

Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki
Kierunek Informatyka
Studia stacjonarne II stopnia

Praca dyplomowa magisterska

**Analiza porównawcza algorytmów
metaheurystycznych do
rozwiązywania wybranego problemu
optymalizacyjnego**

Kierujący projektem:
dr inż. Henryk Josiński

Autor:
Sebastian Nalepka

Gliwice 2017

Spis treści

| | |
|--|-----------|
| Wstęp | 3 |
| 1. Wybór badanego problemu optymalizacji | 5 |
| 1.1. Opis minimalizacji funkcji ciągłych wielu zmiennych | 5 |
| 1.2. Funkcje testowe | 5 |
| 1.2.1. Funkcja Bohachevsky'ego I | 6 |
| 1.2.2. Funkcja Colville'a | 7 |
| 1.2.3. Funkcja Ackley'a | 7 |
| 1.2.4. Funkcja Rosenbrocka | 8 |
| 1.2.5. Funkcja Eggholdera | 9 |
| 1.2.6. Funkcja Griewanka | 9 |
| 2. Metody rozwiązania wybranego problemu optymalizacji | 11 |
| 2.1. Algorytmy dokładne | 11 |
| 2.2. Metaheurystyki optymalizacyjne | 12 |
| 2.2.1. Metoda optymalizacji rojem cząstek | 12 |
| 2.2.2. Symulowane wyżarzanie | 14 |
| 2.2.3. Algorytm genetyczny | 15 |
| 3. Automatyzacja przeprowadzanych badań | 19 |
| 4. Wykorzystane rozwiązania technologiczne | 21 |
| 4.1. Zastosowane technologie | 21 |
| 4.1.1. .NET Framework/C# | 21 |
| 4.1.2. WPF / XAML | 22 |
| 4.1.3. TSQL | 22 |
| 4.2. Zastosowane narzędzia | 23 |
| 4.2.1. Matlab | 23 |
| 4.2.2. SQL Server Management Studio 17 | 23 |
| 4.2.3. Visual Studio 2015 | 24 |
| 4.2.4. Resharper | 25 |

| | |
|--|-----------|
| 4.3. Wykorzystane biblioteki zewnętrzne | 26 |
| 4.3.1. Matlab Application Type Library v.1.0 | 26 |
| 5. Architektura budowanej aplikacji | 27 |
| 5.1. Architektura aplikacji | 27 |
| 5.1.1. Wzorce architektoniczne oprogramowania | 27 |
| 5.1.2. Zastosowany wzorzec architektoniczny – MVVM | 28 |
| 5.1.3. Model aplikacji | 30 |
| 5.2. Architektura bazy danych | 34 |
| 5.2.1. Microsoft SQL Server – użyty system do zarządzania bazą danych | 34 |
| 5.2.2. Budowa bazy danych użytej w projekcie | 35 |
| 5.3. Komunikacja bazy danych z projektem programistycznym | 38 |
| 5.3.1. Mapowanie obiektowo-relacyjne | 38 |
| 5.3.2. Zastosowane narzędzie ORM – Entity Framework | 39 |
| 5.4. Komunikacja Matlaba z projektem programistycznym | 40 |
| 6. Badania eksperymentalne | 41 |
| 6.1. Metody porównawcze wybranych algorytmów | 41 |
| 6.2. Opis przeprowadzonych badań | 41 |
| 6.2.1. Skalowanie algorytmów | 41 |
| 6.2.2. Przeprowadzenie doświadczeń w zaimplementowanym środowisku testowym | 42 |
| 6.3. Wyniki doświadczeń dla zadanych funkcji testowych | 43 |
| 6.3.1. Funkcja Bohachevsky’ego 1 | 43 |
| 6.3.2. Funkcja Colville’a | 49 |
| 6.3.3. Funkcja Ackley’a | 55 |
| 6.3.4. Funkcja Rosenbrocka | 61 |
| 6.3.5. Funkcja Eggholdera | 67 |
| 6.3.6. Funkcja Griewanka | 73 |
| 6.4. Wnioski z przeprowadzonych doświadczeń | 79 |
| 7. Podsumowanie | 83 |
| Bibliografia | 85 |

Wstęp

W otaczającym nas świecie obecnych jest wiele problemów, z którymi zmagają się ludzie. Część z nich jest problemami prostymi, do rozwiązania których wystarczy wyłącznie nieduży nakład czasu. Istnieją jednak problemy trudniejsze, które wymagają długotrwałych przemyśleń i obliczeń pomimo których nie zawsze otrzymujemy najlepsze rozwiązanie. Do prostych problemów możemy zaliczyć codzienne decyzje podejmowane przez każdego człowieka. Dla przykładu, jeśli chcemy dojechać z punktu *A* do punktu *B* komunikacją miejską, wystarczające będzie sprawdzenie rozkładu jazdy i wybranie połączenia, które będzie pasowało nam w kontekście godziny przyjazdu na dane miejsce. Przedstawiony problem posiada inną skalę trudności ze strony przewoźnika. Wyznaczenie optymalnych tras przewozowych dla określonej liczby środków transportu, w celu obsłużenia danego zbioru klientów, którzy rozlokowani są w różnych punktach jest kwestią bardzo skomplikowaną. Znalezienie optymalnych tras, które umożliwią przetransportowanie jak największej ilości osób w celu zmaksymalizowania zysku, przy zachowaniu możliwie najkrótszych tras, których zamysłem jest minimalizowanie kosztów poniesionych z transportem jest złożonym problemem, znanym jako jedna z odmian problemu marszrutyzacji, który jest z kolei rozwinięciem bardzo popularnego problemu komiwojażera [1]. Problem ten polega na znalezieniu najkrótszej drogi łączącej wszystkie zdefiniowane uprzednio punkty, zaczynając i kończąc w tym samym miejscu. Problemy, do rozwiązania których potrzebne są ogromne nakłady obliczeniowe definiowane są jako problemy optymalizacyjne. W problemach takich liczba możliwych rozwiązań w przestrzeni poszukiwań z reguły jest tak duża, że niemożliwe jest w skończonym czasie zastosowanie przeszukiwania wyczerpującego w celu znalezienia najlepszego z nich.

Cel pracy

Celem pracy jest dokonanie analizy porównawczej algorytmu symulowanego wyzarzania, algorytmu genetycznego oraz metody roju cząstek za pomocą zaimplementowanej dedykowanej aplikacji bazodanowej, której przeznaczeniem jest zautomatyzowanie procesu szukania minimum globalnego dla zadanych funkcji testowych.

Zawartość pracy

Dla osiągnięcia wyżej wymienionego celu zrealizowana została praca składająca się ze wstępu, sześciu rozdziałów oraz podsumowania. W rozdziale pierwszym przedstawiony został wybór problemu optymalizacyjnego wraz z jego opisem oraz listą funkcji testowych, które użyte zostały do przeprowadzenia badań. Rozdział drugi przedstawia możliwe sposoby rozwiązania wybranego problemu optymalizacyjnego wraz z opisem użytych algorytmów metaheurystycznych. Na bazie dwóch pierwszych rozdziałów utworzony został w rozdziale trzecim projekt automatyzacji przeprowadzanych doświadczeń, na podstawie którego bazowała budowana aplikacja. Kolejny rozdział to zastosowane rozwiązania technologiczne oraz opis wykorzystanych technologii, bibliotek zewnętrznych, a także narzędzi. Na ich podstawie utworzona została architektura aplikacji bazodanowej, której specyfikacja umiejscowiona została w rozdziale piątym. Kolejny rozdział prezentuje już ściśle aspekt badawczy. Zawiera on opis metod porównawczych oraz przeprowadzonych badań, a także wyniki doświadczeń dla użytych funkcji testowych wraz z ich podsumowaniem.

1. Wybór badanego problemu optymalizacji

1.1. Opis minimalizacji funkcji ciągłych wielu zmiennych

W prezentowanej pracy dokonano porównania algorytmów metaheurystycznych odnosząc się do problemu minimalizacji funkcji wielu zmiennych. Problem ten polega na znalezieniu minimum globalnego rzeczywistej funkcji poprzez systematyczne wybieranie parametrów wejściowych z dozwolonego zakresu i obliczaniu ich wartości funkcji. Trudność problemu sprowadza się do wielkości przestrzeni przeszukiwania. Traktując problem jako czysto matematyczny, funkcja n zmiennych posiada nieskończenie wiele wartości w każdym wymiarze. Mamy więc nieskończenie wielką przestrzeń poszukiwań. Biorąc jednak pod względ aspekt technologiczny i to, iż komputery bazują na danych skończonych, można w prosty sposób przedstawić skalę trudności. W czasie implementacji algorytmu, którego celem jest znalezienie minimum globalnego funkcji, należy wziąć pod uwagę dostępną dokładność obliczeniową maszyny. Zakładając, iż dokładność ta wynosi osiem miejsc po przecinku, każda zmienna ograniczona w przedziale $[0, 100]$ może przyjąć $100 \cdot 10^8$ różnych wartości. Już dla funkcji dwóch zmiennych, wielkość przestrzeni przeszukiwania wynosi $(100 \cdot 10^8)^2 = 10^{20}$.

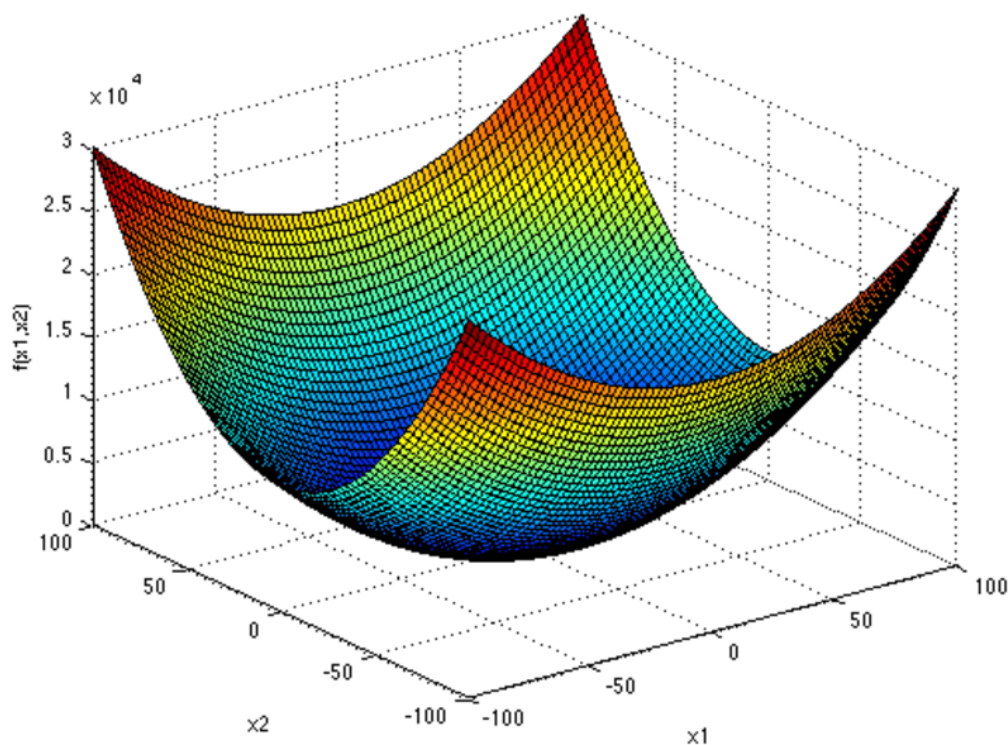
1.2. Funkcje testowe

Problem minimalizacji funkcji ciągłych bardzo dobrze nadaje się do porównania algorytmów metaheurystycznych przez wzgląd na powszechnie dostępne funkcje testowe. Funkcje te posiadają pewny specyficzny element, dzięki któremu możliwe jest porównanie wyników otrzymanych przez dany algorytm. Element ten to znajomość minimum globalnego dla danej funkcji testowej. Dzięki znajomości wartości najlepszej (najmniejszej) dla danej funkcji, wiadomo jak szybko oraz czy w ogóle badany algorytm znalazł najlepsze możliwe rozwiązanie. Posiadając ten punkt odniesienia można zestawiać otrzymane rezultaty wszystkich algorytmów pod kątem czasowym lub liczby wyliczeń wartości funkcji dla ustalonych przez algorytm punktów.

Do analizy wybranych zostało sześć funkcji testowych, których dobór brał pod uwagę stopień ich skomplikowania. Funkcje te zostały wybrane z listy zawartej w [2] uwzględniając różną liczbę wymiarów oraz skalę ich trudności. Każda z funkcji posiada specyficzne właściwości, które zostaną wzięte pod uwagę podczas porównania rezultatów algorytmów heurystycznych.

1.2.1. Funkcja Bohachevsky'ego I

Funkcja Bohachevsky'ego 1 jest stosunkowo prostą do minimalizacji funkcją dwóch zmiennych, której wykres przedstawiony na wykresie 1 przybiera kształt łuku. Z reguły jest ona badana w zakresie $x_i \in [-100, 100]$ dla $i = 1, 2$. W przedstawianej pracy magisterskiej granice to zostały jednak dwukrotnie powiększone i wynoszą $[-200, 200]$. Według doświadczeń przeprowadzonych w [2], 81.75% doświadczeń zakończonych zostało znalezieniem minimum globalnego, które dla funkcji Bohachevsky'ego I wynosi 0, dla punktu $x = (0, 0)$.



$$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$$

Rysunek 1: Wykres oraz wzór funkcji Bohachevsky'ego I dwóch zmiennych, Źródło: <https://www.sfu.ca/~ssurjano/boha.html>

1.2.2. Funkcja Colville'a

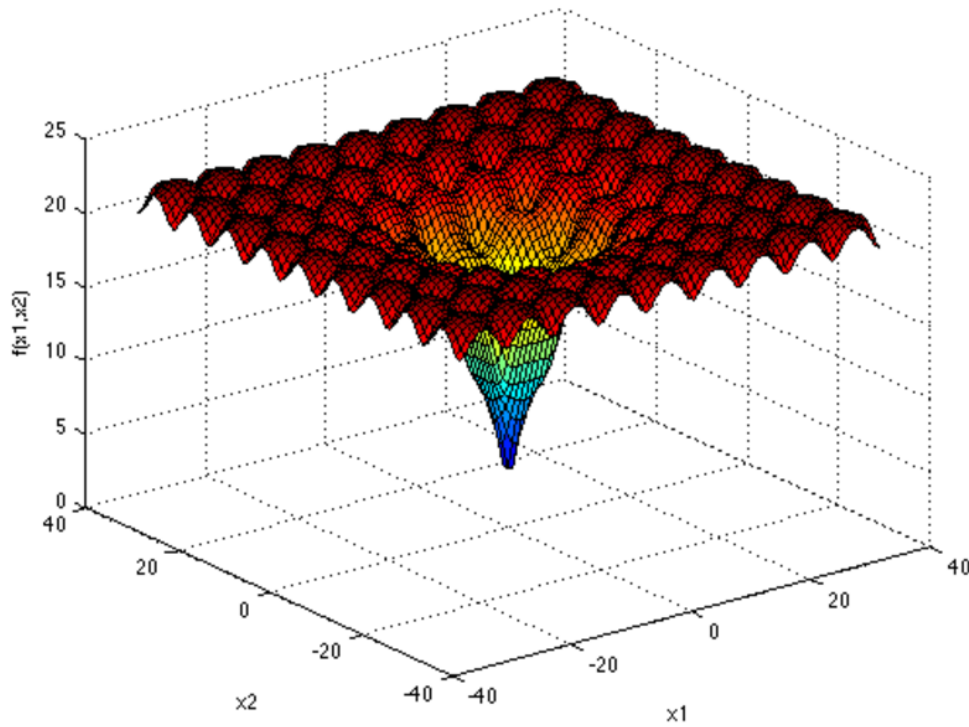
Jest to funkcja czterech zmiennych, która posiada jedno minimum globalne o wartości 0 dla punktu $x = (1, 1, 1, 1)$ oraz analizowana jest zazwyczaj na hipersześcianie $x_i \in [-200, 200]$ dla $i = 1, 2, 3, 4$. Według danych zawartych w [2], jest to funkcja trudniejsza w minimalizacji od funkcji Bohachevsky'ego I, której procentowy sukces dotyczący znalezienia minimum globalnego wyniósł 72.00%.

Wzór funkcji Colville'a:

$$f(x) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$$

1.2.3. Funkcja Ackley'a

Funkcja Ackley'a jest popularną funkcją dla testów optymalizacyjnych algorytmów. W swojej dwuwymiarowej postaci, którą ukazano na obrazku 2, charakteryzuje się stosunkowo płaskim obszarem zewnętrznym i dużym otworem w centrum. Funkcja stwarza ryzyko dla algorytmów optymalizacyjnych poprzez możliwość zatrzymania się w jednym z licznych lokalnych minimów. Istnieje możliwość rozszerzenia funkcji Ackley'a na wiele wymiarów co też uczyniono w prezentowanej pracy magisterskiej, w której rozpatrywana jest ona dla czterech zmiennych. Minimum globalne umiejscowione jest w punkcie $x = (0, 0, 0, 0)$ i jego wartość wynosi 0. Dziedzina funkcji analizowana będzie z kolei w przedziale $x_i \in [-150, 150]$ dla $i = 1, 2, 3, 4$. Dla wspomnianych już testów, których wyniki dostępne są do wglądu w odnośniku [2], procentowy sukces znalezienia minimum globalnego wyniósł 48.25%.



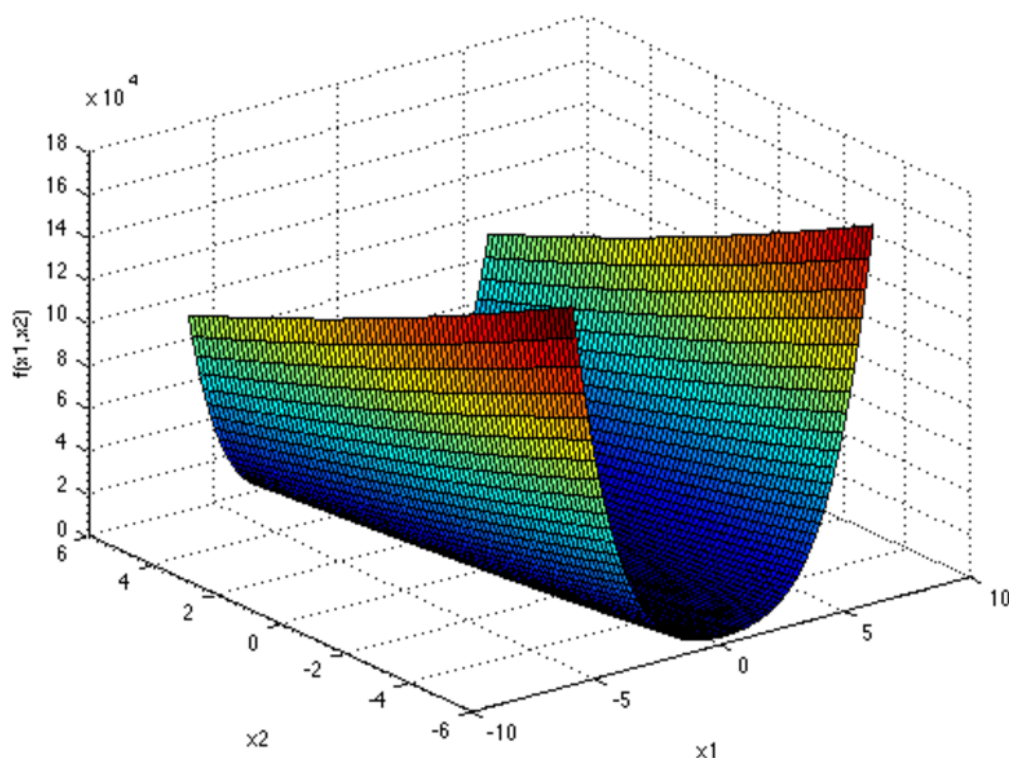
$$f(x) = -a \exp\left(-b \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2}\right) - \exp\left(\frac{1}{d} \sum_{i=1}^d \cos(cx_i)\right) + a + \exp(1)$$

$$a = 20, b = 0.2, c = 2\pi$$

Rysunek 2: Wykres oraz wzór funkcji Ackley'a dwóch zmiennych, Źródło: <https://www.sfu.ca/~ssurjano/ackley.html>

1.2.4. Funkcja Rosenbrocka

Funkcja Rosenbrocka jest to niewypukła funkcja, która jest bardzo popularna w kontekście optymalizacji jako test dla algorytmów optymalizacyjnych. Nazywana jest również przez wzgląd na swój kształt "*Funkcją Bananową Rosenbrocka*" albo "*Doliną Rosenbrocka*" [3]. Funkcja Rosenbrocka powszechnie jest stosowana w kontekście optymalizacji funkcji dwóch zmiennych. Istnieje jednak możliwość rozszerzenia jej do wielu wymiarów. W prezentowanej pracy magisterskiej rozpatrywana jest funkcja Rosenbrocka czterech zmiennych, której minimum globalne mieści się w punkcie $x = (1, 1, 1, 1)$ i wynosi 0. Ustaloną dziedziną funkcji jest $x_i \in [-200, 200]$ dla $i = 1, 2, 3, 4$. Opisywana funkcja Rosenbrocka jest funkcją trudną do minimalizacji i według wyników testów zawartych w [2], procentowy sukces znalezienia minimum globalnego wyniósł tylko 8,42%.



$$f(x) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

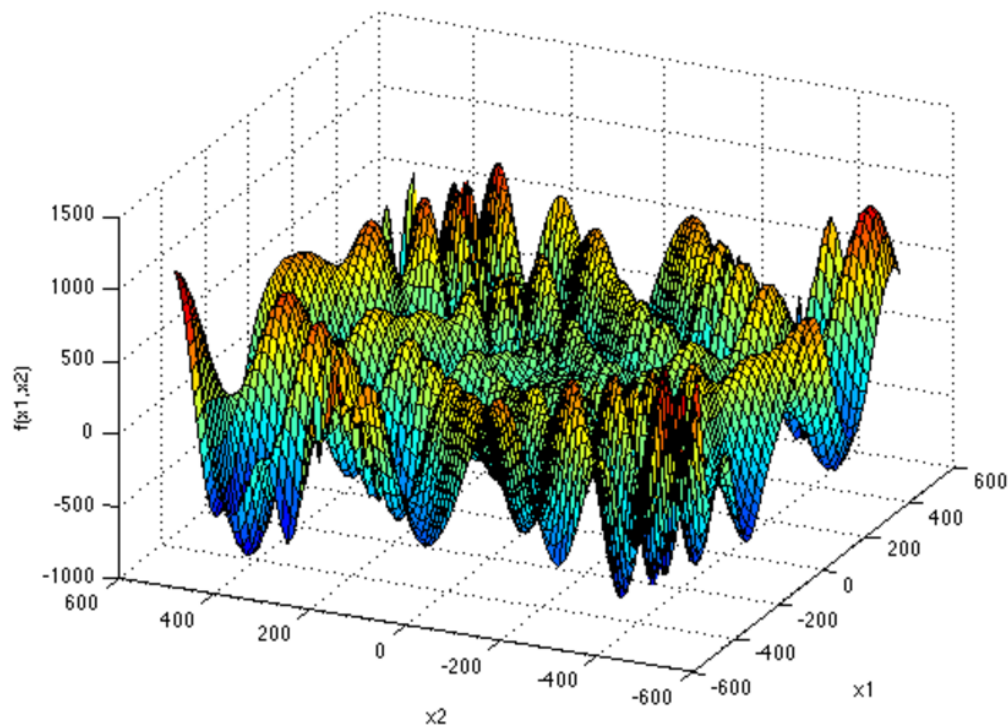
Rysunek 3: Wykres oraz wzór funkcji Rosenbrocka dla $d = 4$ zmiennych, Źródło: <https://www.sfu.ca/~ssurjano/rosen.html>

1.2.5. Funkcja Eggholdera

Kolejną wybraną funkcją została dwuwymiarowa funkcja Eggholdera. Jest to funkcja, której graficzna reprezentacja zawarta w rysunku 4 przypomina pasmo górskie z licznymi dolinami. Jest to funkcja trudna do optymalizacji przez wzgląd na licznie występujące minima lokalne. Dziedzina funkcji powszechnie rozpatrywana jest w przedziale $x_i \in [-512, 512]$ dla $i = 1, 2$. Minimum globalne znajduje się w punkcie $x = (512, 404.2319)$ i wynosi -959.6407 . Przez wzgląd na jego nietypowe położenie i wspomniane już liczne minima lokalne, wyłącznie 18.92% testów zawartych w [2] zakończyło się jego znalezieniem.

1.2.6. Funkcja Griewanka

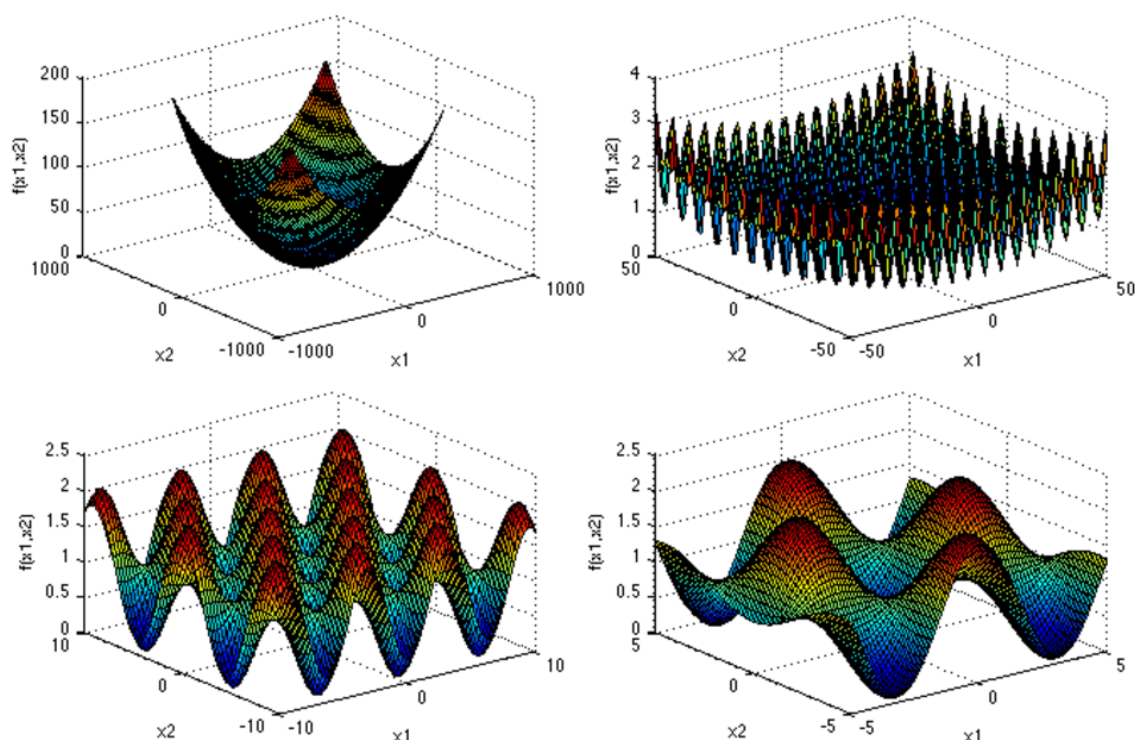
Ostatnią wybraną do testów funkcją jest funkcja Griewanka, której minimum globalne znalezione zostało w tylko 6.08% testów zawartych w [2]. Jest to funkcja po-



$$f(x) = -(x_2 + 47) \sin(\sqrt{|x_2 + \frac{x_1}{2} + 47|}) - x_1 \sin(\sqrt{|x_1 - (x_2 + 47)|})$$

Rysunek 4: Wykres oraz wzór funkcji Eggholdera dwóch zmiennych, Źródło: <https://www.sfu.ca/~ssurjano/egg.html>

siadająca bardzo dużo regularnie rozmieszczonych minimów lokalnych które utrudniają znalezienie globalnego minimum znajdującego się w punkcie $x = (0, \dots, 0)$, dla którego wartość funkcji wynosi 0. Opisywana funkcja jest funkcją wielowymiarową, która w prezentowanej pracy magisterskiej zastosowana została jako funkcja testowa trzech zmiennych o dziedzinie $x_i \in [-200, 200]$ dla $i = 1, 2, 3$. Na rysunku 5 zaprezentowano opisywaną funkcję w postaci dwuwymiarowej na której można dostrzec liczne minima globalne oraz ich regularne rozmieszczenie.



$$f(x) = \sum_{i=1}^d \frac{x_i^2}{4000} - \prod_{i=1}^d \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Rysunek 5: Wykres oraz wzór funkcji Griewanka dla $d = 3$ zmiennych, Źródło: <https://www.sfu.ca/~ssurjano/gri.html>

2. Metody rozwiązywania wybranego problemu optymalizacji

2.1. Algorytmy dokładne

Klasycznym podejściem do znalezienia minimalnej wartości zadanej funkcji testowej jest próba porównania wartości funkcji dla wszystkich możliwych parametrów wejściowych i wybrania w ten sposób najlepszego rozwiązania. W rzeczywistości jednak takie rozwiązanie nie jest praktyczne przez wzgląd na olbrzymią możliwą liczbę takich parametrów. Jak już wspomniano w rozdziale 1.1, liczba rozwiązań dla funkcji ograniczonej już do dwóch zmiennych może sięgać wartości 10^{20} , a w przy-

padku trzech zmiennych – 10^{30} . Zakładając, iż możliwe by było wyliczenie miliarda wartości funkcji na sekundę, to w godzinę wartość ta wynosiłaby $3.6 * 10^{12}$, a w rok – $3.2 * 10^{16}$. Prowadząc dalej obliczenia wychodzi, iż wyliczenie 10^{20} wartości funkcji trwałoby około 3125 lat. Widać więc, że liczba kombinacji jest tak duża, iż podejście to jest niemożliwe do wykonania w akceptowalnym czasie.

2.2. Metaheurystyki optymalizacyjne

Analizując podejście z rozdziału 2.1 może przyjść na myśl sposób, który polegać będzie na wyliczaniu wartości funkcji dla wyrywkowych parametrów. Skąd jednak wiadomo które punkty wybrać? Na podstawie czego bazować? W przypadku problemów, w których przez wzgląd na czas niemożliwe jest dojście do jednoznacznego rozwiązania na ratunek przychodzą algorytmy heurystyczne, które umożliwiają skrócenie czasu obliczeń. Ceną którą trzeba jednak za to zapłacić jest możliwość otrzymania gorszego rozwiązania od rozwiązania najlepszego. Samo pojęcie heurystyki pochodzi od greckiego słowa *heuresis*, które oznacza "odnaleźć" [4]. Metody heurystyczne polegają na użyciu reguł oraz faktów, które uzyskane na drodze badania danego problemu umożliwiają jego rozwiązanie lub zbliżenie się do poprawnej odpowiedzi. Podejście heurystyczne stosowane może być w sposób piętrowy, tworząc metaheurystyki. Metaheurystyka jest to ogólny algorytm do rozwiązywania problemów obliczeniowych, który inspirację często bierze z mechanizmów biologicznych lub fizycznych. Określenie to oznacza tak zwaną heurystykę wyższego poziomu, co wynika z faktu, iż algorytmy tego typu bezpośrednio nie rozwiązują żadnego problemu, a wyłącznie podają metodę na utworzenie odpowiedniego algorytmu.

2.2.1. Metoda optymalizacji rojem cząstek

Metoda roju cząstek (PSO – Particle Swarm Optimization) jest przykładem optymalizacji z kategorii metod inteligencji stadnej. Powstała ona w wyniku inspiracji biologicznej, której źródłem był układ lotu stada ptaków tworzony w celu znalezienia pożywienia lub gniazda oraz uniknięcia drapieżników. Zastosowanie prostych zasad umożliwia ptakom zsynchronizowany oraz bezkolizyjny ruch, który daje efekt zachowania jednego organizmu. Ruch stada ptaków, czy ławicy ryb jest wypadkową działania wszystkich osobników i koncentruje się na utrzymaniu optymalnego dystansu od swoich sąsiadów, przy jednoczesnym podążaniu za lide-

rem. Badania nad optymalizacją roju cząstek zapoczątkowano od próby graficznej symulacji zachowań takich grup [5]. Bardzo szybko okazało się, iż stworzony matematyczny model może być również zastosowany jako metoda optymalizacyjna. W optymalizacji rojem cząstek, rozwiązania (cząstki) współpracują ze sobą w celu odnalezienia cząstki optymalnej. W czasie procesu optymalizacji następuje zmiana położenia każdej cząstki w przestrzeni rozwiązań poprzez wyznaczenie wektora prędkości. Wektor ten jest modyfikowany przy użyciu informacji o historii poszukiwań danej cząstki oraz jej cząstek sąsiednich. Metoda PSO w problemie optymalizacji funkcji wielowymiarowych dąży do otrzymania cząsteczki, która reprezentuje jak najmniejszą wartość funkcji i może być opisana dwoma równaniami:

$$v = W * v + c_1 * r_1 * (p - x) + c_2 * r_2 * (g - x) \quad (1)$$

$$x_{next} = x + v \quad (2)$$

gdzie,

v - aktualny wektor prędkości cząstki

x - aktualna pozycja cząstki

W – parametr z zakresu $[0, 1]$, który determinuje wpływ poprzedniego położenia cząstki na jej obecną pozycję

p - najlepsze rozwiązanie dla cząstki

g – najlepsze rozwiązanie dla sąsiedztwa cząstek

r_1, r_2 – losowe liczby z zakresu $[0, 1]$

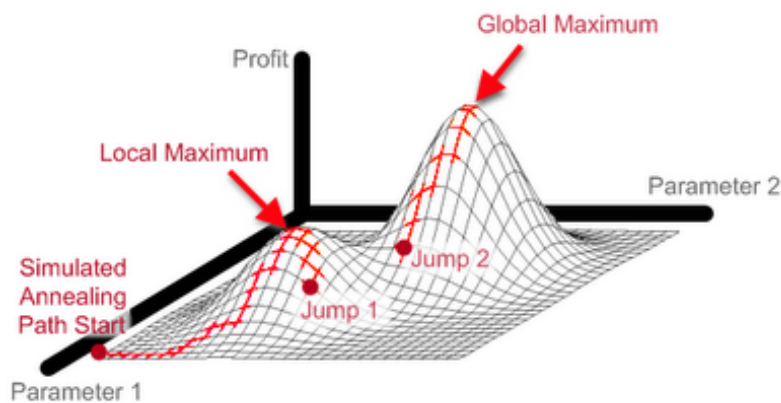
c_1, c_2 – parametry skalujące z zakresu $[0, 1]$

x_{next} - nowa pozycja cząstki

Nawiązując do powyższych równań, każda cząstka roju przeszukuje przestrzeń rozwiązań zmieniając położenie na podstawie swoich najlepszych rozwiązań p , jednocześnie wykorzystując informację o najlepszym rozwiązaniu w sąsiedztwie g . Parametry skalujące umożliwiają kontrolę wpływu danych składowych wektora prędkości na wynik. W przypadku, w którym c_1 będzie równe zero, cząstka będzie wykorzystywała tylko i wyłącznie informację o najlepszym rozwiązaniu w roju. Z kolei jeśli wartość parametru c_2 zostanie ustawiona na zero, cząstka będzie poszukiwała rozwiązania samodzielnie, bez uwzględnienia rozwiązań, które uzyskane zostały przez inne cząstki.

2.2.2. Symulowane wyżarzanie

Algorytm symulowanego wyżarzania po raz pierwszy został opisany w 1953 roku przez Nicolasa Metropolis. Sposób działania algorytmu jak i również jego nazwa odnosi się do procesów fizycznych, które wykorzystywane są w metalurgii. Proces wyżarzania polega na rozgrzaniu ciała stałego do określonej temperatury, a następnie jego powolnym studzeniu. Konsekwencją tego działania jest zmiana struktury krystalicznej materiału, który poddany został wyżarzaniu. W czasie procesu ochładzania metali dostrzeżono, iż cząsteczki ciała wraz z jego powolnym schładzaniem tworzą bardziej regularne struktury, niż w przypadku szybszego obniżenia temperatury, kiedy to chłodzone cząsteczki nie potrafią znaleźć optymalnego położenia. Algorytm symulowanego wyżarzania jest usprawnieniem starszych metod iteracyjnych, które polegały na ciągłym ulepszaniu istniejącego rozwiązania do momentu braku możliwości jego poprawy. W metodach tych zatrzymanie algorytmu mogło nastąpić przy rozwiązaniu pseudo-optymalnym – lokalnym minimum. Nie istniała wówczas możliwość wyjścia z owego lokalnego minimum i kierowania się w kierunku minimum globalnego. Bardzo ważną cechą opisywanego algorytmu jest możliwość wyboru, z pewnym prawdopodobieństwem, gorszego rozwiązania.



Rysunek 6: Zasada działania algorytmu symulowanego wyżarzania w postaci graficznej, Źródło: <http://iacs-courses.seas.harvard.edu/courses/am207/blog/images/mcmc.png>

Dzięki temu problem utknięcia w lokalnym minimum nie jest już groźny. Za wybór gorszego rozwiązania ma wpływ podstawowy parametr przeniesiony z podstaw termodynamicznych algorytmu – temperatura. Im jest ona wyższa, tym większe ist-

nieje prawdopodobieństwo wyboru i zaakceptowania gorszego rozwiązania. W czasie działania algorytmu, temperatura obniża się i działanie algorytmu zbliża się w swoim zachowaniu do typowych metod iteracyjnych.

W celu wykonania algorytmu symulowanego wyżarzania w kontekście optymalizacji funkcji wielu zmiennych należy na początku losowo wygenerować punkt startowy, który mieści się na płaszczyźnie poszukiwań, wyliczyć dla niego wartość funkcji oraz wybrać temperaturę startową z dostępnego zakresu $[0, 100]$. Każda iteracja polega na wyborze losowego rozwiązania z sąsiedztwa, wyliczenia dla niego wartości funkcji i porównaniu z obecnie najlepszym rezultatem oraz obniżeniu temperatury. W przypadku, w którym wartość funkcji nowego punktu jest mniejsza (lepsz), jest on zaklasyfikowany jako najlepszy. W przeciwnnej sytuacji punkt nie jest natychmiastowo odrzucany. Algorytm akceptuje gorsze rezultaty bazując na funkcji akceptacyjnej, która prezentuje się następująco:

$$\frac{1}{1 + \exp(\frac{\Delta}{\max(T)})} \quad (3)$$

gdzie,

Δ – różnica pomiędzy wartościami najlepszego oraz nowego punktu

T – wartość temperatury

W sytuacji, w której Δ i T są wartościami dodatnimi, prawdopodobieństwo akceptacji mieści się pomiędzy 0 i $\frac{1}{2}$. Niższa temperatura prowadzi do mniejszego prawdopodobieństwa zaakceptowania gorszego rezultatu. Podobnie jest z deltą – im większa delta tym mniejsza szansa na zaakceptowanie.

2.2.3. Algorytm genetyczny

Model algorytmu genetycznego po raz pierwszy zaprezentowany został w 1975 roku przez Johna Hollanda, który w pracy *"Adaptation in Natural and Artificial Systems"* [6] przedstawił fundamenty założeń dotyczących adaptacji darwinowskiej teorii ewolucji w systemach informatycznych.

W opisie algorytmu genetycznego posługuje się powszechną terminologią biologiczną. Z tego też powodu mówi się, iż algorytmy genetyczne przetwarzają populację osobników, która reprezentuje rozwiązanie danego problemu. Każdy element populacji nazywany jest chromosomem, a jego składowe genami. Allele z kolei, są to możliwe stany (wartości) genu, które umiejscowione są na pozycjach zdefiniowanych

jako locus. W badanych modelach komputerowych, osobniki (chromosomy) mogą być opisane jako różne struktury – zaczynając od łańcuchów binarnych, a kończąc na bardzo złożonych obiektach. W określonej iteracji zwanej zamiennie pokoleniem albo generacją, dane chromosomy na bazie określonej miary ich dostosowania podlegają ocenie. Ocena ta skutkuje wyborem najlepiej przystosowanych osobników, które wezmą udział w kolejnych iteracjach algorytmu. Nim jednak wybrane osobniki populacji utworzą nową generację, zostają poddane modyfikacjom bazujących na podstawowych operacjach genetycznych – krzyżowaniu, selekcji oraz mutacji.

W kontekście problemu optymalizacji funkcji wielu zmiennych, inicjalizacja algorytmu genetycznego polega na wygenerowaniu populacji początkowej, która złożona jest z określonej liczby chromosomów (punktów). Każdy chromosom reprezentowany w populacji posiada taką samą długość, która ustalona jest zależnie od rozwiązywanego problemu na etapie implementowania algorytmu. Przed rozpoczęciem etapu generowania musi być jednak określony sposób kodowania informacji zawartej w chromosomie. W algorytmie Hollanda nie było domyślnie zdefiniowanego sposobu kodowania chromosomów. Powszechnie uznaje się jednak stosowanie kodowania binarnego [7]. Takie też kodowanie jest zastosowane w kontekście omawianego problemu optymalizowania funkcji wielu zmiennych. Kolejnym etapem, który następuje po wygenerowaniu populacji początkowej oraz wyborze kodowania chromosomów jest wyznaczenie jakości chromosomów danej populacji. W tym celu obliczana jest wartość tak zwanej funkcji oceny, która definiuje poziom dopasowania konkretnego chromosomu. Tym sposobem można stwierdzić, które z nich lepiej rozwiązują dane zagadnienie, a które gorzej. Znalezienie rozwiązania danego problemu sprowadza się do znalezienia ekstremum wspomnianej funkcji oceny. Kolejną częścią algorytmu jest zastosowanie mechanizmu selekcji, który definiuje sposób wyboru rozwiązań rodzicielskich, z których tworzone będą tak zwane rozwiązania potomne użyte w następnej generacji. Prostym w implementacji operatorem selekcji dla algorytmu genetycznego jest metoda ruletki [8]. Metoda ta polega na przydzieleniu każdemu chromosomowi z danej populacji prawdopodobieństwa według wzoru:

$$p_i = \frac{f_i}{\sum_{j=1}^N f_j} \quad (4)$$

gdzie,

f_i - wartość funkcji oceny chromosomu i -tego

p_i – prawdopodobieństwo reprodukcji chromosomu i -tego
 N - liczba chromosomów w populacji

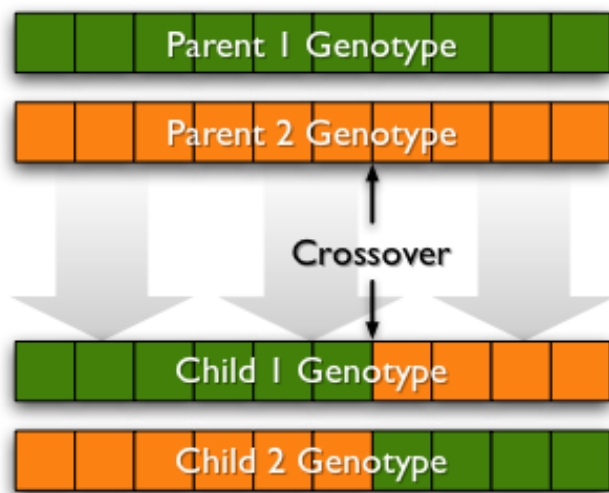
W celu wybrania puli rodzicielskiej, koło ruletki o obwodzie jeden dzielone jest na części o długości p_i , a następnie z zakresu $[0, 1]$ losowana jest liczba, która jednoznacznie identyfikuje punkt na ruletce, a co za tym idzie konkretny chromosom. Chromosom ten brany będzie pod uwagę w procesie następnej reprodukcji, a losowanie powtarzane jest tak długo, aż wylosowana zostanie ustalona liczba chromosomów. W problemie minimalizacji funkcji wielu zmiennych istnieje możliwość, iż wartość funkcji oceny będzie ujemna. W celu zniwelowania problemu ujemnego prawdopodobieństwa powyższy wzór został zmodyfikowany stosując skalowanie przystosowania:

$$p_i = \frac{f_i - f_{min}}{\sum_{j=1}^N f_j - f_{min}} \quad (5)$$

gdzie f_{min} jest wartością funkcji przystosowania najgorszego chromosomu.

Po etapie selekcji pozostaje do zdefiniowania kwestia wymiany pokoleń. W implementacjach często stosowana jest metoda całkowitego zastępowania, w której cała aktualna populacja, podlega operacjom krzyżowania i mutacji. Innym sposobem jest metoda zastępowania częściowego, w której część najlepszych chromosomów obecnej populacji przechodzi do populacji potomnej bez żadnych zmian, a pozostałe elementy z kolei biorą udział w operacji krzyżowania i mutacji. Kolejną popularną praktyką jest zastosowanie zastępowania elitarnego, w którym na podstawie parametru określającego wielkość elity, część najlepszych osobników jest kopiowana do nowej generacji już na samym początku. Umożliwia to zapamiętanie najlepszych chromosomów, które mogą być zmienione w wyniku działania operatorów genetycznych rozpoczynających swoją pracę po zakończeniu etapu selekcji.

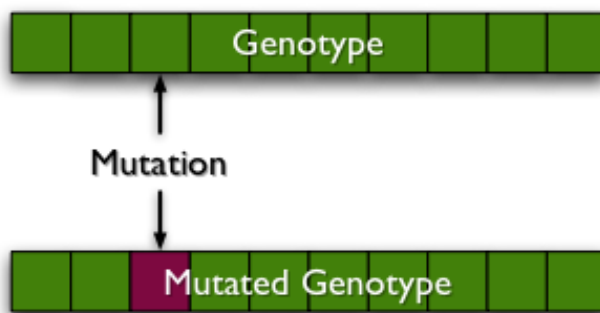
Pierwszym z takich operatorów jest krzyżowanie, które jest operacją umożliwiającą tworzenie nowych rozwiązań. Jego koncept bazuje na procesie rozmnażania organizmów, w czasie trwania którego potomek dziedziczy część genów rodziców. W kontekście omawianego algorytmu genetycznego, krzyżowanie polega na przecięciu chromosomów w ustalonym punkcie i ich wzajemnego zamienienia. Operacja ta została przedstawiona na obrazku 7.



Rysunek 7: Graficzna prezentacja operacji krzyżowania jednopunktowego,
 Źródło: <https://www.linkedin.com/pulse/genetic-algorithms-sharmishtha-mahajan-patwardhan>

Drugim operatorem genetycznym jest mutacja, która umożliwia wprowadzenie nowego elementu do populacji poprzez tworzenie różnorodności. Analogicznie jak w otaczającym nas świecie, w algorytmie genetycznym mutacje zdarzają się rzadko. Ich skala zależy od parametru, który przyjmuje zazwyczaj niskie wartości. W odniesieniu do chromosomów w postaci binarnej, mutacja może polegać na zamianie losowego genu na wartość przeciwną.

Tak utworzona nowa generacja ponownie przechodzi przez wszystkie punkty algorytmu. Dzieje się tak aż do czasu, w którym spełnione zostaną warunki zatrzymania, które w kontekście przedstawianego zagadnienia opisane zostały w rozdziale 6.1 opisującym metody porównawcze wybranych algorytmów obejmujące w swojej treści warunki stopu opisanych trzech metaheurystyków.



Rysunek 8: Graficzna prezentacja operacji mutacji,

Źródło: [http://web.arch.usyd.edu.au/](http://web.arch.usyd.edu.au/~rob/applets/house/images/single_point_mutation.png)

[~rob/applets/house/images/single_point_mutation.png](http://web.arch.usyd.edu.au/~rob/applets/house/images/single_point_mutation.png)

3. Automatyzacja przeprowadzanych badań

Porównanie algorytmów metaheurystycznych w odniesieniu do problemu minimalizacji funkcji wielu zmiennych jest zadaniem bardzo skomplikowanym przez fakt ilości doświadczeń, które należy wykonać. Badania porównawcze przeprowadzane będą pod kątem kilku parametrów. Dla każdego z nich wymagane będzie przeprowadzenie wielokrotnych testów dla każdego testowanego algorytmu. Ustalając próbkę testu na poziomie 10^3 prób, z której obliczana będzie wartość średnia oraz biorąc pod uwagę dwa parametry porównawcze, z których każdy przyjmować może 10 wartości, liczba testów dla jednego algorytmu oraz jednej funkcji testowej osiąga już wartość $2 * 10^4$. Manualne przetwarzanie w środowisku Matlab przy ograniczonej funkcjonalności agregowania wyników jest bardzo czasochłonnym zadaniem. W tym celu zdecydowano się zbudować dedykowaną aplikację bazodanową, która będzie umożliwiała automatyzację przedstawianej operacji. Użytkownik z poziomu aplikacji będzie miał możliwość wyboru funkcji testowej, algorytmu metaheurystycznego oraz jego parametrów, a także liczby testów, która ma być przeprowadzona przy zadanych wartościach. Aplikacja automatycznie będzie wysyłała zapytania do środowiska obliczeniowego Matlab, obierała wyniki oraz umieszczała je w bazie danych, z której w prosty i efektywny sposób można będzie przeprowadzić ich agregację w

celu przedstawienia wyników i ich porównania. W ten sposób narzut pracy, który niezbędny jest do implementacji przedstawionej aplikacji zwróci się oraz umożliwi dalszą analizę algorytmów w kontekście problemu optymalizacji funkcji wielu zmiennych ponieważ aplikacja umożliwiać będzie również proste dodanie do niej kolejnych algorytmów oraz funkcji testowych.

4. Wykorzystane rozwiązania technologiczne

4.1. Zastosowane technologie

Obecny rozdział obejmuje opis wykorzystanych technologii, które umożliwiły implementację aplikacji. Opis ten dotyczy języka programowania, w którym napisany został backend [9] aplikacji oraz silnika graficznego programu wraz z językiem zapytań bazy danych.

4.1.1. .NET Framework/C#

.NET Framework jest to platforma programistyczna wydana przez firmę Microsoft w 2002 roku. Przeznaczona jest ona do wytwarzania oprogramowania przeznaczonego dla systemów operacyjnych z rodziny Windows. Główną składową przedstawianej platformy są kompilatory języków wysokiego poziomu, które umożliwiają przeprowadzenie operacji kompilacji programów napisanych w językach Visual Basic, F#, C++/CLI. Platforma .NET wspiera również jeden z najpopularniejszych języków programowania na świecie - C#, w którym napisana została aplikacja odnosząca się do prezentowanej pracy magisterskiej. Według rankingu TIOBE Software z sierpnia 2017 roku, język C# uplasowany jest na czwartym miejscu, spośród pięćdziesięciu najpopularniejszych języków, które zostały wzięte pod uwagę [10]. C# jest to nowoczesny, zorientowany obiektowo język programowania stworzony przez firmę Microsoft. Jego pierwsza wersja wydana została już w połowie 2000 roku, przy dużej zasłudze głównego projektanta języka - duńskiego inżyniera oprogramowania Andersa Hejlsberga [11]. Język ten może zostać użyty w celu pisania aplikacji webowych, desktopowych oraz przeznaczonych na urządzenia przenośne. Programy w nim napisane kompilowane są do pośredniego kodu, który zapisany jest w CIL (ang. Common Intermediate Language) i wykonany w środowisku uruchomieniowym .NET Frameworka. Poziom trudności nauki C# uznawany jest za stosunkowo niski, głównie z powodu posiadania licznych udogodnień oraz modułów, które upraszczają pracę programistyczną [12]. Do elementów tych można zaliczyć brak konieczności dodawania plików nagłówkowych, które niezbędne były w języku C++, automatyczne zwalnianie dynamicznie przydzielonej pamięci za które odpowiedzialny jest Garbage

Collection, inicjalizowanie zmiennych ich domyślnymi wartościami oraz wprowadzenie dodatkowych elementów składowych klas, takich jak indeksery oraz właściwości (ang. properties).

4.1.2. WPF / XAML

Windows Presentation Foundation (WPF) [13] jest to silnik graficzny, który wprowadzony został wraz z trzecią wersją środowiska .NET. Okna w aplikacjach zaimplementowanych w WPF wyświetlane są za pomocą grafiki wektorowej, która to wspomagana jest przez akceleratory grafiki 3D. Frontendowe API w technologii WPF opiera się na języku XML [14], a konkretniej jego odmianie - XAML. Extensible Application Markup Language (XAML) jest to deklaratywny język znaczników, którego przeznaczeniem jest opis interfejsu użytkownika implementowanego w WPF. Pozwala on zaprojektowanie oraz rozmieszczenie wszelakich elementów wizualnych, a także umożliwia zrównoleglenie pracy programistów pracujących nad logiką biznesową budowanej aplikacji oraz grafików, którzy odpowiedzialni są za tworzenie graficznego interfejsu użytkownika. Zdarza się, iż graficy przez wzgląd na zakres swoich obowiązków nie znają żadnego języka programowania. Problem ten został zniwelowany dzięki językowi XAML, który pozwala na zrozumienie przez osoby nietechniczne zasady działania oraz powiązań poszczególnych okien. Umożliwia on także projektowanie graficznego interfejsu użytkownika w prosty sposób z poziomu drzewiastej struktury lub dedykowanego programu graficznego Expression Blend, który to umożliwia przeprowadzenie wszelkich operacji z poziomu graficznego środowiska.

4.1.3. TSQL

Transaction-SQL (T-SQL) [15] jest to rozwinięcie standardowego języka SQL, który to utworzony został w latach siedemdziesiątych specjalnie na potrzeby relacyjnych baz danych przez firmę IBM. T-SQL pozwala na tworzenie konstrukcji takich jak instrukcje warunkowe i pętle oraz umożliwia stosowanie zmiennych. Aktualnie stosowany on jest do tworzenia zapytań bazodanowych przez firmy, które potrzebują bardziej zaawansowanych struktur niż te, które dostępne są w standardowym SQL. T-SQL wprowadził możliwość stosowania bazodanowych wyzwalaczy, procedur oraz funkcji składowanych, które to przy rozbudowanej strukturze tabel ułatwiają pracę przy bazie danych.

4.2. Zastosowane narzędzia

W celu efektywnej pracy z technologiami, które opisane zostały w rozdziale 4.1 warto użyć dedykowanych narzędzi, które ułatwiają zastosowanie funkcjonalności wspomnianych technologii. W tym celu w trakcie pracy nad prezentowanym projektem magisterskim, użytych zostało kilka narzędzi, które pozwoliły skrócić czas potrzebny na implementację aplikacji, przeprowadzenie badań oraz zagregowanie ich rezultatów.

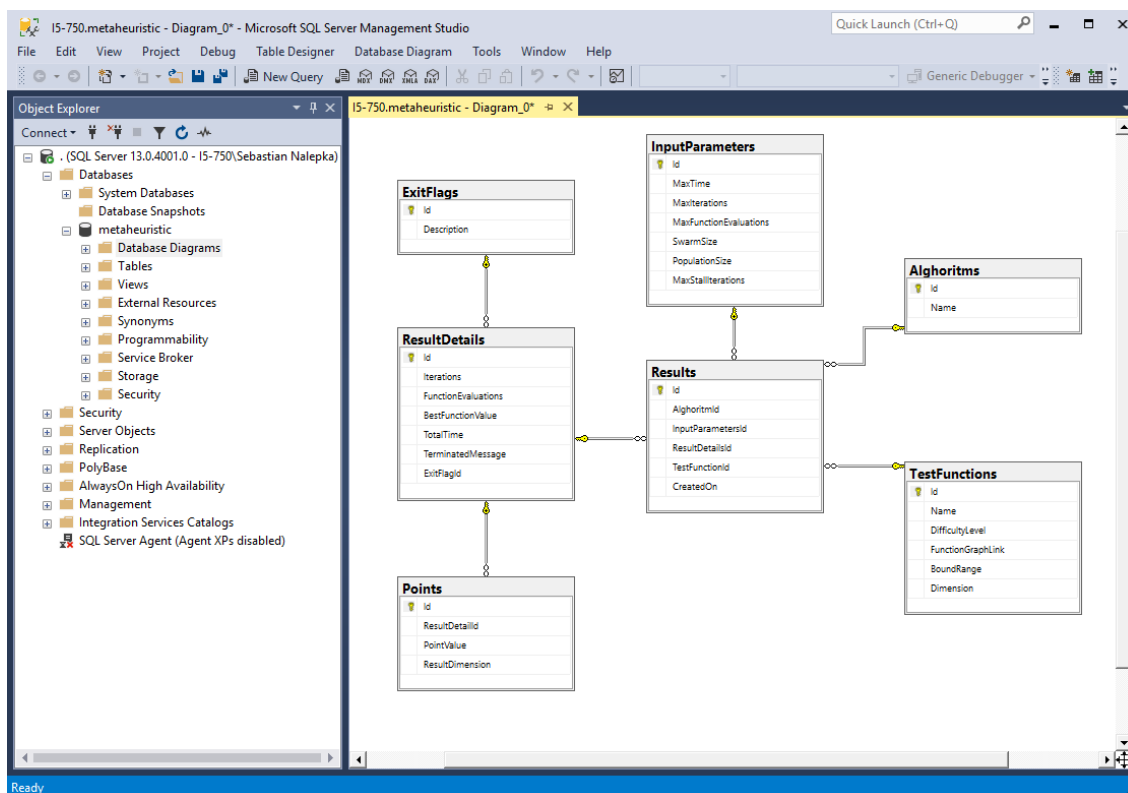
4.2.1. Matlab

Matlab jest to środowisko przeznaczone do wykonywania obliczeń numerycznych, matematycznych oraz symulacyjnych. Nazwa środowiska odnosi się do słów MATrix LABoratory, gdyż Matlab przeznaczony był początkowo do numerycznych obliczeń macierzowych. Aktualnie aplikacja ta ma zdecydowanie większe możliwości - posiada liczne funkcje biblioteczne, umożliwia ich rozbudowę, a także definiowanie nowych przez użytkownika. Matlab posiada również dedykowany język programowania, czego konsekwencją jest możliwość pisania w pełni funkcjonalnych aplikacji pracujących w opisywanym środowisku. W aspekcie grafiki, Matlab pozwala na rysowanie dwu oraz trójwymiarowych wykresów, a także wizualizację rezultatu obliczeń w postaci animacji. Istotną kwestią w kontekście prezentowanej pracy magisterskiej jest posiadanie przez środowisko Matlab terminala, który pozwala na wprowadzanie komend i wykonywanie funkcji. Istnieje możliwość jego samodzielnego uruchomienia oraz połączenia z innymi systemami za pośrednictwem dedykowanych bibliotek, które umożliwiają synchronizację Matlab z aplikacjami napisanymi w technologii .NET, Java czy C++.

4.2.2. SQL Server Management Studio 17

W czasie pracy nad aplikacją automatyzującą proces przeprowadzania testów wykorzystano darmowe narzędzie, którego przeznaczeniem jest zarządzanie bazą danych - SQL Server Management Studio 17. Potrzeba użycia narzędzia wynikła z istniejącej infrastruktury aplikacji, która opiera się na rozwiązaniach Microsoftu. Zastosowane narzędzie znacznie ułatwiło pracę i wspomogło proces projektowania bazy danych dzięki funkcjonalności generowania diagramów, które umożliwiły podgląd struktury bazy oraz relacji pomiędzy poszczególnymi tabelami. Kolejną olbrzymią

zaletą SQL Management Studio jest możliwość tworzenia zapytań bazodanowych, które pozwalają w bardzo elastyczny sposób pogrupować olbrzymie ilości danych w wybrany przez użytkownika sposób. Umożliwia to w krótkim czasie stworzenie statystyk zastosowanych algorytmów, które dynamicznie będą się aktualizowały wraz z napływem kolejnych danych do bazy.



Rysunek 9: Interfejs SQL Server Management Studio 2017 Źródło: Opracowanie własne.

4.2.3. Visual Studio 2015

Visual Studio [16] jest to bardzo rozbudowane środowisko deweloperskie firmy Microsoft. Stosowane jest do procesu wytwarzania oprogramowania z graficznym interfejsem użytkownika w technologii WPF, WinForms, Web Sites oraz Xamarin. Visual Studio posiada zaawansowany edytor kodu, który wspiera mechanizm podpowiadania składni kodu - IntelliSense, który może być dodatkowo rozbudowany poprzez dedykowane narzędzia np. ReSharper. Zintegrowany debugger zawarty w środowisku Visual Studio umożliwia również analizę programu w czasie jego dzia-

łania, dzięki czemu możliwe jest odnalezienie w bardzo łatwy sposób potencjalnych błędów oraz sprawdzenie poprawności zaimplementowanego rozwiązania. Funkcje te, wraz z wbudowanym designerem do tworzenia aplikacji WPF oraz narzędziem do projektowania baz danych umożliwiły skoncentrowanie pracy programistycznej przy jednym narzędziu, dzięki czemu możliwa była duża oszczędność czasowa w zakresie poszukiwań niezbędnych narzędzi i ich integracji. Visual Studio, SQL Server Management Studio 17 oraz zastosowane technologie są wytwarzane przez jedną firmę - Microsoft. Dzięki temu aspektowi ich połączenie i synchronizacja jest bezproblemowa i bardzo szybka.

4.2.4. Resharper

Wydajność pracy w temacie komercyjnych projektów programistycznych jest priorytetem, który bezpośrednio przekłada się na oszczędność czasową, a co za tym idzie na korzyści finansowe. Dostępne są narzędzia, których przeznaczeniem jest poprawienie efektywności osoby implementującej aplikację poprzez kontrolę pisanego przez niego kodu według zdefiniowanych uprzednio zasad oraz zautomatyzowanie często powtarzanych czynności. Resharper jest to dodatek do środowiska Visual Studio, który w znacznym stopniu rozbudowuje dostępne jego funkcjonalności, ułatwiając przy tym refaktoryzację oraz pisanie kodu. Funkcje, które zastosowane być mogą z poziomu Resharpera można pogrupować na kilkanaście grup, z których jedną z najważniejszych jest moduł, który zajmuje się inspekcją kodu. W trakcie pisania kodu z uruchomionym w tle Resharperem następuje sprawdzanie w czasie rzeczywistym ponad 2000 zasad, które dotyczą jego poprawności i w sytuacji znalezienia niespójności, następuje poinformowanie programisty na poziomie interfejsu Visual Studio o zaistniałym błędzie wraz z jego szczegółowym opisem i miejscem wystąpienia. Niespójności te mogą dotyczyć dla przykładu możliwości zastąpienia części kodu jego wydajniejszą wersją, ostrzeżeniem programisty przed fragmentem kodu, który potencjalnie może spowodować błędne działanie całej aplikacji oraz fragmentami martwego kodu. Kolejną bardzo ważną grupą funkcji jest funkcjonalność generowania kodu. W czasie pracy implementacyjnej sporo czynności takich jak edycja nazw czy pisanie nowych klas oraz metod notorycznie się powtarza. Za pomocą skrótów klawiaturowych, wszystkie powyższe akcje Resharper wykona za programistę. Wygeneruje on niezbędne przy dziedziczeniu interfejsu wszystkie jego składowe, przeniesie daną klasę do odrębnego pliku oraz automatycznie dokona zamiany wy-

branej nazwy na inną w obrębie całej solucji. Zaprezentowane funkcje Resharpera wraz z innymi, które dostępne są do wglądu w odnośniku [17], umożliwiły zmniejszenie czasu potrzebnego na implementację niezbędnych w projekcie modułów poprzez zminimalizowanie liczby pojawiających się zagrożeń we wczesnej fazie pisania kodu oraz zautomatyzowanie powtarzalnych czynności.

4.3. Wykorzystane biblioteki zewnętrzne

W trakcie budowy systemu informatycznego zdarzają się sytuacje, w których konieczne jest korzystanie z aplikacji różnych firm, które domyślnie nie umożliwiają ich połączenia. Problem taki wystąpił również w czasie pracy nad prezentowanym projektem magisterskim. Użycie narzędzia innego producenta niż Microsoft, jakim jest Matlab doprowadziło do konieczności znalezienia metody jego połączenia ze środowiskiem .NET.

4.3.1. Matlab Application Type Library v.1.0

Matlab Application Type Library w wersji pierwszej jest to otwarta biblioteka przeznaczona dla środowiska .NET posiadająca API umożliwiające wykonywanie operacji Matlabowych stosując składnię C#. Biblioteka ta umożliwia odwoływanie się do uprzednio zdefiniowanych w Matlabie funkcji, dostarczając odpowiednich parametrów oraz odbiera wyniki obliczeń, które mogą być dalej przetwarzane bez konieczności ich manualnego kopiowania. Biblioteka posiada bardzo ogólnie zdefiniowane metody, do których prawidłowego wykonania wymagana jest dokładna wiedza, która jednak ograniczona jest przez wzgląd na brak dokumentacji. Przy dłuższej pracy jest jednak możliwe dojście do sposobu skorzystania na poziomie C# z zaawansowanych funkcji które dostarcza środowisko Matlab. W rozdziale 5.4 przedstawiono dokładny sposób przekazywania i odbierania danych pomiędzy środowiskiem .NET oraz Matlabem.

5. Architektura budowanej aplikacji

Budowanie bazodanowej aplikacji komputerowej komunikującej się z systemem trzecim wymaga szczegółowej analizy wszystkich elementów, które mają odniesienie do jej architektury. Przez konieczność budowy dwóch oddzielnych elementów - bazy danych oraz aplikacji desktopowej, analiza ta podzielona została na część odwołującą się do architektury aplikacji, modelowania bazy danych oraz komunikacji bazy danych z projektem programistycznym oraz komunikacji środowiska Matlab z implementowaną aplikacją.

5.1. Architektura aplikacji

W aspekcie programowania, podobnie jak w innych dziedzinach, w których przeprowadzana jest operacja budowy zadanego elementu, należy dokładnie zaplanować jego proces. W aktualnym rozdziale przedstawiony został opis metodologii, który pozwolił na uporządkowanie procesu budowania aplikacji komputerowej wraz z jego odmianą, która zastosowana została podczas pracy nad prezentowaną pracą magisterską. W rozdziale 5.1.3 został również zawarty szczegółowy opis poszczególnych elementów ze strony projektu programistycznego, który to jest efektem pracy implementacyjnej.

5.1.1. Wzorce architektoniczne oprogramowania

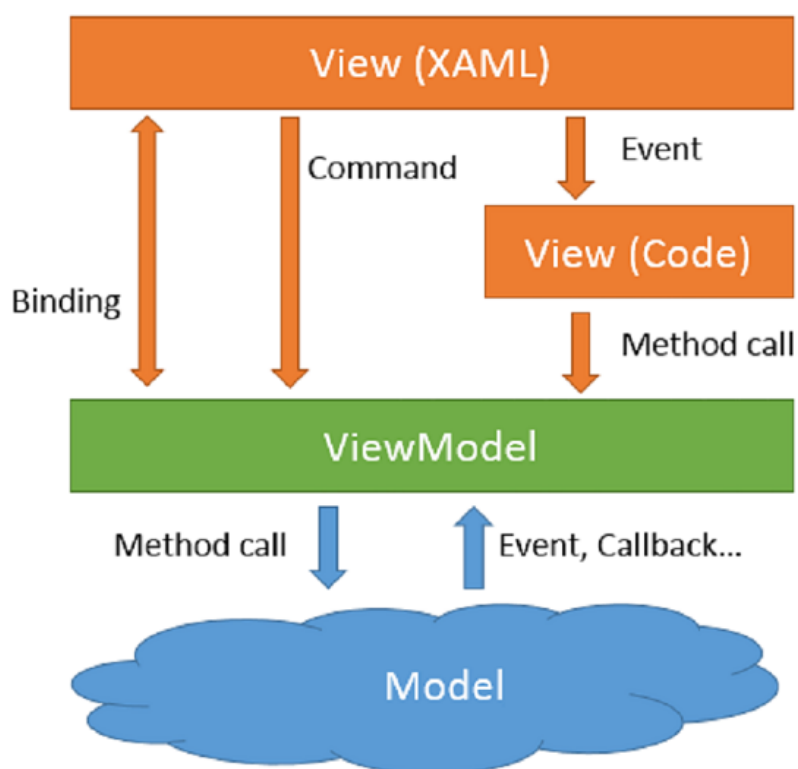
Bardzo dobrą praktyką programistyczną jest przeprowadzenie procesu budowy aplikacji w taki sposób, aby możliwy był efektywny jej rozwój oraz dokonanie w łatwy sposób zmian w istniejących już funkcjonalnościach. W tym celu budowę aplikacji należy przeprowadzać zgodnie z ustalonymi etapami produkcji oprogramowania [18]. Etapy te definiują obowiązek ustalenia wymagań budowanego systemu oraz określenia jego ogólnej architektury. Część dotycząca określenia architektury jest bardzo ważna ponieważ jej zmiana podczas fazy implementacji jest niebywale problematyczna do wykonania i w wielu przypadkach wręcz niemożliwa bez konieczności powtórnego rozpoczęcia pracy od początku. Kolejne etapy mają na celu

zrealizowanie zdefiniowanej wcześniej architektury poprzez zaprogramowanie każdego komponentu oprogramowania wraz z ich wszystkim wzajemnymi połączeniami, przetestowanie całości zaimplementowanego systemu, a także jego uruchomienie oraz zniwelowanie błędów, które pojawiły się w trakcie jego działania. W kontekście planowania i budowania architektury systemu bardzo użyteczne są wzorce architektoniczne, które to są powszechnymi, sprawdzonymi oraz ogólnie przyjętymi sposobami rozwiązania określonego problemu z zakresu architektury oprogramowania. Definiują one ogólną strukturę systemu informatycznego, elementy, które wchodzi w jego skład oraz zasadę komunikowania się komponentów pomiędzy sobą. Dokonanie wyboru odpowiedniego wzorca jest w dużym stopniu zależne od technologii, która stosowana jest w projekcie. W przypadku prezentowanej aplikacji oraz technologii WPF na której ona bazuje, stosowanym powszechnie wzorcem jest *Model View ViewModel (MVVM)* [19], który to jest odmianą ogólnego wzorca MVC.

5.1.2. Zastosowany wzorzec architektoniczny – MVVM

MVVM jest to wzorzec, który dzięki separacji warstwy logiki biznesowej oraz warstwy prezentacji, pozwala na tworzenie łatwo testowalnej aplikacji, której fragmenty kodu mogą być ponownie użyte w innych projektach programistycznych. Aplikacje implementowane w technologii WPF przy użyciu wzorca MVVM umożliwia również ich prostą rozbudowę, a poprzez obowiązek zachowania odpowiedniej struktury kodu, zrozumienie kodu przez nową osobę jest o wiele prostsze i mniej czasochłonne niż w przypadkach aplikacji pisanych bez żadnej ogólnie zachowanej struktury. MVVM jest to wzorzec, który jest bardzo popularny w gronie programistów skoncentrowanych przy technologii WPF. Powodem tego jest możliwość użycia największych zalet tej technologii, takich jak *binding* [20] (wiązania), *behavior* (zachowania) oraz *command* [21] (komendy). Struktura kodu aplikacji bazującej na opisywanym wzorcu, podzielona jest na trzy oddzielne warstwy, których nazwy składają się na nazwę wzorca: Model, Widok (View) oraz Model Widoku (ViewModel). Każda z warstw spełnia w aplikacji specjalnie określone funkcje i przetwarza dedykowane dla siebie dane. Graficzna prezentacja oraz kierunki przesyłania danych dostępna jest do wglądu na rysunku 10.

Warstwa Modelu we wzorcu MVVM odpowiedzialna jest za tak zwaną logikę biznesową budowanego systemu. W aplikacji implementowanej na potrzeby prezentowanej pracy magisterskiej, warstwa ta zawiera wszystkie klasy, które stworzone zo-



Rysunek 10: Przepływ danych we wzorcu MVVM, Źródło: <https://i-msdn.sec.s-msft.com/dynimg/IC648329.png>

stały za pomocą narzędzia mapowania obiektowo-relacyjnego Entity Framework w celu odwzorowania relacyjnej bazy danych na poziom obiektów dostępnych w kodzie. Każda klasa zawarta w części modelu, której dane mają zostać wysłane do warstwy widoku w celu ich wyświetlenia użytkownikowi zobligowane są do implementacji interfejsu *INotifyPropertyChanged*, który aktywnie współpracuje z wiązaniem stosowanym w WPF. Drugą z kolei warstwą jest warstwa widoku, której odpowiedzialność sprowadza się wyłącznie do wyświetlania danych. Jej rolą jest prezentowanie danych w oknie aplikacji, do którego dostęp mają użytkownicy systemu. Ostatnią warstwą modelu MVVM jest warstwa ViewModelu, której przeznaczeniem jest pośredniczenie w wymianie danych pomiędzy modelem oraz widokiem, do którego nie posiada jednak żadnej referencji. Elementy widoku odnoszą się do ViewModelu dzięki wspomnianym już komendom oraz wiązaniom. Taki mechanizm zapewnia w pełni separację warstwy ViewModelu, a co za tym idzie, umożliwia pełne jego przetestowanie bez żadnej zależności od rzeczywistej warstwy widoku oraz modelu.

5.1.3. Model aplikacji

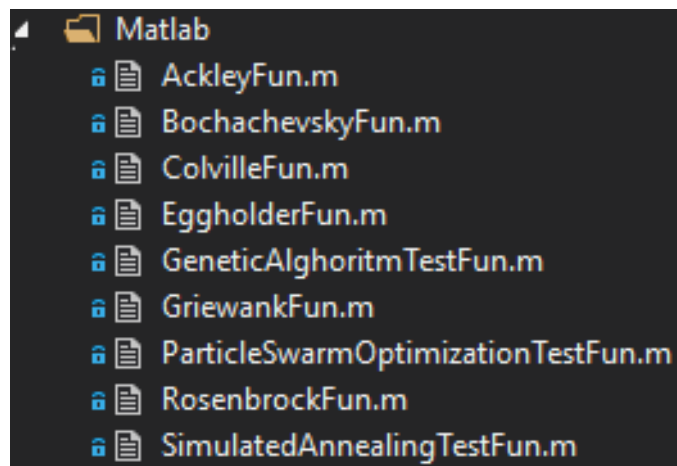
Użycie wzorca MVVM w aplikacji implementowanej na potrzeby prezentowanej pracy magisterskiej doprowadziło do powstania obszernej solucji, na którą składa się ponad siedemdziesiąt plików. Dzięki zaplanowanej strukturze podziału elementów na foldery, zachowano porządek, który umożliwia szybkie oraz intuicyjne wyszukiwanie potrzebnych składowych aplikacji. Wszelkie pliki klas zostały umiejscowione w katalogach nazwanych w sposób adekwatny do ich przeznaczenia, przez co odnalezienie przez programistę szukanego elementu nie przysparza żadnych problemów.

Managers

Managers jest to katalog zawierający klasę menadżera tworzącego i wysyłającego żądanie przeprowadzenia doświadczenia do środowiska Matlab. Menadżer ten za pośrednictwem klas dopasowujących tworzy żądanie zgodne ze standardem zastosowanej biblioteki opisanej w 4.3.1, wysyła je do środowiska obliczeniowego oraz odbiera rezultaty.

Matlab

Folder *Matlab* ukazany na rysunku 11 zawiera pliki funkcji zapisanych w środowisku Matlab. Pliki te zawierają definicje trzech algorytmów metaheurystycznych oraz wybranych funkcji testowych. Przez fakt na ograniczoną funkcjonalność zastosowanego API, łączącego środowisko .NET ze środowiskiem Matlab, definicje funkcji testowych oraz algorytmów musiały znaleźć się w jednym katalogu z powodu braku wsparcia dotyczącego przejścia pomiędzy różnymi katalogami.



Rysunek 11: Widok na pliki funkcji napisanych w środowisku Matlab z poziomu solucji, Źródło: Opracowanie własne.

Model

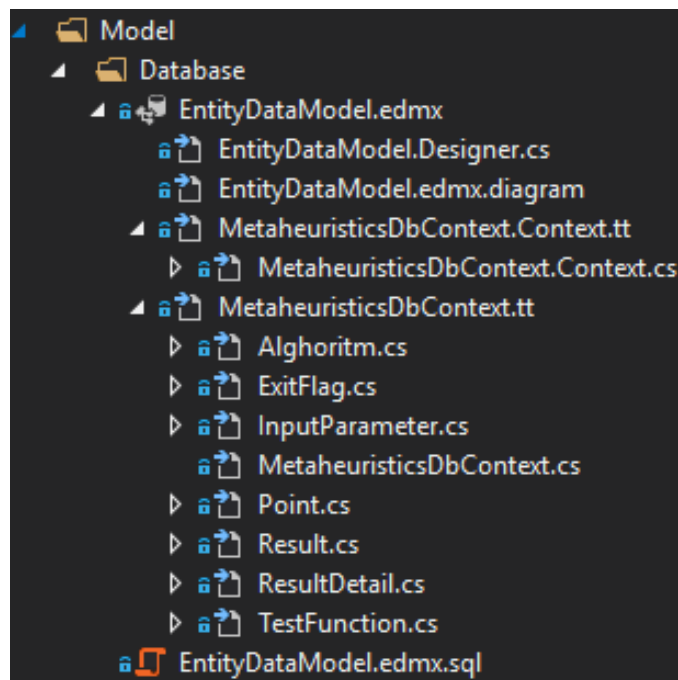
Katalog Modelu przedstawiony na obrazu 12 zawiera utworzoną przez Entity Framework strukturę obiektową relacyjnej bazy danych. Liczba plików, które zawiera uzależniona jest bezpośrednio od liczby tabel, która zastosowana została w bazie danych. W czasie działań implementacyjnych nie jest konieczna wiedza o dokładnej strukturze przedstawianego folderu, a tylko która bazodanowa tabela zawiera jakie dane.

Services

Katalog *Services* zawiera dwa serwisy stosowane w prezentowanej aplikacji. Pierwszym z nich jest serwis bazy danych który umożliwia połączenie się z fizycznie istniejącą bazą danych. Za pomocą dedykowanych obiektów Entity Frameworka umożliwia on zapisywanie do bazy danych rezultatów przeprowadzonych doświadczeń oraz pobieranie niezbędnych danych dotyczących stosowanych algorytmów oraz funkcji testowych. Drugim z kolei serwisem jest serwis Matlab, który przechowuje obiekty umożliwiające dwukierunkową komunikację ze środowiskiem obliczeniowym.

Tools

Katalog narzędzi jest najbardziej rozbudowanym katalogiem w drzewie solucji. Przez wzgląd na jego szerokie zastosowanie posiada on trzy podfoldery, które uści-



Rysunek 12: Widok na pliki wygenerowane przez Entity Framework z poziomu solution, Źródło: Opracowanie własne.

ślają definicję elementów w nich się znajdujących. Pierwszym z nich jest *Parser*, który zawiera tłumacza rezultatów otrzymanych ze środowiska obliczeniowego. Przez wzgląd na nieuporządkowaną formę danych otrzymywanych z Matlaba pojawiła się konieczność ich uporządkowania. W listingu 1 zawarty został kod metody umożliwiającej przeprowadzenie tej operacji. Drugi podkatalog zawarty w sekcji *Tools* to *Utils*, w którym umiejscowione są dwie bardzo ważne metody dopasowujące nazwy Matlabowych plików na podstawie wybranego algorytmu metaheurystycznego oraz funkcji testowej. Ostatnim podkatalogiem jest *MatlabContextWrapper*, który posiada metody ułatwiające odwoływanie się do obiektów komunikacyjnych, które pośredniczą w dwukierunkowym przekazywaniu informacji pomiędzy środowiskami .NET oraz Matlab.

```
public ResultDetail ParseResult(object[] result)
{
    // tworzenie obiektu zawierającego szczegóły otrzymanego
    // rezultatu
    var resultDetails = new ResultDetail
    {
        BestFunctionValue = Convert.ToDecimal(result[1]),
        Iterations = Convert.ToInt32(result[2]),
        FunctionEvaluations = Convert.ToInt32(result[3]),
        TotalTime = Convert.ToDecimal(result[4]),
    };

    // wyciąganie współrzędnych punktu wraz z numerem wymiaru
    // przestrzeni której dotyczą
    var resultPointLoc = result[0] as double[];
    for (var i = 0; i < resultPointLoc.Length; i++)
    {
        resultDetails.Point.Add(new Point
        {
            PointValue = resultPointLoc[0,i],
            ResultDimension = i + 1,
        });
    }

    return resultDetails;
}
```

Kod źródłowy 1: Implementacja oraz opis funkcji tłumaczącej informacje odebrane ze środowiska obliczeniowego Matlab.

View

Katalog widoku jest katalogiem najważniejszym ze strony użytkownika ponieważ zawiera definicję budowy głównego okna aplikacji oraz umiejscowienia wszystkich kontrolek jakie ono zawiera. W tym właśnie miejscu wykorzystywany jest opisany w rozdziale 4.1.2 język XAML, który pozwala na przeprowadzenie operacji wiązań oraz stosowanie komend.

ViewModel

Ostatni opisywany katalog - *ViewModel* - zawiera pliki dostosowane do pracy ze wzorcem MVVM. Odpowiadają one na interakcję użytkownika oraz realizują akcje dostępne z poziomu okna aplikacji, w którym pracuje użytkownik. Ważnym elementem ze strony programistycznej jest plik *ViewModelLocator*, w którym przeprowadzana jest operacja rejestrowania wszelkich typów zastosowanych w aplikacji. Dzięki temu, możliwe jest skorzystanie ze wzorca wstrzykiwania zależności [22], który pozwala na pisanie kodu o luźniejszych powiązaniach, prostszego w testowaniu oraz modyfikacji.

5.2. Architektura bazy danych

Analiza tematu projektu dotycząca porównania algorytmów metaheurystycznych doprowadziła do powstania potrzeby zastosowania środowiska bazodanowego, które umożliwiłoby przechowywanie oraz agregowanie wyników przeprowadzonych doświadczeń. W prezentowanym rozdziale opisany został serwer bazodanowy przechowujący instancję bazy wraz z metodą jego działania oraz model utworzonej bazy danych.

5.2.1. Microsoft SQL Server – użyty system do zarządzania bazą danych

Zastosowanie w projekcie bazy danych wymusiło obowiązek utworzenia oraz uruchomienia serwera bazodanowego. Przez fakt na zastosowane technologie, których opis znajduje się w rozdziale 4.1, użyty został Microsoft SQL Server. Jest to relacyjny system zarządzania bazą danych, którego podstawową funkcją jest przechowywanie danych oraz ich przetwarzanie na żądanie innych aplikacji, które mogą działać na tym samym komputerze lub na innej maszynie w sieci LAN czy też WAN. Przydatnym narzędziem przy pracy z Microsoft SQL Serverem jest dedykowane narzędzie z graficznym interfejsem użytkownika, które w znacznym stopniu upraszcza korzystanie z jego funkcji. W przypadku budowanej aplikacji serwer bazodanowy został utworzony na maszynie lokalnej, na której działa również aplikacja automatyzująca testy. Na potrzeby przeprowadzanych doświadczeń rozwiązanie to jest wystarczające przez fakt braku konieczności posiadania bardzo szybkiego środowiska, które wymaga uiszczenia opłat adekwatnych do rodzaju wykupywanego serwera.

5.2.2. Budowa bazy danych użytej w projekcie

Przed rozpoczęciem etapu implementacyjnego oraz po zakończeniu analizy dotyczącej wyboru wzorca architektonicznego aplikacji, zaprojektowana została struktura tabel bazodanowych wraz ze ich wszystkimi dostępnymi kolumnami oraz typami, a także połączenia i relacje między nimi. Proces projektowania bazy danych jest procesem bardzo ważnym, a jednocześnie często bagatelizowanym. W sytuacji nieprzemyślanego zaprojektowania bazy danych istnieje możliwość pojawienia się problemu z jej rozbudową i dodawaniem kolejnych potrzebnych tabel, na których miałyby operować nowe funkcjonalności aplikacji. Nieprzemyślana struktura tabel może doprowadzić do zbędnego powielania danych, której konsekwencją będzie bardzo szybko powiększający się rozmiar bazy danych. Kolejnym efektem złe zaprojektowanej bazy może być spowolniona praca i długie wykonywanie zapytań. W celu uniknięcia przedstawionych problemów, dokonana została szczegółowa analiza wymaganych funkcjonalności oraz danych, które one zwracają. Wykorzystana w budowanej aplikacji baza danych może zostać podzielona na dwa główne składniki - część słownikową zawierającą opisy wybranych algorytmów metaheurystycznych oraz funkcji testowych, a także część dotyczącą przeprowadzanych badań oraz wyników przez nie otrzymanych.

Część słownikowa

Na słownikową część bazy danych składają się trzy tabele przedstawione na rysunku 13, które zawierają dane wejściowe niezbędne w aplikacji: *Alghoritm*s, *TestFunctions* oraz *ExitFlags*.

Tabela *Alghoritm*s jest prostą tabelą słownikową zawierającą identyfikator oraz nazwę algorytmu metaheurystycznego. Lista algorytmów dostępnych do przetestowania w aplikacji znajduje się właśnie w tej tabeli. Na podstawie nazwy algorytmu, dopasowywana jest jej funkcja Matlabowa, w której tkwi główna logika algorytmu. W celu dodania kolejnego algorytmu, należy dodać wpis w opisywanej tabeli oraz według odpowiedniego schematu dostępnego do wglądu w klasie *FunctionNameMatcher* stworzyć funkcję w aplikacji Matlab. Tak utworzony schemat algorytmu należy kolejno dodać do warstwy ViewModelu oraz dostosować logikę okna użytkownika do wymaganych przez algorytm parametrów.

TestFunctions jest to tabela, która opisuje funkcje testowe na bazie których

| Algorithms | | | |
|------------|-------------|---------------|--------------------------|
| | Column Name | Data Type | Allow Nulls |
| 🔑 | Id | int | <input type="checkbox"/> |
| | Name | nvarchar(MAX) | <input type="checkbox"/> |
| | | | <input type="checkbox"/> |

| ExitFlags | | | |
|-----------|-------------|---------------|--------------------------|
| | Column Name | Data Type | Allow Nulls |
| 🔑 | Id | int | <input type="checkbox"/> |
| | Description | nvarchar(MAX) | <input type="checkbox"/> |
| | | | <input type="checkbox"/> |

| TestFunctions | | | |
|---------------|-------------------|---------------|--------------------------|
| | Column Name | Data Type | Allow Nulls |
| 🔑 | Id | int | <input type="checkbox"/> |
| | Name | nvarchar(MAX) | <input type="checkbox"/> |
| | DifficultyLevel | int | <input type="checkbox"/> |
| | FunctionGraphLink | nvarchar(MAX) | <input type="checkbox"/> |
| | BoundRange | int | <input type="checkbox"/> |
| | Dimension | int | <input type="checkbox"/> |
| | | | <input type="checkbox"/> |

Rysunek 13: Przetawienie części słownikowej bazy danych, Źródło: Opracowanie własne.

dokonywane są testy algorytmów. Zawarte w niej są takie informacje jak identyfikator funkcji, jej nazwa, poziom trudności ustalony na bazie stopnia skomplikowania funkcji testowej zaczerpnięty z [2], odnośnik do wykresu danej funkcji wraz z informacjami na jej temat, zakres płaszczyzny w obrębie której ma być poszukiwana wartość minimalna oraz wymiar funkcji czyli liczba zmiennych, z której owa funkcja się składa. W celu dodania do aplikacji kolejnej funkcji testowej należy tylko dodać jej definicję w opisywanej tabeli oraz utworzyć plik Matlabowy zawierający jej wzór. Nazwa pliku powinna zgadzać się ze schematem zawartym w klasie dopasowującej - *FunctionNameMatcher*. Tak utworzony algorytm automatycznie będzie dostępny do wyboru z poziomu aplikacji.

Ostatnia tabela z części słownikowej bazy danych to tabela *ExitFlag* w której zdefiniowane są numery flag wyjścia z Matlabowych funkcji oraz ich opis. Przeznaczeniem jej jest uzyskanie informacji dotyczącej powodu zakończenia pracy danego algorytmu. Powodem tym może być dla przykładu osiągnięcie maksymalnego założonego czasu działania algorytmu, liczby iteracji lub liczby odwołań do funkcji testowej.

Część dotycząca przeprowadzonych badań

Część bazy danych odnosząca się do przeprowadzanych badań jest częścią bardziej rozbudowaną niż część słownikowa. Zawiera ona wszelkie informacje dotyczące parametrów wejściowych danego testu oraz rezultaty przeprowadzanych badań. Główną tabelą, która jest swojego rodzaju trzonem bazy danych jest tabela *Re-*

sults. Po przeprowadzeniu pojedynczego doświadczenia w tabeli tej tworzony jest wpis, który zawiera identyfikatory wierszy odnoszące się do pozostałych tabel, które umożliwiają uzyskanie wszelkich informacji o doświadczeniu.

W tabeli *InputParameters* zawarte są informacje o parametrach wejściowych algorytmów takich jak maksymalny czas działania algorytmu, maksymalna liczba iteracji, dopuszczalna liczba odwołań do funkcji testowej oraz w zależności od wybranego algorytmu wielkość roju lub wielkość populacji.

Rezultaty przeprowadzonego badania umiejscawiane są w tabeli *ResultDetail* (TODO: sprawdź nazwę), w której to można uzyskać informację na temat liczby przeprowadzonych iteracji, liczby odwołań do funkcji testowej, najmniejszej znalezionej wartości funkcji, czasu działania algorytmu oraz informację o powodzeniu jego wykonania. Dodatkowym elementem, który przechowywany jest w bazie jest lokalizacja punktu posiadającego najmniejszą znaną wartość funkcji. Początkowo temat pracy magisterskiej zakładał porównanie algorytmów metaheurystycznych bazując wyłącznie na funkcjach dwóch zmiennych. Lokalizacja najlepszego punktu przechowywana była w dwóch kolumnach tabeli *ResultDetails*. Po zmianie wymagań, które dotyczyły konieczności obsłużenia funkcji wielu zmiennych musiała nastąpić modyfikacja struktury bazy danych. Wartym wspomnienia są tutaj fazy dokładnej analizy potrzeb, które baza danych ma spełniać oraz proces jej projektowania. Już na tym etapie jako jedno z ryzyk projektowych założone zostało właśnie zmienienie podejścia do liczby wymiarów funkcji testowych. W celu zapewnienia prawidłowego przetwarzania i przechowywania danych została w tym celu stworzona nowa tabela - *Points*. Tabela ta jest w relacji wiele do jednego z tabelą *ResultDetails* oraz umożliwia przechowywanie wartości punktu dla każdego wymiaru jako oddzielny wpis. Dla przykładu lokalizacja punktu dla funkcji pięciu zmiennych przechowywana będzie w postaci pięciu wpisów w omawianej tabeli, z których każdy opisywać będzie jego lokalizację oraz numer wymiaru do którego ma odniesienie.

5.3. Komunikacja bazy danych z projektem programistycznym

Umożliwienie kontaktu aplikacji z bazą daną jest kluczową kwestią, która odnosi się do prawidłowego działania bazodanowego projektu programistycznego. Bezpieczne, stabilne oraz ciągle połączenie umożliwia pozbawione problemów przesyłanie danych z gwarancją ich niezmienności oraz odpowiednią informacją w momencie

pojawienia się błędu komunikacyjnego. W technologii .NET, w której implementowana jest aplikacja bazodanowa możliwe jest kilka podejść do omawianego zagadnienia. Jednym z nich jest standardowe podejście udostępniane przez ADO.NET oraz nowsze, które dotyczy komunikacji na poziomie obiektów.

5.3.1. Mapowanie obiektowo-relacyjne

Mapowanie obiektowo-relacyjne (ang. ORM - Object-Relational Mapping) jest to pojęcie, które dotyczy informatycznego terminu odnoszącego się do współpracy z bazą danych przy wykorzystaniu programowania obiektowego. Dokładniej chodzi tutaj o translację danych, które przechowywane są w tabelach bazodanowych w relacyjnej bazie danych na postać obiektową, która dostępna jest z poziomu projektu programistycznego. Przedstawiony sposób komunikacji jest sposobem bardzo użytecznym przez wzgląd na brak konieczności analizy struktury bazy danych przez programistę oraz umożliwienie pobierania oraz zapisu danych z poziomu kodu, unikając bazodanowego języka zapytań SQL. Aplikacje komputerowe, które używane są przez wielu użytkowników oraz wymagają pobierania i zapisywania dużych ilości danych często korzystają z serwerów bazodanowych. Serwery te pozwalają na przechowywanie danych bez żadnych przerw oraz ich szybkie wyszukiwanie, dodawanie oraz edycję. Aspekt ten jest bardzo trudny do osiągnięcia korzystając wyłącznie ze standardowych aplikacji. W technologii .NET, standardowa komunikacja z bazą danych przebiega za pomocą protokołu ADO.NET. Przy jego zastosowaniu, w celu pobrania danych, programista musi za każdym razem połączyć się z bazą danych, manualnie zapisać zapytanie SQL, odebrać wynik, zapisać go w zmiennej oraz zamknąć połączenie. Schemat ten bardzo komplikuje kwestię utrzymania aplikacji w sytuacji modyfikacji struktury bazy danych. Zapisane przez programistę zapytanie SQL nie jest w ogóle walidowane pod kątem poprawności jego formy, a w sytuacji edycji kolumny bazodanowej, następuje obowiązek uaktualnienia wszystkich zapytań, które jej dotyczyły. Przykładowy schemat pracy podczas korzystania z ADO.NET wraz z porównaniem analogicznej operacji w technologii ORM zaprezentowane zostało na poziomie kodu źródłowego (TODO: dodać numer)

tututu

W implementowanej aplikacji dotyczącej prezentowanej pracy magisterskiej, zdecydowano się przeciwdziałać opisanym problemom, stosując framework, który w znacznym stopniu ułatwia programiście zastosowanie wszystkich operacji na pozio-

mie komunikacji z bazą danych.

5.3.2. Zastosowane narzędzie ORM – Entity Framework

Entity Framework jest to dedykowane dla platformy .NET narzędzie mapowania obiektowo-relacyjnego, którego zadaniem jest wspieranie budowy trójwarstwowych aplikacji bazodanowych. Budowa obiektowego modelu bazy danych w opisywanym narzędziu może być wykonana na trzy sposoby.

Pierwszą metodą jest podejście *Database First*, które jest użyteczne w sytuacji, w której dostępna jest działająca fizycznie baza danych. Korzystając z dostępnego w frameworku kreatora, istnieje możliwość określenia lokalizacji bazy oraz poddania jej automatycznemu mapowaniu, w wyniku którego wygenerowane zostają niezbędne klasy obiektowego modelu bazy danych.

Kolejnym sposobem jest *Code First*, który pozwala na utworzenie fizycznego modelu bazy danych na podstawie klas napisanych w języku C# oraz ich odpowiednimi przypisami.

Ostatnią metodą jest *Model First*, które polega na utworzeniu fizycznego modelu bazy danych posługując się wbudowanym w Entity Framework Designerem, który w dużym stopniu upraszcza tworzenie właściwości oraz encji bazodanowych. Ścieżka ta stosowana jest, gdy posiadany jest wyłącznie sam schemat bazy danych. Ze zbudowanego modelu następuje wygenerowanie fizycznego modelu bazy danych jak i również klas, które reprezentują model obiektowy. Sposób ten został zastosowany w prezentowanej pracy magisterskiej. Po analizie wymagań i utworzeniu schematu bazy danych, został on przeniesiony na poziom Designera opisywanego frameworka, który automatycznie stworzył skrypt napisany w języku SQL po wykonaniu którego utworzony został fizyczny model bazy danych, a także wygenerował wszystkie niezbędne klasy reprezentujące model obiektowy zadanego schematu bazy.

5.4. Komunikacja Matlab z projektem programistycznym

Poprawne skomunikowanie aplikacji bazodanowej ze środowiskiem obliczeniowym Matlab było najtrudniejszą częścią podczas całej pracy implementacyjnej. Podczas tworzenia architektury budowanej aplikacji, poszukiwany był efektywny sposób komunikacji środowiska .NET ze środowiskiem obliczeniowym Matlab. Przez fakt tego, iż systemy te utworzone zostały przez różnych producentów, nie ma pro-

stego sposobu na ich sprzężenie. Poszukiwania doprowadziły do zastosowania biblioteki dedykowanej dla środowiska .NET opisanej w rozdziale 4.3.1, która łączy się z pośredniczącym serwerem Matlaba. Serwer ten nie jest domyślnie dodawany przy instalacji środowiska obliczeniowego, dlatego też nachodzi konieczność jego dodatkowego doinstalowania. Biblioteka umożliwia stworzenie żądania na poziomie C#, które następnie przekazywane jest do serwera pośredniczącego tłumaczącego żądanie na język Matlaba. Żądanie to jest wykonywane przez środowisko obliczeniowe, a jego rezultat analogiczną ścieżką jest przekazywany do aplikacji bazodanowej. Sporym problemem okazało się prawidłowe tworzenie żądań przez fakt na bardzo ograniczoną dokumentację biblioteki i małą liczbę przykładów. Na podstawie metody prób i błędów udało się jednak utworzyć prawidłowy typ żądań oraz w sposób właściwy dokonywać odbioru ich odpowiedzi. Żądanie w kodzie źródłowym numer (TODO: dodać alias kodu oraz kod metody wykonującej żądanie) prezentuje sposób wykonania pojedynczego doświadczenia dotyczącego algorytmu symulowanego wyżarzania, natomiast kod źródłowy numer (TODO: dodać alias kodu oraz kod metody parsującej wynik) ukazuje sposób przetwarzania otrzymanej odpowiedzi.

6. Badania eksperymentalne

Cały proces poczynając od pomysłu automatyzacji przeprowadzanych badań zawarty w rozdziale 3, przez analizę rozwiązań technologicznych opisanych w rozdziale 4, po budowę architektury oraz samej aplikacji bazodanowej ujęty w rozdziale 5 miał za zadanie ułatwić przeprowadzenie eksperymentalnych badań porównawczych wybranych algorytmów metaheurystycznych. W obecnym rozdziale przedstawiony został opis metod porównawczych, a także samych badań doświadczalnych wraz z ich zagregowanymi wynikami zawartymi w 6.3.

6.1. Metody porównawcze wybranych algorytmów

W celu zrównania szans każdego z porównywanych algorytmów, wszystkie przeprowadzane testy dokonane zostały na tej samej maszynie obliczeniowej przy tych samych warunkach obciążeniowych systemu. Pierwszym kryterium porównawczym był maksymalny czas, w którym algorytm poszukuje najlepszego rezultatu. Parametr ten bierze pod uwagę kilka różnych ustalonych wartości dla których przeprowadzana została próbka doświadczeń, z których wartości średnie zostały zestawione i porównane w obrębie wszystkich algorytmów oraz funkcji testowych. Drugim kryterium porównawczym była liczba odwołań algorytmu do zadanej funkcji testowej. W kryterium tym porównane zostały, w sposób analogiczny jak w kryterium pierwszym, rezultaty otrzymane przez algorytmy przy takim założeniu, iż mogły one wyliczyć wartość funkcji testowej określoną liczbę razy.

6.2. Opis przeprowadzonych badań

6.2.1. Skalowanie algorytmów

Przez wzgląd na dużą liczbę parametrów każdego porównywanego algorytmu heurystycznego, zdecydowano się dokonać ich dostrojenia przed przeprowadzeniem zaplanowanych badań. Na bazie wybranych funkcji testowych wykonano liczne testy, na bazie których wybrano parametry, dla których uzyskano najlepsze rezultaty. Dla

algorytmu symulowanego wyżarzania jako metoda schładzająca, wybrana została standardowa funkcja

$$NowaTemperatura = ObecnaTemperatura * 0.95^k \quad (6)$$

, gdzie k jest numerem iteracji od wyżarzenia. Uwzględniając fakt dużej liczby iteracji, którą algorytm może wykonać podczas testów, zdecydowano się wprowadzić możliwość wystąpienia operacji ponownego wyżarzenia (ang. reannealing), w czasie której algorytm ponownie ustala temperaturę na domyślną wartość 100 i kontynuuje pracę. Wartość parametru ponownego wyżarzania ustalona została na poziomie 40 i definiuje liczbę punktów, które algorytm ma obowiązek zaakceptować przed wystąpieniem ponownego wyżarzenia. W metodzie roju cząstek domyślne parametry wzoru (TODO: numer wzoru 1) zawartego w rozdziale 2.2.1 ustalone zostały na następujące wartości: $W = 0.6$, $c_1 = c_2 = 0.75$. Algorytm genetyczny z kolei wymagał największej ilości testów skalujących przez fakt posiadania największej liczby możliwych kombinacji parametrów. Na ich podstawie wybrano metodę ruletki jako metodę selekcji, zastępowanie elitarne o wielkości elity równej $0.05 * WielkoscPopulacji$ jako metodę wymiany pokoleń oraz metodę krzyżowania jednopunktowego.

6.2.2. Przeprowadzenie doświadczeń w zaimplementowanym środowisku testowym

W zaimplementowanej aplikacji dla każdego opisanego w 6.1 kryterium, umieszczone zostały ustalone wartości parametrów, po czym przeprowadzana była operacja uruchomienia systemu. Dla kryterium maksymalnego czasu, w którym algorytmy poszukują globalnego minimum funkcji, doświadczenia wykonywane zostały dla zakresu od jednej do pięciu sekund, z jednosekundowym interwałem. Liczba testów dla każdej wartości z ustalonego zakresu ustalona została na poziomie stu prób. Pozostałe parametry, takie jak maksymalna liczba iteracji oraz odwołań do funkcji wraz z maksymalną liczbą iteracji bez poprawy wyniku, ustalone zostały na niemożliwą do osiągnięcia wartość 999999. Spowodowane jest to tym, iż jednym warunkiem stopu w przedstawianej metodzie porównawczej jest maksymalny ustalony czas działania algorytmu. W przypadku algorytmu genetycznego oraz roju cząstek dodatkowo przeprowadzone zostały doświadczenia dla różnych wielkości populacji i roju, dzięki czemu możliwa do wykonania była analiza wpływu tych parametrów na otrzymane rezultaty. Przeprowadzenie badań dotyczących kryterium maksymalnej

liczby odwołań do funkcji testowej przebiegało w sposób analogiczny, z tą różnicą, iż niemożliwą do osiągnięcia wartością 999999 został objęty maksymalny czas poszukiwań, który w opisywanym kryterium nie jest brany pod uwagę. Dla kryterium tego maksymalna liczba odwołań do funkcji ustalona została na wartości: 1000, 2000, 3000, 4000, 5000, 6000.

6.3. Wyniki doświadczeń dla zadanych funkcji testowych

Aktualny rozdział prezentuje zagregowane wyniki przeprowadzanych doświadczeń dla funkcji testowych opisanych w rozdziale 1.2. Prezentacja wyników dla każdej funkcji testowej podzielona została na dwie części przez wzgląd na dwa kryteria porównawcze algorytmów zdefiniowane w rozdziale 6.1. Wartym wspomnienia jest znaczenie kolorów tabel zawartych w poniższych podrozdziałach. Zielony kolor definiuje najlepsze otrzymane rezultaty w obrębie jednego testu, czerwony natomiast wyniki gorsze.

6.3.1. Funkcja Bohachevsky'ego 1

W tabeli 1 przedstawione zostały wyniki otrzymane przez algorytm genetyczny dla kryterium czasowego. Wynika z niej, iż najlepsze rezultaty uzyskane zostały dla wielkości populacji równej 100. Dla metody roju cząstek otrzymane rezultaty zaprezentowane zostały w tabeli 2, z której wynika, iż najlepsze wyniki uzyskane zostały dla wielkości roju równej 50 oraz 100. W tym jednak przypadku, jako najlepsze rezultaty wybrane zostały wyniki z wielkością roju równą 100, z powodu mniejszej liczby odwołań do funkcji testowej niż w przypadku roju o wielkości 50.

Najlepsze rezultaty algorytmu genetycznego oraz roju cząstek zostały zestawione wraz z rezultatami symulowanego wyżarzania w tabeli 3. Wynika z niej, iż najlepsze wyniki dla każdej wartości parametru maksymalnego czasu uzyskała metoda roju cząstek, która w każdym przypadku znalazła minimum globalne dla funkcji Bohachevsky'ego, którego wartość wynosi 0. Na drugim miejscu uplasował się algorytm genetyczny, którego wyniki oscylowały wokół minimum globalnego, jednak w ani jednym przypadku go nie osiągnęły. Algorytm symulowanego wyżarzania uzyskał lepszy rezultat od algorytmu genetycznego tylko w przypadku testów dotyczących maksymalnego czasu działania algorytmu równego jednej sekundzie. W pozostałych przypadkach uzyskał najgorsze rezultaty ze wszystkich porównywanych algorytmów

metaheurystycznych.

| Maksymalny czas | Algorytm Genetyczny | | |
|--------------------|-----------------------|--------------------------------|------------------------------|
| | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | 0,50533 | 8954 |
| 2 | 10 | 0,27054 | 17538 |
| 3 | 10 | 0,25915 | 24586 |
| 4 | 10 | 0,35312 | 34576 |
| 5 | 10 | 0,43571 | 43202 |
| 6 | 10 | 0,81062 | 52668 |
| 1 | 50 | 0,08259 | 15110 |
| 2 | 50 | 0,16517 | 29850 |
| 3 | 50 | 0,00000 | 45130 |
| 4 | 50 | 0,00000 | 60150 |
| 5 | 50 | 0,08259 | 75440 |
| 6 | 50 | 0,00000 | 90160 |
| 1 | 100 | 0,07656 | 16100 |
| 2 | 100 | 0,00003 | 31660 |
| 3 | 100 | 0,00003 | 46740 |
| 4 | 100 | 0,00001 | 60200 |
| 5 | 100 | 0,00001 | 74320 |
| 6 | 100 | 0,00012 | 92580 |

Tabela 1: Zestawienie wyników algorytmu genetycznego dla funkcji Bohachevsky'ego
- kryterium czasowe, Źródło: Opracowanie własne.

| | Metoda Roju Cząstek | | |
|-----------------|---------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | 0,00002 | 27692 |
| 2 | 10 | 0,08259 | 57212 |
| 3 | 10 | 0,00003 | 83842 |
| 4 | 10 | 0,00002 | 119112 |
| 5 | 10 | 0,00001 | 148196 |
| 6 | 10 | 0,00002 | 177158 |
| 1 | 50 | 0,00000 | 41560 |
| 2 | 50 | 0,00000 | 83860 |
| 3 | 50 | 0,00000 | 126240 |
| 4 | 50 | 0,00000 | 170930 |
| 5 | 50 | 0,00000 | 214920 |
| 6 | 50 | 0,00000 | 259030 |
| 1 | 100 | 0,00000 | 37660 |
| 2 | 100 | 0,00000 | 73460 |
| 3 | 100 | 0,00000 | 102800 |
| 4 | 100 | 0,00000 | 132160 |
| 5 | 100 | 0,00000 | 180540 |
| 6 | 100 | 0,00000 | 215840 |

Tabela 2: Zestawienie wyników metody roju cząstek dla funkcji Bohachevsky'ego - kryterium czasowe, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|-----------------------|-----------------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wartość znalezionej minimum | | Liczba odwołań do funkcji |
| 1 | 0,01978 | | 2038 |
| 2 | 0,00950 | | 3937 |
| 3 | 0,00211 | | 5958 |
| 4 | 0,00349 | | 7755 |
| 5 | 0,00095 | | 9713 |
| 6 | 0,00024 | | 11644 |
| Algorytm Genetyczny | | | |
| Maksymalny czas | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 100 | 0,17656 | 16100 |
| 2 | 100 | 0,00003 | 31660 |
| 3 | 100 | 0,00003 | 46740 |
| 4 | 100 | 0,00001 | 60200 |
| 5 | 100 | 0,00001 | 74320 |
| 6 | 100 | 0,00012 | 92580 |
| Metoda Roju Częstek | | | |
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 100 | 0,00000 | 37660 |
| 2 | 100 | 0,00000 | 73460 |
| 3 | 100 | 0,00000 | 102800 |
| 4 | 100 | 0,00000 | 132160 |
| 5 | 100 | 0,00000 | 180540 |
| 6 | 100 | 0,00000 | 215840 |

Tabela 3: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Bohachevsky'ego - kryterium czasowe, Źródło: Opracowanie własne.

W przypadku kryterium dotyczącym określonej maksymalnej liczby odwołań algorytmu do funkcji testowej, otrzymane rezultaty są bardzo zbliżone do rezultatów poprzedniego kryterium. Algorytm genetyczny najlepsze wyniki uzyskał dla wielkości populacji równej 100, co można zaobserwować na wykresie 4. Metoda roju cząstek, według wykresu 5, najlepsze rezultaty zanotowała dla wielkości roju równego 50, co jest inną wartością niż w przypadku kryterium czasowego. Wyniki te zostały zestawione w tabeli 6 z rezultatami otrzymanymi przez algorytm symulowanego wyżarzania, z której wynika iż ponownie dla każdej wartości zadanego kryterium, najlepsza okazała się metoda roju cząstek.

| Liczba odwołań do funkcji | Algorytm Genetyczny | | |
|------------------------------|-----------------------|--------------------------------|------------------|
| | Wielkość populacji | Wartość znalezionej minimum | Liczba generacji |
| 1000 | 10 | 0,27054 | 100 |
| 2000 | 10 | 0,58187 | 200 |
| 3000 | 10 | 0,25345 | 300 |
| 4000 | 10 | 0,35312 | 400 |
| 5000 | 10 | 0,22355 | 500 |
| 6000 | 10 | 0,08828 | 600 |
| 1000 | 50 | 0,05025 | 20 |
| 2000 | 50 | 0,04699 | 40 |
| 3000 | 50 | 0,08828 | 60 |
| 4000 | 50 | 0,22875 | 80 |
| 5000 | 50 | 0,00000 | 100 |
| 6000 | 50 | 0,04699 | 120 |
| 1000 | 100 | 0,03825 | 10 |
| 2000 | 100 | 0,00374 | 20 |
| 3000 | 100 | 0,00016 | 30 |
| 4000 | 100 | 0,00000 | 40 |
| 5000 | 100 | 0,00000 | 50 |
| 6000 | 100 | 0,00000 | 60 |

Tabela 4: Zestawienie wyników algorytmu genetycznego dla funkcji Bohachevsky'ego - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne., Źródło: Opracowanie własne.

| Liczba odwołań do funkcji | Metoda Roju Cząstek | | |
|------------------------------|---------------------|---------------------------------|-----------------|
| | Wielkość roju | Wartość znalezionego minimum | Liczba iteracji |
| 1000 | 10 | 0,09398 | 100 |
| 2000 | 10 | 0,00000 | 200 |
| 3000 | 10 | 0,04699 | 300 |
| 4000 | 10 | 0,00000 | 400 |
| 5000 | 10 | 0,00000 | 500 |
| 6000 | 10 | 0,00000 | 600 |
| 1000 | 50 | 0,02503 | 20 |
| 2000 | 50 | 0,00000 | 40 |
| 3000 | 50 | 0,00000 | 60 |
| 4000 | 50 | 0,00000 | 80 |
| 5000 | 50 | 0,00000 | 100 |
| 6000 | 50 | 0,00000 | 120 |
| 1000 | 100 | 0,37419 | 10 |
| 2000 | 100 | 0,00316 | 20 |
| 3000 | 100 | 0,00003 | 30 |
| 4000 | 100 | 0,00000 | 40 |
| 5000 | 100 | 0,00000 | 50 |
| 6000 | 100 | 0,00000 | 60 |

Tabela 5: Zestawienie wyników metody roju cząstek dla funkcji Bohachevsky'ego - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|---------------------------|----------------------------|----------------------------|------------------|
| Liczba odwołań do funkcji | Wartość znalezione minimum | | Liczba iteracji |
| 1000 | 0,15152 | | 1000 |
| 2000 | 0,04706 | | 2000 |
| 3000 | 0,00492 | | 3000 |
| 4000 | 0,00325 | | 4000 |
| 5000 | 0,00230 | | 5000 |
| 6000 | 0,00217 | | 6000 |
| Algorytm Genetyczny | | | |
| Liczba odwołań do funkcji | Wielkość populacji | Wartość znalezione minimum | Liczba generacji |
| 1000 | 100 | 0,03825 | 10 |
| 2000 | 100 | 0,00374 | 20 |
| 3000 | 100 | 0,00016 | 30 |
| 4000 | 100 | 0,00000 | 40 |
| 5000 | 100 | 0,00000 | 50 |
| 6000 | 100 | 0,00000 | 60 |
| Metoda Roju Częstek | | | |
| Liczba odwołań do funkcji | Wielkość roju | Wartość znalezione minimum | Liczba iteracji |
| 1000 | 50 | 0,02503 | 20 |
| 2000 | 50 | 0,00000 | 40 |
| 3000 | 50 | 0,00000 | 60 |
| 4000 | 50 | 0,00000 | 80 |
| 5000 | 50 | 0,00000 | 100 |
| 6000 | 50 | 0,00000 | 120 |

Tabela 6: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Bohachevsky'ego - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

6.3.2. Funkcja Colville'a

Dla funkcji Colville'a, wyniki testów sprawdzających optymalną wielkość populacji dla algorytmu genetycznego przedstawione zostały w tabeli 7, z której wynika, iż najbliżej minimum globalnego dla kryterium czasowego uplasowały się rezultaty z wielkością populacji równą 100. Taka sama wartość została ustalona dla analogicznego testu sprawdzającego optymalną wielkość roju dla metody roju cząstek

widocznego na wykresie 8. Wyniki obu testów zostały zestawione z rezultatami otrzymanymi przez algorytm symulowanego wyżarzania na wykresie 9. Z wykresu tego wynika, iż najbliższej wartości 0, która jest minimum globalnym badanej funkcji, uplasowały się rezultaty otrzymane przez metodę roju cząstek. Wyniki otrzymane przez algorytm genetyczny uplasowały go na drugim miejscu, przed algorytmem symulowanego wyżarzania, którego rezultaty były bardzo odległe od poszukiwanego minimum globalnego.

| Maksymalny czas | Algorytm Genetyczny | | |
|----------------------------|-------------------------------|--|--------------------------------------|
| | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | 1,01206 | 8800 |
| 2 | 10 | 0,64249 | 17748 |
| 3 | 10 | 2,44841 | 26264 |
| 4 | 10 | 0,10143 | 34958 |
| 5 | 10 | 1,11109 | 44078 |
| 6 | 10 | 0,19459 | 52572 |
| 1 | 50 | 2,21722 | 14340 |
| 2 | 50 | 0,46911 | 29940 |
| 3 | 50 | 0,10756 | 43730 |
| 4 | 50 | 0,10229 | 59420 |
| 5 | 50 | 0,06617 | 74080 |
| 6 | 50 | 1,57922 | 88530 |
| 1 | 100 | 0,37593 | 15560 |
| 2 | 100 | 0,15028 | 31020 |
| 3 | 100 | 0,04512 | 46540 |
| 4 | 100 | 0,08684 | 61880 |
| 5 | 100 | 0,05409 | 77520 |
| 6 | 100 | 0,05227 | 93120 |

Tabela 7: Zestawienie wyników algorytmu genetycznego dla funkcji Colville'a - kryterium czasowe, Źródło: Opracowanie własne.

| | Metoda Roju Cząstek | | |
|-----------------|---------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | 0,04142 | 27380 |
| 2 | 10 | 0,00701 | 55344 |
| 3 | 10 | 0,00009 | 83208 |
| 4 | 10 | 0,00002 | 109760 |
| 5 | 10 | 0,00000 | 139726 |
| 6 | 10 | 0,00000 | 167042 |
| 1 | 50 | 0,00243 | 46050 |
| 2 | 50 | 0,01416 | 92180 |
| 3 | 50 | 0,00005 | 136090 |
| 4 | 50 | 0,00005 | 181070 |
| 5 | 50 | 0,00000 | 228990 |
| 6 | 50 | 0,00000 | 276880 |
| 1 | 100 | 0,00241 | 42060 |
| 2 | 100 | 0,00023 | 85780 |
| 3 | 100 | 0,00002 | 128740 |
| 4 | 100 | 0,00003 | 170920 |
| 5 | 100 | 0,00000 | 213840 |
| 6 | 100 | 0,00000 | 261920 |

Tabela 8: Zestawienie wyników metody roju cząstek dla funkcji Colville'a - kryterium czasowe, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|-----------------------|-----------------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wartość znalezionej minimum | | Liczba odwołań do funkcji |
| 1 | 114,10349 | | 1789 |
| 2 | 58,08332 | | 3552 |
| 3 | 21,16509 | | 5197 |
| 4 | 10,04012 | | 6959 |
| 5 | 1,23721 | | 8606 |
| 6 | 0,19772 | | 10247 |
| Algorytm Genetyczny | | | |
| Maksymalny czas | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 100 | 0,37593 | 15560 |
| 2 | 100 | 0,15028 | 31020 |
| 3 | 100 | 0,04512 | 46540 |
| 4 | 100 | 0,08684 | 61880 |
| 5 | 100 | 0,05409 | 77520 |
| 6 | 100 | 0,05227 | 93120 |
| Metoda Roju Częstek | | | |
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 100 | 0,00241 | 42060 |
| 2 | 100 | 0,00023 | 85780 |
| 3 | 100 | 0,00002 | 128740 |
| 4 | 100 | 0,00003 | 170920 |
| 5 | 100 | 0,00000 | 213840 |
| 6 | 100 | 0,00000 | 261920 |

Tabela 9: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Colville'a - kryterium czasowe, Źródło: Opracowanie własne.

Rezultaty dotyczące testów dla kryterium odnoszącego się do maksymalnej liczby odwołań algorytmu do funkcji testowej okazały się bardzo ciekawe przez wzgląd na uzyskanie najlepszych rezultatów przez inny algorytm niż w kryterium poprzednim. Do podsumowującego testu wszystkich trzech algorytmów metaheurystycznych zostały wybrane dla algorytmu genetycznego rezultaty przedstawione w tabeli 10 z wielkością populacji równą 100, natomiast dla metody roju cząstek wyniki z wielkością roju równą 10, które dostępne są do wglądu w tabli 11. Podsumowanie widoczne w tabeli 12 pozwala dostrzec, iż najlepsze wyniki uzyskał algorytm genetyczny, który w testach pierwszego kryterium uzyskał drugi rezultat. Porównując wyniki dla obu kryteriów porównawczych z tabel 9 oraz 12, widoczna jest zależność wyjaśniająca taki efekt. Metoda roju cząstek przeprowadza znacznie więcej odwołań do badanej funkcji Colville'a niż algorytm genetyczny dla tego samego czasu pracy algorytmów. Z tego powodu przy limicie owych odwołań w kryterium drugim, wyniki otrzyma-

ne przez algorytm genetyczny są lepsze. Wartym wspomnienia jest także to, iż w analizowanym przykładzie wyniki uzyskane przez metodę roju cząstek są nie tylko gorsze od wyników uzyskanych przez algorytm genetyczny, lecz także przez symulowane wyżarzanie, które to uzyskało drugi rezultat dla każdego badanego parametru maksymalnej liczby odwołań do funkcji testowej.

| Liczba odwołań do funkcji | Algorytm Genetyczny | | |
|---------------------------|---------------------|----------------------------|------------------|
| | Wielkość populacji | Wartość znalezione minimum | Liczba generacji |
| 1000 | 10 | 3,20273 | 100 |
| 2000 | 10 | 3,94219 | 200 |
| 3000 | 10 | 5,05571 | 300 |
| 4000 | 10 | 2,25915 | 400 |
| 5000 | 10 | 2,17518 | 500 |
| 6000 | 10 | 1,68003 | 600 |
| 1000 | 50 | 3,60476 | 20 |
| 2000 | 50 | 1,80067 | 40 |
| 3000 | 50 | 1,59701 | 60 |
| 4000 | 50 | 2,04576 | 80 |
| 5000 | 50 | 0,94703 | 100 |
| 6000 | 50 | 2,45713 | 120 |
| 1000 | 100 | 2,54692 | 10 |
| 2000 | 100 | 1,47420 | 20 |
| 3000 | 100 | 1,44208 | 30 |
| 4000 | 100 | 1,25604 | 40 |
| 5000 | 100 | 0,99649 | 50 |
| 6000 | 100 | 0,66766 | 60 |

Tabela 10: Zestawienie wyników algorytmu genetycznego dla funkcji Colville'a - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne., Źródło: Opracowanie własne.

| | Metoda Roju Cząstek | | |
|---------------------------|---------------------|-----------------------------|-----------------|
| Liczba odwołań do funkcji | Wielkość roju | Wartość znalezionej minimum | Liczba iteracji |
| 1000 | 10 | 399,68527 | 100 |
| 2000 | 10 | 148,28911 | 200 |
| 3000 | 10 | 2,29084 | 300 |
| 4000 | 10 | 1,51750 | 400 |
| 5000 | 10 | 1,21875 | 500 |
| 6000 | 10 | 1,19613 | 600 |
| 1000 | 50 | 2786,25419 | 20 |
| 2000 | 50 | 152,12862 | 40 |
| 3000 | 50 | 3,18737 | 60 |
| 4000 | 50 | 1,58407 | 80 |
| 5000 | 50 | 1,58627 | 100 |
| 6000 | 50 | 1,83847 | 120 |
| 1000 | 100 | 4289,10984 | 10 |
| 2000 | 100 | 1135,85957 | 20 |
| 3000 | 100 | 265,14082 | 30 |
| 4000 | 100 | 7,98078 | 40 |
| 5000 | 100 | 1,98679 | 50 |
| 6000 | 100 | 1,55706 | 60 |

Tabela 11: Zestawienie wyników metody roju cząstek dla funkcji Colville'a - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|---------------------------|-----------------------------|-----------------------------|------------------|
| Liczba odwołań do funkcji | Wartość znalezionej minimum | | Liczba iteracji |
| 1000 | 68,74295 | | 1000 |
| 2000 | 46,06927 | | 2000 |
| 3000 | 2,16472 | | 3000 |
| 4000 | 1,44973 | | 4000 |
| 5000 | 1,11868 | | 5000 |
| 6000 | 0,91541 | | 6000 |
| Algorytm Genetyczny | | | |
| Liczba odwołań do funkcji | Wielkość populacji | Wartość znalezionej minimum | Liczba generacji |
| 1000 | 100 | 2,54692 | 10 |
| 2000 | 100 | 1,47420 | 20 |
| 3000 | 100 | 1,44208 | 30 |
| 4000 | 100 | 1,25604 | 40 |
| 5000 | 100 | 0,99649 | 50 |
| 6000 | 100 | 0,66766 | 60 |
| Metoda Roju Częstek | | | |
| Liczba odwołań do funkcji | Wielkość roju | Wartość znalezionej minimum | Liczba iteracji |
| 1000 | 10 | 399,68527 | 100 |
| 2000 | 10 | 148,28911 | 200 |
| 3000 | 10 | 2,29084 | 300 |
| 4000 | 10 | 1,51750 | 400 |
| 5000 | 10 | 1,21875 | 500 |
| 6000 | 10 | 1,19613 | 600 |

Tabela 12: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Colville'a - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

6.3.3. Funkcja Ackley'a

Kolejną funkcją, która poddana została testom jest funkcja Ackleya, dla której algorytm genetyczny oraz metoda roju cząstek dla kryterium czasowego uzyskała najlepsze rezultaty dla wielkości populacji oraz roju wynoszącej 50. Wyniki te, dostępne do wglądu w tabeli 13 oraz 14, zestawione zostały w tabeli 15 z wynikami otrzymanymi przez algorytm symulowanego wyżarzania. Z podsumowania

tego jednoznacznie wynika, iż dla każdego badanego parametru maksymalnego czasu algorytm genetyczny zwrócił najlepsze rezultaty, które to są bardzo zbliżone do minimum globalnego wynoszącego 0. Pozostałe algorytmy nawet nie zbliżyły się do tych rezultatów i utknęły minimach lokalnych funkcji.

| Maksymalny czas | Algorytm Genetyczny | | |
|-----------------|---------------------|-----------------------------|---------------------------|
| | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | 1,48507 | 8698 |
| 2 | 10 | 1,41709 | 17464 |
| 3 | 10 | 1,65735 | 26128 |
| 4 | 10 | 1,74368 | 34772 |
| 5 | 10 | 1,65734 | 42662 |
| 6 | 10 | 0,99338 | 52302 |
| 1 | 50 | 0,00003 | 14520 |
| 2 | 50 | 0,00001 | 27170 |
| 3 | 50 | 0,36800 | 37160 |
| 4 | 50 | 0,00001 | 54810 |
| 5 | 50 | 0,00000 | 66790 |
| 6 | 50 | 0,00000 | 75070 |
| 1 | 100 | 1,10400 | 15220 |
| 2 | 100 | 0,00001 | 31020 |
| 3 | 100 | 0,00001 | 46000 |
| 4 | 100 | 0,00004 | 61300 |
| 5 | 100 | 0,00000 | 76920 |
| 6 | 100 | 0,51599 | 93060 |

Tabela 13: Zestawienie wyników algorytmu genetycznego dla funkcji Ackley'a - kryterium czasowe, Źródło: Opracowanie własne.

| | Metoda Roju Cząstek | | |
|-----------------|---------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | 20,00000 | 28114 |
| 2 | 10 | 20,00000 | 55876 |
| 3 | 10 | 20,00000 | 84280 |
| 4 | 10 | 20,00000 | 112162 |
| 5 | 10 | 20,00000 | 141600 |
| 6 | 10 | 20,00000 | 171352 |
| 1 | 50 | 19,99956 | 39140 |
| 2 | 50 | 19,99956 | 69480 |
| 3 | 50 | 19,99956 | 108790 |
| 4 | 50 | 16,51610 | 156580 |
| 5 | 50 | 16,71478 | 196910 |
| 6 | 50 | 19,99956 | 238150 |
| 1 | 100 | 20,00000 | 33760 |
| 2 | 100 | 20,00000 | 70500 |
| 3 | 100 | 20,00000 | 105040 |
| 4 | 100 | 20,00000 | 139960 |
| 5 | 100 | 20,00000 | 176480 |
| 6 | 100 | 20,00000 | 212060 |

Tabela 14: Zestawienie wyników metody roju cząstek dla funkcji Ackley'a - kryterium czasowe, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|-----------------------|-----------------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wartość znalezionej minimum | | Liczba odwołań do funkcji |
| 1 | 19,99956 | | 2038 |
| 2 | 20,00000 | | 3937 |
| 3 | 20,00000 | | 5958 |
| 4 | 16,85489 | | 7755 |
| 5 | 19,84388 | | 9713 |
| 6 | 19,99997 | | 11644 |
| Algorytm Genetyczny | | | |
| Maksymalny czas | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 50 | 0,00003 | 14520 |
| 2 | 50 | 0,00001 | 27170 |
| 3 | 50 | 0,36800 | 37160 |
| 4 | 50 | 0,00001 | 54810 |
| 5 | 50 | 0,00000 | 66790 |
| 6 | 50 | 0,00000 | 75070 |
| Metoda Roju Cząstek | | | |
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 50 | 19,99956 | 39140 |
| 2 | 50 | 19,99956 | 69480 |
| 3 | 50 | 19,99956 | 108790 |
| 4 | 50 | 16,51610 | 156580 |
| 5 | 50 | 16,71478 | 196910 |
| 6 | 50 | 19,99956 | 238150 |

Tabela 15: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Ackley'a - kryterium czasowe, Źródło: Opracowanie własne.

Wyniki dotyczące kryterium limitu odwołań do funkcji testowej są analogiczne jak w przypadku pierwszego kryterium. Jediną różnicą są wielkości populacji oraz roju, których wyniki zostały wzięte do testu podsumowującego widocznego w tabeli 18. Funkcja Ackley'a jest pierwszą testowaną funkcją, do której minimum globalnego w obu badanych kryteriach zbliżył się tylko jeden porównywany algorytm metaheurystyczny - algorytm genetyczny.

| | Algorytm Genetyczny | | |
|---------------------------|---------------------|-----------------------------|------------------|
| Liczba odwołań do funkcji | Wielkość populacji | Wartość znalezionej minimum | Liczba generacji |
| 1000 | 10 | 1,48507 | 100 |
| 2000 | 10 | 1,41709 | 200 |
| 3000 | 10 | 1,65735 | 300 |
| 4000 | 10 | 1,74368 | 400 |
| 5000 | 10 | 1,65734 | 500 |
| 6000 | 10 | 0,99338 | 600 |
| 1000 | 50 | 1,44904 | 20 |
| 2000 | 50 | 0,35100 | 40 |
| 3000 | 50 | 0,00291 | 60 |
| 4000 | 50 | 0,18426 | 80 |
| 5000 | 50 | 0,44204 | 100 |
| 6000 | 50 | 0,00006 | 120 |
| 1000 | 100 | 0,61431 | 10 |
| 2000 | 100 | 0,30613 | 20 |
| 3000 | 100 | 0,00074 | 30 |
| 4000 | 100 | 0,01294 | 40 |
| 5000 | 100 | 0,00501 | 50 |
| 6000 | 100 | 0,00000 | 60 |

Tabela 16: Zestawienie wyników algorytmu genetycznego dla funkcji Ackley'a - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne., Źródło: Opracowanie własne.

| Liczba odwołań do | Metoda Roju Cząstek | | |
|----------------------|---------------------|--------------------------------|-----------------|
| | Wielkość roju | Wartość znalezionej minimum | Liczba iteracji |
| 1000 | 10 | 20,00000 | 100 |
| 2000 | 10 | 20,00000 | 200 |
| 3000 | 10 | 20,00000 | 300 |
| 4000 | 10 | 20,00000 | 400 |
| 5000 | 10 | 20,00000 | 500 |
| 6000 | 10 | 20,00000 | 600 |
| 1000 | 50 | 20,00000 | 20 |
| 2000 | 50 | 20,00000 | 40 |
| 3000 | 50 | 20,00000 | 60 |
| 4000 | 50 | 20,00000 | 80 |
| 5000 | 50 | 20,00000 | 100 |
| 6000 | 50 | 20,00000 | 120 |
| 1000 | 100 | 19,99956 | 10 |
| 2000 | 100 | 19,99956 | 20 |
| 3000 | 100 | 19,99956 | 30 |
| 4000 | 100 | 19,99956 | 40 |
| 5000 | 100 | 19,27688 | 50 |
| 6000 | 100 | 18,18411 | 60 |

Tabela 17: Zestawienie wyników metody roju cząstek dla funkcji Ackley'a - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|---------------------------|-----------------------------|-----------------------------|------------------|
| Liczba odwołań do funkcji | Wartość znalezionej minimum | | Liczba iteracji |
| 1000 | 19,99956 | | 1000 |
| 2000 | 19,99956 | | 2000 |
| 3000 | 19,84388 | | 3000 |
| 4000 | 18,21715 | | 4000 |
| 5000 | 16,85489 | | 5000 |
| 6000 | 12,52854 | | 6000 |
| Algorytm Genetyczny | | | |
| Liczba odwołań do funkcji | Wielkość populacji | Wartość znalezionej minimum | Liczba generacji |
| 1000 | 100 | 0,61431 | 10 |
| 2000 | 100 | 0,30613 | 20 |
| 3000 | 100 | 0,00074 | 30 |
| 4000 | 100 | 0,01294 | 40 |
| 5000 | 100 | 0,00501 | 50 |
| 6000 | 100 | 0,00000 | 60 |
| Metoda Roju Częstek | | | |
| Liczba odwołań do funkcji | Wielkość roju | Wartość znalezionej minimum | Liczba iteracji |
| 1000 | 100 | 19,99956 | 10 |
| 2000 | 100 | 19,99956 | 20 |
| 3000 | 100 | 19,99956 | 30 |
| 4000 | 100 | 19,99956 | 40 |
| 5000 | 100 | 19,27688 | 50 |
| 6000 | 100 | 18,18411 | 60 |

Tabela 18: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Ackley'a - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

6.3.4. Funkcja Rosenbrocka

Funkcja Rosenbrocka jest pierwszą funkcją w której analizując ustalone kryterium, otrzymane najlepsze rezultaty nie należą do jednego algorytmu heurystycznego. Po wyborze optymalnych wielkości populacji oraz roju dla kryterium czasowego, których testy dostępne są do wglądu w tabeli 19 oraz 20, dokonano zestawienia najlepszych rezultatów wszystkich trzech algorytmów w tabeli 21. Wynika z niej, iż dla

wartości maksymalnego czasu od 1 do 3, najlepsze rezultaty uzyskała metoda roju cząstek, podczas gdy dla wyższych z zakresu 4 do 6 - algorytm genetyczny.

| Maksymalny czas | Algorytm Genetyczny | | |
|--------------------|-----------------------|--------------------------------|------------------------------|
| | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | 0,82550 | 8772 |
| 2 | 10 | 0,26159 | 17590 |
| 3 | 10 | 0,82115 | 26534 |
| 4 | 10 | 0,74509 | 35322 |
| 5 | 10 | 1,02229 | 43770 |
| 6 | 10 | 0,00199 | 52470 |
| 1 | 50 | 1,53272 | 14770 |
| 2 | 50 | 0,76142 | 29360 |
| 3 | 50 | 1,53482 | 42960 |
| 4 | 50 | 0,00562 | 57480 |
| 5 | 50 | 2,88337 | 72960 |
| 6 | 50 | 0,00729 | 87580 |
| 1 | 100 | 0,12195 | 15420 |
| 2 | 100 | 0,03372 | 30760 |
| 3 | 100 | 0,24795 | 47360 |
| 4 | 100 | 0,00470 | 61760 |
| 5 | 100 | 0,80783 | 78420 |
| 6 | 100 | 0,00102 | 93740 |

Tabela 19: Zestawienie wyników algorytmu genetycznego dla funkcji Rosenbrocka - kryterium czasowe, Źródło: Opracowanie własne.

| | Metoda Roju Cząstek | | |
|-----------------|---------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | 0,92458 | 27340 |
| 2 | 10 | 20,76899 | 54880 |
| 3 | 10 | 59,03184 | 83430 |
| 4 | 10 | 3,61168 | 102100 |
| 5 | 10 | 4,49411 | 138022 |
| 6 | 10 | 18,06645 | 162724 |
| 1 | 50 | 0,06268 | 46010 |
| 2 | 50 | 0,00618 | 91690 |
| 3 | 50 | 0,00009 | 136180 |
| 4 | 50 | 2,03191 | 182110 |
| 5 | 50 | 3,94942 | 228490 |
| 6 | 50 | 10,89048 | 272280 |
| 1 | 100 | 0,18802 | 42160 |
| 2 | 100 | 41,31897 | 83840 |
| 3 | 100 | 8,63839 | 127680 |
| 4 | 100 | 5,50275 | 166720 |
| 5 | 100 | 4,95079 | 208280 |
| 6 | 100 | 11,04257 | 235120 |

Tabela 20: Zestawienie wyników metody roju cząstek dla funkcji Rosenbrocka - kryterium czasowe, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|-----------------------|-----------------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wartość znalezionej minimum | | Liczba odwołań do funkcji |
| 1 | 254,15459 | | 1937 |
| 2 | 162,15133 | | 3860 |
| 3 | 2407,35639 | | 5771 |
| 4 | 27,87462 | | 7621 |
| 5 | 1710,39496 | | 9439 |
| 6 | 2712,85974 | | 11110 |
| Algorytm Genetyczny | | | |
| Maksymalny czas | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 100 | 0,12195 | 15420 |
| 2 | 100 | 0,03372 | 30760 |
| 3 | 100 | 0,24795 | 47360 |
| 4 | 100 | 0,00470 | 61760 |
| 5 | 100 | 0,80783 | 78420 |
| 6 | 100 | 0,00102 | 93740 |
| Metoda Roju Częstek | | | |
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 50 | 0,06268 | 46010 |
| 2 | 50 | 0,00618 | 91690 |
| 3 | 50 | 0,00009 | 136180 |
| 4 | 50 | 2,03191 | 182110 |
| 5 | 50 | 3,94942 | 228490 |
| 6 | 50 | 10,89048 | 272280 |

Tabela 21: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Rosenbrocka - kryterium czasowe, Źródło: Opracowanie własne.

Dla drugiego analizowanego kryterium dotyczącego ustalonej maksymalnej liczby odwołań do funkcji testowej, rezultaty, które dostępne są do wglądu w tabeli 24 okazały się już bardziej jednolite. Algorytm genetyczny dla wielkości populacji równej 50 (tabela 22) osiągnął lepsze rezultaty od metody roju cząstek z wielkością roju równą 10 (tabela 23) dla każdego przeprowadzonego testu.

| Liczba odwołań do funkcji | Algorytm Genetyczny | | |
|------------------------------|-----------------------|--------------------------------|------------------|
| | Wielkość populacji | Wartość znalezionej minimum | Liczba generacji |
| 1000 | 10 | 11,42978 | 100 |
| 2000 | 10 | 2,08431 | 200 |
| 3000 | 10 | 0,72318 | 300 |
| 4000 | 10 | 2,09036 | 400 |
| 5000 | 10 | 1,65619 | 500 |
| 6000 | 10 | 1,34928 | 600 |
| 1000 | 50 | 12,04365 | 20 |
| 2000 | 50 | 0,89409 | 40 |
| 3000 | 50 | 0,95516 | 60 |
| 4000 | 50 | 1,11253 | 80 |
| 5000 | 50 | 0,92540 | 100 |
| 6000 | 50 | 1,08177 | 120 |
| 1000 | 100 | 20,15711 | 10 |
| 2000 | 100 | 1,83451 | 20 |
| 3000 | 100 | 0,96928 | 30 |
| 4000 | 100 | 1,85377 | 40 |
| 5000 | 100 | 1,71043 | 50 |
| 6000 | 100 | 1,10941 | 60 |

Tabela 22: Zestawienie wyników algorytmu genetycznego dla funkcji Rosenbrocka - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne., Źródło: Opracowanie własne.

| | Metoda Roju Cząstek | | |
|---------------------------|---------------------|-----------------------------|-----------------|
| Liczba odwołań do funkcji | Wielkość roju | Wartość znalezionej minimum | Liczba iteracji |
| 1000 | 10 | 34,72416 | 100 |
| 2000 | 10 | 35,15995 | 200 |
| 3000 | 10 | 40,64258 | 300 |
| 4000 | 10 | 103,63134 | 400 |
| 5000 | 10 | 23,92185 | 500 |
| 6000 | 10 | 32,38185 | 600 |
| 1000 | 50 | 248,14856 | 20 |
| 2000 | 50 | 191,35804 | 40 |
| 3000 | 50 | 217,19964 | 60 |
| 4000 | 50 | 127,14346 | 80 |
| 5000 | 50 | 24,90371 | 100 |
| 6000 | 50 | 133,61537 | 120 |
| 1000 | 100 | 165,99137 | 10 |
| 2000 | 100 | 300,72208 | 20 |
| 3000 | 100 | 117,06707 | 30 |
| 4000 | 100 | 56,55604 | 40 |
| 5000 | 100 | 30,80656 | 50 |
| 6000 | 100 | 42,39143 | 60 |

Tabela 23: Zestawienie wyników metody roju cząstek dla funkcji Rosenbrocka - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|---------------------------|-----------------------------|-----------------------------|------------------|
| Liczba odwołań do funkcji | Wartość znalezionej minimum | | Liczba iteracji |
| 1000 | 403,21334 | | 1000 |
| 2000 | 333,77587 | | 2000 |
| 3000 | 412,82061 | | 3000 |
| 4000 | 163,51327 | | 4000 |
| 5000 | 126,54864 | | 5000 |
| 6000 | 124,45350 | | 6000 |
| Algorytm Genetyczny | | | |
| Liczba odwołań do funkcji | Wielkość populacji | Wartość znalezionej minimum | Liczba generacji |
| 1000 | 50 | 12,04365 | 20 |
| 2000 | 50 | 0,89409 | 40 |
| 3000 | 50 | 0,95516 | 60 |
| 4000 | 50 | 1,11253 | 80 |
| 5000 | 50 | 0,92540 | 100 |
| 6000 | 50 | 1,08177 | 120 |
| Metoda Roju Częstek | | | |
| Liczba odwołań do funkcji | Wielkość roju | Wartość znalezionej minimum | Liczba iteracji |
| 1000 | 10 | 34,72416 | 100 |
| 2000 | 10 | 35,15995 | 200 |
| 3000 | 10 | 40,64258 | 300 |
| 4000 | 10 | 103,63134 | 400 |
| 5000 | 10 | 23,92185 | 500 |
| 6000 | 10 | 32,38185 | 600 |

Tabela 24: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Rosenbrocka - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

6.3.5. Funkcja Eggholdera

Czwartą z kolei funkcją jest funkcja Eggholdera, dla której rezultaty testów weryfikujących optymalną wartość populacji oraz roju cząstek zaprezentowane zostały na wykresach 25 oraz 26. Dla obu algorytmów w kontekście testów dotyczących kryterium maksymalnego czasu, wartości te zostały ustalone na poziomie 100. Ich rezultaty, wraz z rezultatami otrzymanymi dla algorytmu symulowanego wyżarzania

zgrupowane zostały na wykresie 27. Analizując wykres można dostrzec, iż metoda roju cząstek poradziła sobie najlepiej, oscylując bardzo blisko minimum globalnego funkcji Eggholdera, które wynosi $-959,6407$. Tuż za rojem cząstek uplasował się algorytm genetyczny, którego rezultaty osiągały średnio wartość bliską wartości -900 . Wynik ten jest wciąż dużo lepszy od wyników zwróconych przez algorytm symulowanego wyżarzania, który zatrzymywał się na poziomie wartości około -500 .

| Maksymalny czas | Algorytm Genetyczny | | |
|--------------------|-----------------------|--------------------------------|------------------------------|
| | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | -641,37362 | 8249 |
| 2 | 10 | -710,10740 | 17046 |
| 3 | 10 | -743,86777 | 25225 |
| 4 | 10 | -689,52398 | 34094 |
| 5 | 10 | -665,27107 | 40654 |
| 6 | 10 | -705,61882 | 48645 |
| 1 | 50 | -806,45333 | 14575 |
| 2 | 50 | -807,15515 | 28455 |
| 3 | 50 | -881,24751 | 43720 |
| 4 | 50 | -842,77199 | 57395 |
| 5 | 50 | -839,99882 | 70320 |
| 6 | 50 | -874,10131 | 82395 |
| 1 | 100 | -899,91582 | 14790 |
| 2 | 100 | -906,98633 | 31170 |
| 3 | 100 | -899,75344 | 44990 |
| 4 | 100 | -905,75450 | 63010 |
| 5 | 100 | -895,72991 | 79950 |
| 6 | 100 | -918,98375 | 93610 |

Tabela 25: Zestawienie wyników algorytmu genetycznego dla funkcji Eggholdera - kryterium czasowe, Źródło: Opracowanie własne.

| | Metoda Roju Cząstek | | |
|-----------------|---------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | -949,53672 | 28617 |
| 2 | 10 | -911,91620 | 58934 |
| 3 | 10 | -907,80450 | 87470 |
| 4 | 10 | -928,62314 | 118262 |
| 5 | 10 | -946,06533 | 152606 |
| 6 | 10 | -940,60027 | 178690 |
| 1 | 50 | -959,64066 | 42620 |
| 2 | 50 | -959,64066 | 85915 |
| 3 | 50 | -953,13449 | 128705 |
| 4 | 50 | -953,13449 | 173690 |
| 5 | 50 | -959,64066 | 219210 |
| 6 | 50 | -959,64066 | 260525 |
| 1 | 100 | -959,64066 | 36350 |
| 2 | 100 | -959,64066 | 73340 |
| 3 | 100 | -959,64066 | 111470 |
| 4 | 100 | -959,64066 | 148230 |
| 5 | 100 | -959,64066 | 183010 |
| 6 | 100 | -959,64066 | 215380 |

Tabela 26: Zestawienie wyników metody roju cząstek dla funkcji Eggholdera - kryterium czasowe, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|-----------------------|-----------------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wartość znalezionej minimum | | Liczba odwołań do funkcji |
| 1 | -507,29728 | | 1623 |
| 2 | -477,46049 | | 3105 |
| 3 | -496,56648 | | 4954 |
| 4 | -590,61418 | | 6649 |
| 5 | -503,45821 | | 8457 |
| 6 | -496,50273 | | 9105 |
| Algorytm Genetyczny | | | |
| Maksymalny czas | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 100 | -899,91582 | 14790 |
| 2 | 100 | -906,98633 | 31170 |
| 3 | 100 | -899,75344 | 44990 |
| 4 | 100 | -905,75450 | 63010 |
| 5 | 100 | -895,72991 | 79950 |
| 6 | 100 | -918,98375 | 93610 |
| Metoda Roju Częstek | | | |
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 100 | -959,64066 | 36350 |
| 2 | 100 | -959,64066 | 73340 |
| 3 | 100 | -959,64066 | 111470 |
| 4 | 100 | -959,64066 | 148230 |
| 5 | 100 | -959,64066 | 183010 |
| 6 | 100 | -959,64066 | 215380 |

Tabela 27: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Eggholdera - kryterium czasowe, Źródło: Opracowanie własne.

Przechodząc do drugiego analizowanego kryterium - maksymalnej liczby odwołań do funkcji - można zaobserwować, iż wyniki są bardzo zbliżone do wyników otrzymanych w kryterium pierwszym. Testy badające optymalną wielkość populacji dla algorytmu genetycznego oraz wielkości roju dla metody roju cząstek dostępne są do wglądu na wykresie 28 oraz 29, natomiast testy porównujące wszystkie trzy badane algorytmy heurystyczne na wykresie 30.

| Liczba odwołań do funkcji | Algorytm Genetyczny | | |
|------------------------------|-----------------------|--------------------------------|------------------|
| | Wielkość populacji | Wartość znalezionej minimum | Liczba generacji |
| 1000 | 10 | -652,59142 | 100 |
| 2000 | 10 | -739,09817 | 200 |
| 3000 | 10 | -659,72212 | 300 |
| 4000 | 10 | -712,86772 | 400 |
| 5000 | 10 | -597,89480 | 500 |
| 6000 | 10 | -633,92186 | 600 |
| 1000 | 50 | -815,88664 | 20 |
| 2000 | 50 | -852,23377 | 40 |
| 3000 | 50 | -823,89477 | 60 |
| 4000 | 50 | -866,67945 | 80 |
| 5000 | 50 | -822,23638 | 100 |
| 6000 | 50 | -800,93763 | 120 |
| 1000 | 100 | -869,40202 | 10 |
| 2000 | 100 | -864,24074 | 20 |
| 3000 | 100 | -905,17581 | 30 |
| 4000 | 100 | -898,60492 | 40 |
| 5000 | 100 | -915,98754 | 50 |
| 6000 | 100 | -882,63617 | 60 |

Tabela 28: Zestawienie wyników algorytmu genetycznego dla funkcji Eggholdera - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne., Źródło: Opracowanie własne.

| Liczba odwołań do funkcji | Metoda Roju Cząstek | | |
|------------------------------|---------------------|--------------------------------|-----------------|
| | Wielkość roju | Wartość znalezionej minimum | Liczba iteracji |
| 1000 | 10 | -925,67409 | 100 |
| 2000 | 10 | -914,77872 | 200 |
| 3000 | 10 | -933,05286 | 300 |
| 4000 | 10 | -904,80550 | 400 |
| 5000 | 10 | -914,48484 | 500 |
| 6000 | 10 | -915,04781 | 600 |
| 1000 | 50 | -959,07401 | 20 |
| 2000 | 50 | -959,58579 | 40 |
| 3000 | 50 | -953,13449 | 60 |
| 4000 | 50 | -959,64066 | 80 |
| 5000 | 50 | -959,64066 | 100 |
| 6000 | 50 | -959,64066 | 120 |
| 1000 | 100 | -959,64066 | 10 |
| 2000 | 100 | -959,64066 | 20 |
| 3000 | 100 | -959,64066 | 30 |
| 4000 | 100 | -959,64066 | 40 |
| 5000 | 100 | -959,64066 | 50 |
| 6000 | 100 | -959,64066 | 60 |

Tabela 29: Zestawienie wyników metody roju cząstek dla funkcji Eggholdera - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|---------------------------|-----------------------------|-----------------------------|------------------|
| Liczba odwołań do funkcji | Wartość znalezionej minimum | | Liczba iteracji |
| 1000 | -480,19223 | | 1000 |
| 2000 | -473,27827 | | 2000 |
| 3000 | -486,51610 | | 3000 |
| 4000 | -489,33426 | | 4000 |
| 5000 | -543,22278 | | 5000 |
| 6000 | -619,15628 | | 6000 |
| Algorytm Genetyczny | | | |
| Liczba odwołań do funkcji | Wielkość populacji | Wartość znalezionej minimum | Liczba generacji |
| 1000 | 100 | -869,40202 | 10 |
| 2000 | 100 | -864,24074 | 20 |
| 3000 | 100 | -905,17581 | 30 |
| 4000 | 100 | -898,60492 | 40 |
| 5000 | 100 | -915,98754 | 50 |
| 6000 | 100 | -882,63617 | 60 |
| Metoda Roju Cząstek | | | |
| Liczba odwołań do funkcji | Wielkość roju | Wartość znalezionej minimum | Liczba iteracji |
| 1000 | 100 | -959,64066 | 10 |
| 2000 | 100 | -959,64066 | 20 |
| 3000 | 100 | -959,64066 | 30 |
| 4000 | 100 | -959,64066 | 40 |
| 5000 | 100 | -959,64066 | 50 |
| 6000 | 100 | -959,64066 | 60 |

Tabela 30: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Eggholdera - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

6.3.6. Funkcja Griewanka

Ostatnią użytą funkcją testową była funkcja Griewanka, dla której podobnie jak w przypadku funkcji Rosenbrocka rezultaty otrzymane dla kryterium maksymalnego czasu nie są jednoznaczne. Porównując najlepsze wyniki otrzymane dla algorytmu genetycznego z tabeli 31 oraz dla metody roju cząstek z tabeli 32 z wynikami otrzymanymi przez algorytm symulowanego wyżarzania, można dostrzec w

tabeli 33, iż lepszy rezultat otrzymany w teście 1 oraz 3 sekund należy do algorytmu genetycznego, a dla pozostałych wartości maksymalnego czasu dla metody roju cząstek. Otrzymane przez algorytm genetyczny oraz metodę roju cząstek rezultaty są jednak bardzo do siebie zbliżone i nie mogą jednoznacznie determinować lepszego algorytmu do zastosowania.

| Maksymalny czas | Algorytm Genetyczny | | |
|-----------------|---------------------|-----------------------------|---------------------------|
| | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | 0,01380 | 8684 |
| 2 | 10 | 0,04242 | 17494 |
| 3 | 10 | 0,03403 | 26182 |
| 4 | 10 | 0,10179 | 35348 |
| 5 | 10 | 0,09381 | 43874 |
| 6 | 10 | 0,00497 | 50872 |
| 1 | 50 | 0,09815 | 14280 |
| 2 | 50 | 0,03550 | 27960 |
| 3 | 50 | 0,02316 | 42270 |
| 4 | 50 | 0,06786 | 58340 |
| 5 | 50 | 0,06254 | 73200 |
| 6 | 50 | 0,00448 | 86740 |
| 1 | 100 | 0,00148 | 15400 |
| 2 | 100 | 0,01035 | 31040 |
| 3 | 100 | 0,00592 | 46460 |
| 4 | 100 | 0,00592 | 61160 |
| 5 | 100 | 0,00789 | 77700 |
| 6 | 100 | 0,00395 | 93600 |

Tabela 31: Zestawienie wyników algorytmu genetycznego dla funkcji Griewanka - kryterium czasowe, Źródło: Opracowanie własne.

| | Metoda Roju Cząstek | | |
|-----------------|---------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 10 | 0,00790 | 27728 |
| 2 | 10 | 0,00592 | 55890 |
| 3 | 10 | 0,00987 | 84976 |
| 4 | 10 | 0,01627 | 114240 |
| 5 | 10 | 0,01232 | 140956 |
| 6 | 10 | 0,03450 | 172454 |
| 1 | 50 | 0,02194 | 41530 |
| 2 | 50 | 0,00787 | 78980 |
| 3 | 50 | 0,00739 | 125790 |
| 4 | 50 | 0,00542 | 159940 |
| 5 | 50 | 0,00493 | 195560 |
| 6 | 50 | 0,00542 | 242800 |
| 1 | 100 | 0,00296 | 37520 |
| 2 | 100 | 0,00591 | 72440 |
| 3 | 100 | 0,00641 | 103300 |
| 4 | 100 | 0,00197 | 142220 |
| 5 | 100 | 0,00197 | 181280 |
| 6 | 100 | 0,00148 | 218700 |

Tabela 32: Zestawienie wyników metody roju cząstek dla funkcji Griewanka - kryterium czasowe, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|-----------------------|-----------------------------|-----------------------------|---------------------------|
| Maksymalny czas | Wartość znalezionej minimum | | Liczba odwołań do funkcji |
| 1 | 0,71977 | | 2002 |
| 2 | 0,19232 | | 3888 |
| 3 | 0,15288 | | 5809 |
| 4 | 0,16766 | | 7711 |
| 5 | 0,12635 | | 9616 |
| 6 | 0,11727 | | 11396 |
| Algorytm Genetyczny | | | |
| Maksymalny czas | Wielkość populacji | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 100 | 0,00148 | 15400 |
| 2 | 100 | 0,01035 | 31040 |
| 3 | 100 | 0,00592 | 46460 |
| 4 | 100 | 0,00592 | 61160 |
| 5 | 100 | 0,00789 | 77700 |
| 6 | 100 | 0,00395 | 93600 |
| Metoda Roju Cząstek | | | |
| Maksymalny czas | Wielkość roju | Wartość znalezionej minimum | Liczba odwołań do funkcji |
| 1 | 100 | 0,00296 | 37520 |
| 2 | 100 | 0,00591 | 72440 |
| 3 | 100 | 0,00641 | 103300 |
| 4 | 100 | 0,00197 | 142220 |
| 5 | 100 | 0,00197 | 181280 |
| 6 | 100 | 0,00148 | 218700 |

Tabela 33: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Griewanka - kryterium czasowe, Źródło: Opracowanie własne.

Drugie kryterium, pod kątem którego przeprowadzone zostały testy wskazuje już jednak w tabeli 36 jednoznacznie, iż lepsze rezultaty uzyskał algorytm genetyczny. Ponad dwukrotnie większa liczba odwołań metody roju cząstek do funkcji testowej w pierwszym kryterium, w kryterium drugim została zrównana. W tym przypadku przy takiej samej liczbie obliczeń funkcji testowej, algorytm genetyczny uzyskał wyniki bliższe minimum globalnemu niż metoda roju cząstek. Algorytm symulowanego wyżarzania w testach dotyczących obu kryteriów odniósł najgorsze rezultaty ze wszystkich porównywanych algorytmów heurystycznych. Warto jednak nadmienić, iż wyniki przez niego uzyskane oscylowały również blisko minimum globalnego.

| Liczba odwołań do funkcji | Algorytm Genetyczny | | |
|------------------------------|-----------------------|--------------------------------|------------------|
| | Wielkość populacji | Wartość znalezionej minimum | Liczba generacji |
| 1000 | 10 | 0,00462 | 100 |
| 2000 | 10 | 0,04365 | 200 |
| 3000 | 10 | 0,00396 | 300 |
| 4000 | 10 | 0,00788 | 400 |
| 5000 | 10 | 0,01948 | 500 |
| 6000 | 10 | 0,02440 | 600 |
| 1000 | 50 | 0,01314 | 20 |
| 2000 | 50 | 0,01897 | 40 |
| 3000 | 50 | 0,02859 | 60 |
| 4000 | 50 | 0,00395 | 80 |
| 5000 | 50 | 0,00468 | 100 |
| 6000 | 50 | 0,02538 | 120 |
| 1000 | 100 | 0,00391 | 10 |
| 2000 | 100 | 0,00395 | 20 |
| 3000 | 100 | 0,00000 | 30 |
| 4000 | 100 | 0,00198 | 40 |
| 5000 | 100 | 0,00000 | 50 |
| 6000 | 100 | 0,01380 | 60 |

Tabela 34: Zestawienie wyników algorytmu genetycznego dla funkcji Griewanka - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne., Źródło: Opracowanie własne.

| Liczba odwołań do funkcji | Metoda Roju Cząstek | | |
|------------------------------|---------------------|---------------------------------|-----------------|
| | Wielkość roju | Wartość znalezionego minimum | Liczba iteracji |
| 1000 | 10 | 0,19106 | 100 |
| 2000 | 10 | 0,10885 | 200 |
| 3000 | 10 | 0,05545 | 300 |
| 4000 | 10 | 0,07227 | 400 |
| 5000 | 10 | 0,03778 | 500 |
| 6000 | 10 | 0,06479 | 600 |
| 1000 | 50 | 0,17494 | 20 |
| 2000 | 50 | 0,06023 | 40 |
| 3000 | 50 | 0,02455 | 60 |
| 4000 | 50 | 0,03154 | 80 |
| 5000 | 50 | 0,03950 | 100 |
| 6000 | 50 | 0,02333 | 120 |
| 1000 | 100 | 0,16603 | 10 |
| 2000 | 100 | 0,04679 | 20 |
| 3000 | 100 | 0,03319 | 30 |
| 4000 | 100 | 0,02646 | 40 |
| 5000 | 100 | 0,02729 | 50 |
| 6000 | 100 | 0,01758 | 60 |

Tabela 35: Zestawienie wyników metody roju cząstek dla funkcji Griewanka - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

| Symulowane Wyżarzanie | | | |
|---------------------------|-----------------------------|-----------------------------|------------------|
| Liczba odwołań do funkcji | Wartość znalezionej minimum | | Liczba iteracji |
| 1000 | 1,05495 | | 1000 |
| 2000 | 0,29544 | | 2000 |
| 3000 | 0,19756 | | 3000 |
| 4000 | 0,18222 | | 4000 |
| 5000 | 0,16877 | | 5000 |
| 6000 | 0,15037 | | 6000 |
| Algorytm Genetyczny | | | |
| Liczba odwołań do funkcji | Wielkość populacji | Wartość znalezionej minimum | Liczba generacji |
| 1000 | 100 | 0,00391 | 10 |
| 2000 | 100 | 0,00395 | 20 |
| 3000 | 100 | 0,00000 | 30 |
| 4000 | 100 | 0,00198 | 40 |
| 5000 | 100 | 0,00000 | 50 |
| 6000 | 100 | 0,01380 | 60 |
| Metoda Roju Częstek | | | |
| Liczba odwołań do funkcji | Wielkość roju | Wartość znalezionej minimum | Liczba iteracji |
| 1000 | 100 | 0,16603 | 10 |
| 2000 | 100 | 0,04679 | 20 |
| 3000 | 100 | 0,03319 | 30 |
| 4000 | 100 | 0,02646 | 40 |
| 5000 | 100 | 0,02729 | 50 |
| 6000 | 100 | 0,01758 | 60 |

Tabela 36: Zestawienie najlepszych wyników wszystkich trzech porównywanych algorytmów metaheurystycznych dla funkcji Griewanka - kryterium liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

6.4. Wnioski z przeprowadzonych doświadczeń

Przeprowadzone doświadczenia wykonane zostały z myślą o wyrównaniu szans każdego z badanych algorytmów heurystycznych. Metody porównawcze opisane w 6.1 umożliwiły na podstawie ustalonych kryteriów, jednakowych dla wszystkich algorytmów, sprawdzić rezultaty przez nie uzyskane. Przez fakt różną specyfikę algorytmów oraz posiadanie przez nich różnych parametrów, czasochłonną kwestią oka-

zało się zrównanie ich szans oraz ich wyskalowanie. Podsumowując przeprowadzone doświadczenia w kontekście kryterium ustalonego czasu pracy, w obrębie którego algorytm za zadanie miał znaleźć najmniejszą wartość badanej funkcji, algorytm genetyczny w jednej na sześć funkcji testowych - funkcji Ackley'a - uzyskał najlepszy rezultat dla każdej wartości maksymalnego czasu. W dwóch kolejnych funkcjach - Rosenbrocka oraz Griewanka - uzyskał najlepszy wynik dla kolejno trzech z sześciu oraz czterech z sześciu wartości parametru. Funkcja Bohachevsky'ego, Colville'a oraz Eggholdera były funkcjami w której algorytm genetyczny uplasował się na drugim miejscu, za metodą roju cząstek, która zwróciła najlepsze rezultaty. Metoda ta uzyskała również najlepsze wyniki dla trzech z sześciu oraz dwóch z sześciu parametrów dla funkcji Rosenbrocka oraz Griewanka. Najgorzej w zestawieniu wypadł algorytm symulowanego wyżarzania, który dla każdej funkcji testowej ulokował się na miejscu trzecim. Algorytm ten, choć zwracał wyniki bliskie minimum globalnemu, były one gorsze od rezultatów zwróconych przez GA oraz PSO.

Przy analizie wyników dotyczących próby czasowej, zauważalna była różnica w ilości wyliczeń funkcji testowej dokonywana przez porównywane algorytmy. Metoda roju cząstek odwoływała się do funkcji testowych średnio 2,59 raza więcej niż algorytm genetyczny oraz 25.58 raza więcej niż algorytm symulowanego wyżarzania. Różnice te zostały zrównane w wynikach testów odnoszących się do drugiego kryterium porównawczego jakim była ustalona maksymalna liczba odwołań do funkcji testowej.

Przy założeniu jednakowej liczby odwołań do funkcji testowych, klasyfikacja algorytmów uległa dużemu przetasowaniu w porównaniu do kryterium pierwszego. Bez zmiany wyników zachowały się wyłącznie testowane funkcje dwuwymiarowe - Bohachevsky'ego oraz Eggholdera, w której metoda roju cząstek uplasowała się przed algorytmem genetycznym oraz algorytmem symulowanego wyżarzania. W pozostałych czterech funkcjach najlepsze rezultaty uzyskał algorytm genetyczny, który dla funkcji Rosenbrocka oraz Griewanka zajął pierwsze miejsce przed metodą roju cząstek oraz symulowanym wyżarzaniem. Ciekawe wyniki dostarczyła funkcja Colville'a, dla której w pierwszym kryterium najlepsze wyniki uzyskała metoda roju cząstek. W drugim kryterium osiągnęła ona najgorszy rezultat, plasując się za algorytmem symulowanego wyżarzania, który dla funkcji Colville'a oraz Ackley'a uzyskał wyniki pozwalające umiejscowić go na drugiej pozycji.

Ranking algorytmów odnoszący się do wyników przeprowadzonych testów dla

kryterium czasowego oraz kryterium maksymalnej liczby odwołań do funkcji testowej załączone zostały do wglądu w tabeli 37 oraz 38

| Miejsce w rankingu Funkcja testowa | 1 | 2 | 3 |
|---------------------------------------|--------|-----|----|
| Funkcja Bohachevsky'ego 1 | PSO | GA | SA |
| Funkcja Colville'a | PSO | GA | SA |
| Funkcja Ackley'a | GA | PSO | SA |
| Funkcja Rosenbrocka | PSO/GA | | SA |
| Funkcja Eggholdera | PSO | GA | SA |
| Funkcja Griewanka | PSO/GA | | SA |

Tabela 37: Ranking porównanych algorytmów heurystycznych (Algorytm genetyczny - GA; Metoda roju cząstek - PSO; Algorytm symulowanego wyżarzania - SA) dla wyników dotyczących kryterium maksymalnego czasu, Źródło: Opracowanie własne.

| Miejsce w rankingu Funkcja testowa | 1 | 2 | 3 |
|---------------------------------------|-----|-----|-----|
| Funkcja Bohachevsky'ego 1 | PSO | GA | SA |
| Funkcja Colville'a | GA | SA | PSO |
| Funkcja Ackley'a | GA | SA | PSO |
| Funkcja Rosenbrocka | GA | PSO | SA |
| Funkcja Eggholdera | PSO | GA | SA |
| Funkcja Griewanka | GA | PSO | SA |

Tabela 38: Ranking porównanych algorytmów heurystycznych (Algorytm genetyczny - GA; Metoda roju cząstek - PSO; Algorytm symulowanego wyżarzania - SA) dla wyników dotyczących maksymalnej liczby odwołań do funkcji testowej, Źródło: Opracowanie własne.

7. Podsumowanie

Celem prezentowanej pracy magisterskiej było dokonanie analizy porównawczej algorytmu symulowanego wyżarzania, algorytmu genetycznego oraz roju cząstek, która przeprowadzona została dzięki zaimplementowanej dedykowanej aplikacji bazodanowej, której przeznaczeniem było zautomatyzowanie procesu szukania minimum globalnego zadanych dla funkcji testowych. Zastosowanie środowiska bazodanowego umożliwiło bardzo szybkie zapisywanie dużych ilości danych dostarczanej z zaimplementowanej aplikacji, a następnie przy pomocy języka SQL agregować je według uznanych kryteriów. W fazie implementacji założonych modułów prezentowanej aplikacji napotkano na kilka problemów dotyczących zmiany wymagań projektowych, sposobem realizacji ustalonych funkcji oraz integracji systemów. Wraz z biegiem czasu powstawały nowe pomysły, które dotyczyły usprawnienia istniejących rozwiązań, które nie zawsze jednak mogły być wprowadzone przez pryzmat czasu którego to wymagały do ich implementacji, przetestowania oraz połączenia z istniejącym już systemem. Powstające propozycje zmian wiązały się także z modyfikacją istniejącej architektury systemu, co w znacznym stopniu zmieniałoby zaplanowany wcześniej harmonogram prac i wydłużałoby finalne ukończenie projektu. Pomimo tych elementów, przedstawiony cel prezentowanej pracy magisterskiej został zrealizowany, czego efektem jest działająca aplikacja bazodanowa posiadająca funkcjonalności umożliwiające jego osiągnięcie oraz wyniki testów które namacalnie prezentują uzyskane rezultaty otrzymane przez porównywane algorytmy metaheurystyczne.

Bibliografia

- [1] Michalewicz Z., Fogel D.B.: *Jak to rozwiązać czyli nowoczesna heurystyka*, Wydawnictwa Naukowo-Techniczne, Warszawa, 2006, str. 25-228
- [2] http://infinity77.net/global_optimization/test_functions.html
[dostęp 1.06.2017r. - 26.08.2017r.]
- [3] Castillo O., Melin P.: *Soft Computing and Fractal Theory for Intelligent Manufacturing*, Physica-Verlag, Heidelberg, 2003, str. 100-101
- [4] Nola R., Sankey H.: *Theories of Scientific Method*, Routledge, Nowy Jork, 2007, str. 25-28
- [5] http://zsi.ii.us.edu.pl/~mboryczka/IntStad/pso_informacje.php
[dostęp 20.06.2017r. - 26.08.2017r.]
- [6] Holland J. H.: *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, 1975
- [7] Deppa S. N.: *Introduction to Genetic Algorithms*, Springer-Verlag, Heidelberg, 2008, str. 43-44
- [8] Ibidem, str. 47-48.
- [9] <http://dictionary.cambridge.org/dictionary/english/back-end> [dostęp 1.08.2017r. - 26.08.2017r.]
- [10] <https://www.tiobe.com/tiobe-index/> [dostęp 22.08.2017r. - 23.08.2017r.]
- [11] Hejlsberg A., Torgersen M. Wilthamuth S., Golde P.: *The C# Programming Language*, Addison-Wesley, Massachusetts, 2009, str. 12-14
- [12] <http://urriellu.net/en/articles-software/csharp-advantages.html> [dostęp 22.08.2017r. - 23.08.2017r.]
- [13] Nathan A.: *WPF 4.5. Księga eksperta*, Wydawnictwo Helion, Gliwice, 2015

- [14] http://www.w3schools.com/xml/xml_what_is.asp
- [15] Itzik B.: *Microsoft SQL Server 2012. Podstawy języka T-SQL*, Wydawnictwo Helion, Gliwice, 2012
- [16] Matulewski J.: *Visual Studio 2013. Podręcznik programowania w C# z zadaniami*, Wydawnictwo Helion, Gliwice, 2014
- [17] <https://www.jetbrains.com/resharper/features/> [dostęp 1.08.2017r. - 23.08.2017r.]
- [18] Jaskiewicz A.: *Inżynieria oprogramowania*, Wydawnictwo Helion, 1997
- [19] Petzold C.: *Windows 8. Programowanie aplikacji z wykorzystaniem C# i XAML*, Wydawnictwo Helion, Gliwice, 2013, str. 215-235
- [20] MacDonald M.: *Pro WPF 4.5 in VB: Windows Presentation Foundation in .NET 4.5*, Wydawnictwo Apress, 2012, str. 227-242
- [21] Ibidem, str. 243-268.
- [22] Seemann M.: *Dependency Injection in .NET*, Manning Publications Co., Nowy Jork, 2012

Spis rysunków

| | | |
|----|---|----|
| 1 | Wykres oraz wzór funkcji Bohachevsky'ego I dwóch zmiennych, Źródło: https://www.sfu.ca/~ssurjano/boha.html | 6 |
| 2 | Wykres oraz wzór funkcji Ackley'a dwóch zmiennych, Źródło: https://www.sfu.ca/~ssurjano/ackley.html | 8 |
| 3 | Wykres oraz wzór funkcji Rosenbrocka dla $d = 4$ zmiennych, Źródło: https://www.sfu.ca/~ssurjano/rosen.html | 9 |
| 4 | Wykres oraz wzór funkcji Eggholdera dwóch zmiennych, Źródło: https://www.sfu.ca/~ssurjano/egg.html | 10 |
| 5 | Wykres oraz wzór funkcji Griewanka dla $d = 3$ zmiennych, Źródło: https://www.sfu.ca/~ssurjano/gri.html | 11 |
| 6 | Zasada działania algorytmu symulowanego wyżarzania w postaci gra- ficznej, Źródło: http://iacs-courses.seas.harvard.edu/courses/am207/blog/ images/mcmc.png | 14 |
| 7 | Graficzna prezentacja operacji krzyżowania jednopunktowego, Źródło: https://www.linkedin.com/pulse/genetic-algorithms- sharmishtha-mahajan-patwardhan | 18 |
| 8 | Graficzna prezentacja operacji mutacji, Źródło: http://web.arch.usyd.edu.au/ ~rob/applets/house/images/single_point_mutation.png | 19 |
| 9 | Interfejs SQL Server Management Studio 2017 Źródło: Opracowanie własne. | 24 |
| 10 | Przepływ danych we wzorcu MVVM, Źródło: https://i-msdn.sec.s- msft.com/dynimg/IC648329.png | 29 |
| 11 | Widok na pliki funkcji napisanych w środowisku Matlab z poziomu solucji, Źródło: Opracowanie własne. | 31 |
| 12 | Widok na pliki wygenerowane przez Entity Framework z poziomu solucji, Źródło: Opracowanie własne. | 32 |
| 13 | Przestawienie części słownikowej bazy danych, Źródło: Opracowanie własne. | 36 |

Kody źródłowe

| | | |
|---|--|----|
| 1 | Implementacja oraz opis funkcji tłumaczącej informacje odebrane ze środowiska obliczeniowego Matlab. | 33 |
|---|--|----|