;
18          while (numberOfIter != 0)
```

```
19                  {
20                      WhichIndividualBeforeSelection.Add(index);
21                      numberOfIter--;
22                  }
23                  size_tmp--;
24              }
25
26              while (number != 0)
27              {
28                  var tmp = Kit.GiveRandomNumber(WhichIndividualsIndex,
                        WhichIndividualBeforeSelection.Count - 1, rand);
29                  for (int i~= 0; i~< WhichIndividualBeforeSelection.Count; i++) // to~w~celu
                        uniknięcia powtórzeń w~wektorach
30                  {
31                      if (WhichIndividualBeforeSelection[i] ==
                            WhichIndividualBeforeSelection[tmp])
32                          WhichIndividualsIndex.Add(i);
33                  }
34
35                  WhichIndividuals.Add(WhichIndividualBeforeSelection[tmp]);
36                  number--;
37              }
38              return WhichIndividuals;
39          }
```

Metoda ruletki

```
1           public List<int> RouletteMethond()
2           {
3               List<int> WhichIndividuals = new List<int>();
4               List<int> ResultsOfObjectiveFunction =
                    inputdata.ObjectiveFunctionVector(Population);
5               List<Tuple<int, int>> tuple = new List<Tuple<int, int>>();
6               int range_begin = 0;
7               int range_end = 0;
8               var rand = new System.Random();
9               int number = 3;
10
11              foreach (var elem in ResultsOfObjectiveFunction)
12              {
13                  range_end += elem;
14                  tuple.Add(new Tuple<int, int>(range_begin, range_end));
15                  range_begin = elem + 1;
16              }
17
18              while (number != 0)
19              {
20                  var tmp = Kit.GiveRandomNumber(WhichIndividuals,
                        ResultsOfObjectiveFunction.Sum(), rand);
21                  for (int i~= 0; i~< tuple.Count; i++)
22                  {
23                      if (tuple[i].Item2 > tmp)
24                      {
25                          WhichIndividuals.Add(i);
```

```
                        number--;
                        break;
                    }
                }
            }
            return WhichIndividuals;
        }
```

Metoda turniejowa

```
public List<int> TournamentMethond()
        {
            List<int> WhichIndividuals = new List<int>();
            List<int> ResultsOfObjectiveFunction =
                inputdata.ObjectiveFunctionVector(Population);
            List<int> ResultsOfObjectiveFunction_const = new List<int>();
            ResultsOfObjectiveFunction_const.AddRange(ResultsOfObjectiveFunction);
            List<int> TournamentTable = new List<int>();
            int number = 3;

            while (number != 0)
            {
                int sizeOfTournament_const = ResultsOfObjectiveFunction.Count;
                while (sizeOfTournament_const > 1)
                {
                    for (int i~= 0; i~< sizeOfTournament_const; i++)
                    {
                        TournamentTable.Add(Math.Min(ResultsOfObjectiveFunction[i],
                            ResultsOfObjectiveFunction[i + 1]));
                        i++;

                        if ((sizeOfTournament_const/2) == 1 && i~== sizeOfTournament_const -
                            2)
                        {
                            TournamentTable.Add(ResultsOfObjectiveFunction.Last());
                            i++;
                        }
                    }

                    if (TournamentTable.Count == 1)
                        break;
                    sizeOfTournament_const = TournamentTable.Count;
                    ResultsOfObjectiveFunction.RemoveAll(elem =>
                        ResultsOfObjectiveFunction.Contains(elem));
                    ResultsOfObjectiveFunction.AddRange(TournamentTable);
                    TournamentTable.RemoveAll(elem => TournamentTable.Contains(elem));
                }

                var index = ResultsOfObjectiveFunction_const.FindIndex(value => value ==
                    TournamentTable[0]);
                WhichIndividuals.Add(index);
                ResultsOfObjectiveFunction_const[index] = Int32.MaxValue;
                ResultsOfObjectiveFunction.RemoveAll(elem =>
                    ResultsOfObjectiveFunction.Contains(elem));
```

```
            ResultsOfObjectiveFunction.AddRange(ResultsOfObjectiveFunction_const);
            TournamentTable.RemoveAt(0);
            number--;
        }
        return WhichIndividuals;
    }
```

Metoda elitarna

```
    public List<int> ElitistMethond()
    {
        List<int> WhichIndividuals = new List<int>();
        List<int> ResultsOfObjectiveFunction =
            inputdata.ObjectiveFunctionVector(Population);
        int number = 3;

        while (number != 0)
        {
            int index = ResultsOfObjectiveFunction.FindIndex(min => min ==
                ResultsOfObjectiveFunction.Min());
            WhichIndividuals.Add(index);
            ResultsOfObjectiveFunction[index] = Int32.MaxValue;
            number--;
        }

        return WhichIndividuals;
    }
```

Metody mutacji

Klasyczna mutacja: DE/rand/1

```
    public List<int> ToMutate()
    {
        List<int> MutatedIndividual = new List<int>(new int[Population.Count]);
        List<double> MutatedIndividual_tmp = new List<double>();
        List<int> Randoms = RandomWitoutRestrictions();
        double DefaultValueForMut = 0.8;

        for (int j = 0; j < Population.Count; j++)
        {
            double value = Population[Randoms[0]][j] + DefaultValueForMut *
                (Population[Randoms[1]][j] - Population[Randoms[2]][j]);
            MutatedIndividual_tmp.Add(value);
        }

        Normalize(MutatedIndividual_tmp, MutatedIndividual);
        return MutatedIndividual;
    }
```

Strategia II: DE/best/1

```
    public List<int> ToMutate2()
```

```
        {
            List<int> MutatedIndividual = new List<int>(new int[Population.Count]);
            List<double> MutatedIndividual_tmp = new List<double>();

            List<int> Best = ElitistMethond(1);
            List<int> Difference = RandomWitoutRestrictions(2);

            double DefaultValueForMut = 0.8;

            for (int j = 0; j < Population.Count; j++)
            {
                double value = Population[Best[0]][j] + DefaultValueForMut *
                    (Population[Difference[0]][j] - Population[Difference[1]][j]);
                MutatedIndividual_tmp.Add(value);
            }

            Normalize(MutatedIndividual_tmp, MutatedIndividual);
            return MutatedIndividual;
        }
```

Strategia III: DE/rand/n_v

```
public List<int> ToMutate3(int number_of_vectors)
        {
            List<int> MutatedIndividual = new List<int>(new int[Population.Count]);
            List<double> MutatedIndividual_tmp = new List<double>();
            int n = 2 * number_of_vectors + 1;

            if (n > Population.Count)
            {
                number_of_vectors = (Population.Count - 1) / 2;
            }

            List<int> Randoms = RandomWitoutRestrictions(3);
            double DefaultValueForMut = 0.8;
            for (int j = 0; j < Population.Count; j++)
            {
                int difference = 0;
                int number = number_of_vectors;
                while (number != 0)
                {
                    difference += (Population[Randoms[1]][j] - Population[Randoms[2]][j]);
                    number--;
                }

                double value = Population[Randoms[0]][j] + DefaultValueForMut * difference;
                MutatedIndividual_tmp.Add(value);
            }
            Normalize(MutatedIndividual_tmp, MutatedIndividual);
            return MutatedIndividual;
        }
```

Strategia IV: DE/current to best/$n_v + 1$

```
public List<int> ToMutate4(int number_of_vectors, int iter)
        {
            List<int> MutatedIndividual = new List<int>(new int[Population.Count]);
            List<double> MutatedIndividual_tmp = new List<double>();
            int n = 2 * number_of_vectors + 1;

            if (n > Population.Count)
            {
                number_of_vectors = (Population.Count - 1) / 2;
            }

            List<int> Best = ElitistMethond(1);
            List<int> Randoms = RandomWitoutRestrictions(3);
            double DefaultValueForMut = 0.8;
            for (int j = 0; j < Population.Count; j++)
            {
                int difference = 0;
                int number = number_of_vectors;
                while (number != 0)
                {
                    difference += (Population[Randoms[1]][j] - Population[Randoms[2]][j]);
                    number--;
                }

                double value = Population[iter][j] + DefaultValueForMut *
                    (Population[Best[0]][j] - Population[Randoms[0]][j]) + DefaultValueForMut
                    * difference;
                MutatedIndividual_tmp.Add(value);
            }
            Normalize(MutatedIndividual_tmp, MutatedIndividual);
            return MutatedIndividual;
        }
```

Metody krzyżowania

Klasyczne krzyżowanie : krzyżowanie dwumianowe

```
        public List<int> BinomialCrossver(List<int> ParentIndividual, List<int>
            MutatedIndividual)
        {
            List<int> CrossedIndividual = new List<int>();
            Random random = new Random();
            double Cr = random.NextDouble();
            Random random2 = new Random();

            int size = ParentIndividual.Count;
            for (int i~= 0; i~< size; i++){
                double Cr_tmp = random.NextDouble();

                if (Cr_tmp < Cr){
                    if (CrossedIndividual.Contains(MutatedIndividual[i])){
                        var ind = GiveRandomNumber(CrossedIndividual,
                            MutatedIndividual.Count, random2);
```

```
                    CrossedIndividual.Add(ind);
                }
                else
                    CrossedIndividual.Add(MutatedIndividual[i]);
            }
            else //Parent{
                if (CrossedIndividual.Contains(ParentIndividual[i])){
                    var ind = GiveRandomNumber(CrossedIndividual, ParentIndividual.Count,
                        random2);
                    CrossedIndividual.Add(ind);
                }
                else
                    CrossedIndividual.Add(ParentIndividual[i]);
            }
        }
        return CrossedIndividual;
    }
```

Krzyżowanie OX

```
    public List<int> OXCrossver(List<int> ParentIndividual, List<int> MutatedIndividual)
    {
        List<int> CrossedIndividual = new List<int>();
        for (int i~= 0; i~< ParentIndividual.Count; i++){
            CrossedIndividual.Add(0);
        }

        Random rnd = new Random();
        int offset = rnd.Next(0, ParentIndividual.Count);
        int length = rnd.Next(1, ParentIndividual.Count - offset);
        int iter = 0;
        while (iter != length){
            CrossedIndividual[offset + iter] = MutatedIndividual[offset + iter];
            iter++;
        }

        for (int i~= 0; i~< ParentIndividual.Count; i++){
            if (CrossedIndividual[i] == 0){
                var i_tmp = 0;
                while(CrossedIndividual.Contains(ParentIndividual[i_tmp])){
                    i_tmp++;
                }
                CrossedIndividual[i] = ParentIndividual[i_tmp];
            }
        }
        return CrossedIndividual;
    }
```

Krzyżowanie CX

```
    public List<int> CXCrossver(List<int> ParentIndividual, List<int> MutatedIndividual)
    {
        List<int> CrossedIndividual = new List<int>();
        for (int i~= 0; i~< ParentIndividual.Count; i++){
```

```
            CrossedIndividual.Add(0);
        }

        CrossedIndividual[0] = MutatedIndividual[0];
        var index = 0;

        while (ParentIndividual[index] != MutatedIndividual[0]){
            index = MutatedIndividual.FindIndex(x => x == ParentIndividual[index]);
            CrossedIndividual[index] = MutatedIndividual[index];
        }

        for(int i~= 0; i~< ParentIndividual.Count; i++){
            if (CrossedIndividual[i] == 0){
                var i_tmp = 0;
                while (CrossedIndividual.Contains(ParentIndividual[i_tmp])){
                    i_tmp++;
                }
                CrossedIndividual[i] = ParentIndividual[i_tmp];
            }
        }
        return CrossedIndividual;
    }
```

Krzyżowanie PMX

```
    public List<int> PMXCrossver(List<int> ParentIndividual, List<int> MutatedIndividual){
        List<int> MutatedIndividualSubstringCopy = new List<int>();
        List<int> ParentIndividualSubstringCopy = new List<int>();
        List<int> MutatedIndividualSubstring = new List<int>();
        List<int> ParentIndividualSubstring = new List<int>();
        Random rnd = new Random();
        int offset = rnd.Next(0, ParentIndividual.Count);
        int length = rnd.Next(1, ParentIndividual.Count - offset);
        int iter = 0;

        while (iter != length){
            MutatedIndividualSubstring.Add(MutatedIndividual[offset + iter]);
            ParentIndividualSubstring.Add(ParentIndividual[offset + iter]);
            iter++;}

        for (int i~= offset; i~< offset + MutatedIndividualSubstring.Count; i++){
            MutatedIndividual[i] = 0;
            ParentIndividual[i] = 0;}

        for (int i~= 0; i~< MutatedIndividualSubstring.Count; i++){
            MutatedIndividualSubstringCopy.Add(MutatedIndividualSubstring[i]);
            ParentIndividualSubstringCopy.Add(ParentIndividualSubstring[i]);}

        for (int i~= 0; i~< MutatedIndividualSubstring.Count; i++){
            if (ParentIndividualSubstring.Contains(MutatedIndividualSubstring[i])){
                var index = ParentIndividualSubstring.FindIndex(x => x ==
                    MutatedIndividualSubstring[i]);
                MutatedIndividualSubstring[i] = MutatedIndividualSubstring[index];
                MutatedIndividualSubstring.RemoveAt(index);
```

```
29                  ParentIndividualSubstring.RemoveAt(index);
30                  i-= -1;}
31              }
32              for (int i-= 0; i-< MutatedIndividual.Count; i++){
33                  if (!MutatedIndividual.Contains(ParentIndividual[i])){
34                      int  index_2 = MutatedIndividualSubstring.FindIndex(x => x ==
                            ParentIndividual[i]);
35                      var value = ParentIndividualSubstring[index_2];
36                      var index = MutatedIndividual.FindIndex(x => x == value);
37                      MutatedIndividual[index] = ParentIndividual[i];
38                      ParentIndividual[i] = value;}
39              }
40              var i_tmp = 0;
41              var i_tmp2 = 0;
42
43              for (int i-= 0; i-< MutatedIndividual.Count; i++){
44                  if (MutatedIndividual[i] == 0){
45                      MutatedIndividual[i] = ParentIndividualSubstringCopy[i_tmp];
46                      i_tmp ++;}
47                  if (ParentIndividual[i] == 0){
48                      ParentIndividual[i] = MutatedIndividualSubstringCopy[i_tmp2];
49                      i_tmp2 ++;}
50              }
51              return MutatedIndividual;
52          }
```

Bibliografia

[1] Michał Bereta i Paweł Jarosz: „Algorytmy genetyczne”, w: *Politechnika Krakowska*,

[2] Pablo Rodriguez-Mier: „A tutorial on Differential Evolution with Python”, w: 2017

[3] Kilic Haydar: „Tournament selection based antlion optimization algorithm for solving quadratic assignment problem”, w: 2018

[4] Wojciech Chmiel: „Matematyczne Metody Wspomagania Decyzji - Wykłady”, w: nieokreślone

[5] Dariusz Banasiak: „Algorytmy genetyczne”, w: 2009

[6] Politechnika Gdańska: „Uczenie maszynowe - algorytmy genetyczne”, w: 2009

[7] Ewa Figielska: „Zeszyty naukowe 81-92”, w: 2011,
`https://www.wt.pw.edu.pl/Badania-i-nauka/Prace-Naukowe-Politechniki-Warszawskiej-Transport/Zeszyty/(offset)/30`

[8] Gracjan Wilczewski: „Algorytmy ewolucyjne - wprowadzenie”, w: 2005,
`http://zasada.zut.edu.pl/ag/Wilczewski`

[9] Miguel Anjos: „Quadratic Assignment Problem Library”, w: 1997,
`http://coral.ise.lehigh.edu/data-sets/qaplib/`

[10] Przemysław Juszczuk: „Adaptacyjny algorytm ewolucji różnicowej w rozwiązywaniu problemów teorii gier”, w: *Uniwersytet Śląski*, 2013, s. 6–62

[11] Hossein Sharifi Noghabi: „Differential Evolution with Generalized Mutation Operator for Parameters Optimization in Gene Selection for Cancer Classification”, w: 1997,
`https://arxiv.org/ftp/arxiv/papers/1510/1510.02516`

[12] Karol Opara: „Analiza algorytmu ewolucji różnicowej i jego zastosowanie w wyznaczaniu zależności statystycznych”, w: *Uniwersytet Śląski*, 2014, s. 6–62

[13] Ali Wagdy Mohamed: „Differential Evolution with Novel Mutation and Adaptive Crossover Strategies for Solving Large Scale Global Optimization Problems”, w: 1997,
`https://www.hindawi.com/journals/acisc/2017/7974218/`

[14] U. Boryczka i P. Juszczuk: „A new evolutionary method for generating nash equilibria in bimatrix games with known support.”, w: *Central European Journal of Computer Science*,

[15] Author unknown: „Genetic Algorithms”, w: unknown,
`http://www.rubicite.com/Tutorials/GeneticAlgorithms/CrossoverOperators/Order1CrossoverOperator.aspx`

[16] Author unknown: „Genetic Algorithms”, w: unknown,
`http://mat.uab.cat/ alseda/MasterOpt/GeneticOperations`

[17] C.S. Chang: „Differential evolution based tuning of fuzzy automatic train operation for mass rapid transit system.”, w: *IEE Proceedings of Electric Power Applications*, 2000, s. 6–62