

**UNIWERSYTET KAZIMIERZA WIELKIEGO
WYDZIAŁ MATEMATYKI, FIZYKI I TECHNIKI
INSTYTUT TECHNIKI**

Jonasz Kulpinski

79579

**Zastosowanie metod sztucznej inteligencji inspirowanych naturą do
rozwiązywania problemów wielokryterialnych**

**Praca magisterska napisana pod kierunkiem:
dr. inż. Jacka Czerniaka**

Bydgoszcz 2017

Spis treści

1.	Wstęp	7
1.1.	Wprowadzenie	7
1.1.1.	Co to jest sztuczna inteligencja?	8
1.2.	Cel pracy	10
1.3.	Teza pracy	10
1.4.	Zakres i układ pracy	10
2.	Studium literatury	11
2.1.	Heurystyki, strategie poszukiwania i metaheurystyki	11
2.2.	Optymalizacja wielokryterialna	13
2.3.	Tradycyjne metody optymalizacji wielokryterialnej	13
2.3.1.	Metoda kryteriów ważonych	13
2.3.2.	Metoda hierarchiczna	14
2.3.3.	Metoda ograniczonych kryteriów	14
2.3.4.	Metoda kryterium globalnego	15
2.4.	Inspiracja naturą: algorytmy rojowe	15
2.4.1.	Algorytmy kolonii pszczół	15
2.4.2.	Algorytmy optymalizacji rojem częstek	19
2.4.3.	Algorytm świetlików	21
2.4.4.	Algorytm nietoperza	24
2.5.	Inspiracja naturą: algorytmy ewolucyjne	27
2.5.1.	Algorytmy ewolucyjne: reprezentacja osobników	29
2.5.2.	Algorytmy ewolucyjne: Funkcja przystosowania	29
2.5.3.	Algorytmy ewolucyjne: Selekcja osobników	30
2.5.4.	Algorytmy ewolucyjne: Krzyżowanie i mutacja	31
2.5.5.	Algorytmy ewolucyjne: Algorytmy genetyczne	32

2.5.6.	Algorytmy ewolucyjne: Programowanie genetyczne	32
2.5.7.	Algorytmy ewolucyjne: Strategie ewolucyjne	33
2.5.8.	Algorytmy ewolucyjne: Ewolucja różnicowa	33
2.6.	Porównanie zwykłych metod optymalizacji z optymalizacją ewolucyjną	35
3.	Eksperymenty	37
3.1.	Metody i środowisko badawcze wykorzystane w trakcie eksperymentów	37
3.2.	Opis wywołań metod i funkcji rysujących w języku R użytych w badaniach	38
3.2.1.	PSO	38
3.2.2.	BAT	39
3.2.3.	GA	40
3.2.4.	DE	41
3.2.5.	Wykresy 2D z pozycjami osobników	42
3.2.6.	Wykresy 2D zależności znalezioneego minimum od iteracji .	43
3.2.7.	Wykresy 3D funkcji	43
3.3.	Opis funkcji użytych w eksperymencie	43
3.3.1.	Ackley	44
3.3.2.	Beale	45
3.3.3.	Goldstein	46
3.3.4.	Bartels Conn	47
3.3.5.	Leon	48
3.3.6.	Eggholder	49
3.3.7.	Venter	50
3.3.8.	Matyas	51
3.3.9.	Zirilli	52
3.3.10.	Easom	53
3.3.11.	Rastrigin	54
3.3.12.	Levy N.13	55
3.3.13.	Drop Wave	56
3.4.	Optymalizacja rojem częstek	57

3.4.1.	Ackley	63
3.4.2.	Bartels	64
3.4.3.	Beale	65
3.4.4.	Easom	67
3.4.5.	Eggholder	68
3.4.6.	Goldstein	69
3.4.7.	Leon	70
3.4.8.	Matyas	71
3.4.9.	Venter	73
3.4.10.	Zirilli	74
3.4.11.	Drop Wave	75
3.4.12.	Rastrigin	76
3.4.13.	Levy N.13	77
3.5.	Optymalizacja algorytmem nietoperza	78
3.5.1.	Ackley	84
3.5.2.	Bartels	85
3.5.3.	Beale	86
3.5.4.	Easom	88
3.5.5.	Eggholder	89
3.5.6.	Goldstein	90
3.5.7.	Leon	91
3.5.8.	Matyas	92
3.5.9.	Venter	93
3.5.10.	Zirilli	94
3.5.11.	Drop Wave	95
3.5.12.	Rastrigin	96
3.5.13.	Levy N.13	97
3.6.	Optymalizacja algorytmem genetycznym	98
3.6.1.	Ackley	104
3.6.2.	Bartels	105
3.6.3.	Beale	106
3.6.4.	Easom	107

3.6.5.	Eggholder	108
3.6.6.	Goldstein	109
3.6.7.	Leon	110
3.6.8.	Matyas	111
3.6.9.	Venter	112
3.6.10.	Zirilli	113
3.6.11.	Drop Wave	114
3.6.12.	Rastrigin	115
3.6.13.	Levy N.13	116
3.7.	Optymalizacja algorytmem ewolucji różnicowej	117
3.7.1.	Ackley	123
3.7.2.	Bartels	124
3.7.3.	Beale	125
3.7.4.	Easom	126
3.7.5.	Eggholder	127
3.7.6.	Goldstein	128
3.7.7.	Leon	129
3.7.8.	Matyas	130
3.7.9.	Venter	131
3.7.10.	Zirilli	132
3.7.11.	Drop Wave	133
3.7.12.	Rastrigin	134
3.7.13.	Levy N.13	135
3.8.	Omówienie wyników eksperymentu	136
3.8.1.	Ackley	136
3.8.2.	Beale	137
3.8.3.	Goldstein	138
3.8.4.	Bartels Conn	139
3.8.5.	Leon	140
3.8.6.	Eggholder	141
3.8.7.	Venter	142
3.8.8.	Matyas	143

3.8.9.	Zirilli	144
3.8.10.	Easom	145
3.8.11.	Drop Wave, Levy N.13, Rastrigin	146
4.	Zakończenie	148
4.1.	Podsumowanie	148
4.2.	Wnioski	149
	Bibliografia	150
5.	Streszczenie	155

1. Wstęp

1.1. Wprowadzenie

Sztuczna inteligencja to jedna z najnowszych dziedzin w nauce i technice. Prace nad nią zaczęły się wkrótce po II wojnie światowej. Podobnie jak na przykład w biologii molekularnej, w dziedzinie SI jest wciąż sporo do zbadania. W przypadku np. fizyki można mieć wrażenie, że na wszystkie dobre pomysły wpadli już Galileusz, Newton czy Einstein, SI ma jeszcze kilka odkryć przed sobą [1].

Pięćdziesiąt lat temu logika odnosiła niesamowite sukcesy. Wtedy to miały miejsce naukowe osiągnięcia, takie jak formalizacja matematyki, maszyny Turinga czy twierdzenia Gödla. Równocześnie zaczęła się rozwijać technologia informacyjna, która swój początek opierała w znacznym stopniu na logice i jej osiągnięciach, z metody formalnej korzystano powoli w innych dziedzinach nauki, również humanistycznych, w filozofii i metodologii nauk nastąpiła rewolucja dzięki postępu logiki.

W tych warunkach, w roku 1956 powstała nowa dziedzina *sztuczna inteligencja*. Naukowcy zaczęli podejmować działania, zmierzające do spełnienia marzenia ludzkości o stworzeniu sztucznego człowieka. Prowadzono badania w zakresie robotyki, które miały na celu skonstruowanie syntetycznego ciała, jednakże początkowo skupiono na stworzeniu sztucznego umysłu, skąd nazwa dziedziny. Szczególne znaczenie w badaniach miała logika formalna, w której osiągnięcia starano wyposażyć komputer, aby uzyskał zdolność logicznego myślenia.

Powszechnie były informacje o nowych perspektywach, związanych ze sztuczną inteligencją oraz o postępach w tej dziedzinie. Rozpoczęły się dyskusje poruszające aspekty etyczne lub filozoficzne wynikające z rozwoju AI, zaczęto zastanawiać się na nowo nad fenomenem świadomości.

Mijały lata, a myślącej maszyny wciąż nie udało się skonstruować, nawet mimo tego, że możliwości współczesnych komputerów są znaczco wyższe niż prognozowano przed laty, a w innych podobnych dziedzinach miał miejsce rozwój szybszy niż spekulowano.

Z powodu braku dużego sukcesu i powiększającej się różnicy między tym co zapowiadano, a rzeczywistością, nastąpiła szeroka krytyka, również ze strony autorytetów, np. Penrose'a lub Edelmana. Sytuacja ta umożliwiła krytykę również humanistom, część z nich uznała całkowitą klęskę sztucznej inteligencji, a niektórzy ogłosili porażkę metody

formalnej i zakończenie ery kartezjańskiej.

Ostatnie twierdzenia są oczywiście przesadzone i prowokacyjne, bowiem technologia informatyczna ledwo zaczęła zmieniać świat, a metoda formalna to jej podstawa. Mimo wszystko półwiecze nieudanych prób stworzenia takiej sztucznej inteligencji jak oczekiwano daje do myślenia [3].

Metody SI wykorzystuje się obecnie w wielu zadaniach, od ogólnych, do specyficznych np. granie w szachy, wykazywanie teorii matematycznych, pisanie poezji, prowadzenie samochodu, diagnozowanie chorób. SI znajduje zastosowanie w każdym zadaniu wymagającym inteligencji, to naprawdę uniwersalna nauka.

1.1.1. Co to jest sztuczna inteligencja?

Działanie na sposób ludzki: test Turinga

Test Turinga zaproponowany przez Alana Turinga w 1950, został opracowany, aby określić czy dana maszyna jest inteligentna. Człowiek zadaje kilka pytań, a komputer odpowiada, test jest zdany jeśli badacz nie jest w stanie stwierdzić, czy danej odpowiedzi udzielił komputer czy człowiek. Taki komputer powinien mieć następujące możliwości:

- przetwarzanie języka naturalnego,
- reprezentacja wiedzy,
- zautomatyzowane rozumowanie,
- nauczanie maszynowe.

Test Turinga celowo unika bezpośredniego kontaktu fizycznego między badaczem a komputerem, ponieważ fizyczna symulacja osoby jest niepotrzebna, badana jest inteligencja. Jednak tzw. „Total Turing Test” zawiera sygnał video, tak że badający mogą przetestować zdolności percepcyjne badanego obiektu [1]. Od roku 1990, raz do roku odbywa się konkurs, w którym biorą udział napisane przez programistów chatboty, egzaminowane w rozmowie z sędziami na podobieństwo Testu Turinga. Sędzia zadaje poprzez terminal pytania, nie wiedząc czy odpowiada na nie człowiek, czy maszyna. Konkurs ma miejsce w The Cambridge Center of Behavioral Studies, sędziowie, będący pracownikami instytutu są wybierani w losowaniu. Testy trwają jeden dzień i odbywają się przy terminalach, przez które sędziowie piszą z człowiekiem, bądź komputerem. Pod koniec konkursu

podejmowane są decyzje, które terminale są wg sędziów obsługiwane są przez chatboty, a które przez ludzi, oraz który z ocenionych jako komputerowy program miał inteligencję najbardziej zbliżoną do ludzkiej [2].

Myślenie na sposób ludzki: modelowanie poznawcze

Żeby powiedzieć, że dany program posiada coś na kształt rozumowania ludzkiego, trzeba jakoś określić sposób, w jaki myślą ludzie. Trzeba dostać się do środka rzeczywistych, funkcjonujących ludzkich umysłów. Istnieją trzy sposoby, żeby to zrobić: poprzez introspekcję-próby przechwycenia ludzkich myśli podczas ich ”drogi” w umyśle, poprzez eksperymenty psychologiczne-obserwując osobę w działaniu, poprzez obrazowanie mózgu obserwując działania mózgu. Kiedy uzyska się wystarczająco dużo danych o procesie myślenia, można wykorzystać to do stworzenia programu. Jeśli zachowanie programu odpowiada zachowaniu człowieka, jest to dowód, że niektóre mechanizmy programu także mogą funkcjonować u ludzi. Na przykład Allen Newell i Herbert Simon, twórcy „General Program Solver” od poprawnego rozwiązywania zadanych problemów bardziej zainteresowani byli tym, czy ich program rozwiązuje te problemy w sposób podobny do tego jak robi to człowiek. Interdyscyplinarna kognitywistyka łączy komputerowe modele z SI i eksperymentalnymi technikami psychologicznymi w celu stworzenia testowalnych teorii ludzkiego umysłu.

Myślenie racjonalne: „Prawa rządzące myśleniem”

Grecki filozof Arystoteles był jednym z pierwszych, którzy próbowali określić „prawo myślenia”, pozwalające określić, co jest warunkiem rozumowania. Arystoteles wprowadził termin sylogizm oznaczający, że musi powstać właściwy wniosek na temat dwóch przesłanek, jeśli przesłanki te mają element wspólny i we wszystkich przesłankach znajduje się element np. *Wszystkie aligatory to krokodyle. Każdy krokodyl jest gadem. W takim razie każdy aligator jest gadem.* Taki rodzaj myślenia o problemie nazywamy *logiką*.

Logicy w XIX wieku opracowali opisy różnego rodzaju obiektów oraz relacji między nimi. Do roku 1965 istniały programy, które mogłyby w zasadzie rozwiązać każdy problem rozwiązywalny w notacji logicznej. Tzw logiczna tradycja, na której opiera się SI daje nadzieję na intelligentne systemy oparte o tego rodzaju programy.

Jednak istnieje pewien problem dla takiego rodzaju podejścia. Przede wszystkim nieformalna wiedza nie może być podawana w warunkach zapisu logicznego, jeżeli nie mamy pewności, że te wiadomości są prawdziwe [1].

1.2. Cel pracy

Celem jest określenie, które algorytmy - rojowe czy ewolucyjne, są lepsze w rozwiązywaniu problemów optymalizacyjnych.

1.3. Teza pracy

Tezą jest wykorzystanie metod SI inspirowanych naturą w rozwiązywaniu zadań i sprawdzenie czy algorytmy wykorzystujące inteligencję roju są wydajniejsze od algorytmów ewolucyjnych.

1.4. Zakres i układ pracy

Zakres dotyczy wyboru literatury zawierającej informacje na temat wybranych problemów wielokryterialnych oraz literatury opisującej algorytmy SI, które zostaną wykorzystane w pracy. Następnie obliczenia z wykorzystaniem własnych programów, opracowanie i interpretacja wyników.

Układ pracy:

- Studium literatury, w którym analizowane i omawiane są wybrane metody z pozycji literatury oraz podsumowanie tych rozważań,
- Eksperymenty, sprawdzanie różnic pomiędzy poszczególnymi algorytmami i wyniki,
- Zakończenie, podsumowanie pracy i opracowanie wniosków.

2. Studium literatury

2.1. Heurystyki, strategie poszukiwania i metaheurystyki

Pojęcie heurystyka pochodzi od *heurisco*, greckiego słowa znaczącego znajdować, okrywać. Najkrócej heurystykę można określić jako „twórcze rozwiązywanie problemów” matematycznych i logicznych, z użyciem metody prób i błędów albo wykorzystując analogie. Heurystycznych metod używa się zawsze tam, gdzie potrzeba dużych ilości obliczeń w celu rozwiązywania problemu. Heurystyka pomaga zmniejszyć obszar poszukiwań, co umożliwia znaczące obniżenie kosztów obliczeniowych i przyspieszyć rozwiązanie zadania. Nie istnieją co prawda w literaturze formalne dowody skuteczności metod heurystycznych, mimo tego skuteczność tych algorytmów jest potwierdzana w symulacjach. Są popularnie stosowane w systemach podejmowania decyzji, systemach ekspertowych i badaniach operacyjnych. Heurystykę można przybliżyć za pomocą występującego w literaturze przykładu. Upuszczono na ziemię szkło kontaktowe. Trzeba rozważyć następujące opcje jego poszukiwań:

1. Poszukiwanie ślepe - sprawdzanie po omacku. Nie daje gwarancji znalezienia.
2. Poszukiwanie systematyczne - w sposób metodyczny i zorganizowany jest rozszerzana przestrzeń do przeszukania. Znalezienie rozwiązania jest pewne lecz bardzo czasochłonne.
3. Poszukiwanie analityczne - brana jest pod uwagę fizyka np. prędkość spadania szkła, opór powietrza. Sukces pewny, ale ta metoda - niepraktyczna.
4. Poszukiwanie leniwe - kupno nowego szkła.
5. Poszukiwanie heurystyczne - na podstawie kierunku upadku szkła, oceniamy jak daleko może się ono znajdować, a potem próbujemy zlokalizować szkło w tym obszarze. Jest to metoda zazwyczaj wybierana przez człowieka, często nieświadomie.

Poszukiwanie ślepe tym różni się od heurystycznego, że nie korzysta z danych o dziedzinie problemu, który ma być rozwiązany. Szukanie heurystyczne wykorzystuje informacje dodatkowe na temat przestrzeni stanów i można określić jakie postępy poprawiają efektywność działania [4, 5].

Metaheurystyka opisuje sposób na utworzenie algorytmu heurystycznego. Można to określić jako zbiór reguł, takich jakie mogą być najbardziej przydatne w zgadnięciu optymalnego rozwiązania. Metaheurystyki to więc ogólne przepisy, wg których tworzone zostają reguły przeznaczone dla określonego rodzaju algorytmu. Dla jednej metaheurystyki zawsze można zastosować kilka różniących się między sobą algorytmów heurystycznych (będących różnymi opcjami spełniającymi reguły konkretnego spektrum). Z tego powodu algorytmy są nazywane instancjami metaheurystyk [6].

Hansen i Mladenowić umieścili w swej książce [7] listę wymaganych cech, która powinna charakteryzować wszystkie metaheurystyki:

- prostota - powinna istnieć zrozumiała i jasna zasada, na której została zbudowana metaheurystyka o dużym obszarze stosowania,
- spójność - wszystkie etapy heurystyki spełniają ogólną zasadę danej metaheurystyki, z której się wywodzą,
- precyzja - wszystkie etapy mają być określone dokładnymi w obliczeniach matematycznymi wzorami, bez względu na to jakim źródłem się inspirują,
- efektywność - zastosowanie heurystyk na bazie konkretnej metahurystyki powinno dawać częściowo dobre wyniki, nawet przy średnich nakładach mocy obliczeniowej,
- skuteczność - wszystkie albo większość problemów powinno być choć częściowo rozwiązywanych przez heurystyki, jeżeli te problemy są rozwiązywalne przez tę klasę heurystyk,
- żywotność - jakość wyników rozwiązań różnych problemów należących do konkretnej klasy powinna być zbliżona, czyli jeżeli heurystyka dobrze rozwiązuje jakiś problem danej klasy, to powinna dać wynik podobnie dobry wynik dla dowolnego, innego problemu z danej klasy,
- dostępność - łatwość i przystępcość zastosowania heurystyki dla użytkownika, żeby to uzyskać powinna mieć bardzo ograniczoną liczbę parametrów lub nie mieć ich wcale,
- innowacyjność - korzystnie jest jeśli zasada metaheurystyka i powiązane z nią heurystyki nie są sztywne lecz umożliwiają użycie ich do różnych, nowych zastosowań w razie potrzeby.

2.2. Optymalizacja wielokryterialna

Optymalizację wielokryterialną wykonywana jest wtedy, gdy optymalizowany problem posiada wiele kryteriów liczbowych podlegających ocenie, w przeciwieństwie do optymalizacji jednokryterialnej, gdzie funkcja zwraca jedną wartość. Minimalizacja wszystkich kryteriów nie może być przeprowadzona, ponieważ są one ze sobą sprzeczne (minimalizacja jednego kryterium może spowodować wzrost innych). Porównanie rozwiązań wektorowej funkcji kosztu, wykonać można, wprowadzając relację dominacji Pareto. Wg niej wektor $\vec{u} = (u_1, \dots, u_n)$ dominuje nad innym wektorem $\vec{v} = (v_1, \dots, v_n)$, wtedy gdy:

$$\forall i \in \{1, \dots, n\}, \quad \wedge \quad \exists i \in \{1, \dots, n\} : u_i < v_i \quad (1)$$

zakładając, iż problem dotyczy minimalizacji poszczególnych kryteriów. W związku z tym rozwiązanie Pareto-optymalne rozumiane jest jako metoda, dla której nie ma w przestrzeni zdarzeń innego rozwiązania, które by ją zdominowało [8].

2.3. Tradycyjne metody optymalizacji wielokryterialnej

2.3.1. Metoda kryteriów ważonych

Metoda wykorzystuje funkcję skalarną będącą sumą ważoną wartości badanych kryteriów. Zatem optymalizację wielokryterialną wykonuje się za pomocą wzoru optymalizacji skalarnej:

$$f_c = \sum_{i=1}^k w_i * f_i(\vec{x}) \quad (2)$$

gdzie: $w_i \geq 0$ to współczynniki wagowe, określające ważność konkretnego kryterium.

Dodatkowo, zazwyczaj przyjmuje się, że:

$$f_c = \sum_{i=1}^k w_i = 1 \quad (3)$$

Efektem tych przekształceń jest pojedyncza funkcja celu $F(x)$, możliwa do optymalizacji za pomocą metod przeznaczonych dla zadań jednokryterialnych. Metoda ta jest efektywna i ma niewielką złożoność obliczeniową. Głównym minusem są problemy z właściwym dobraniem współczynników wagowych, w przypadku, jeśli jest ograniczona ilość informacji na temat rozwiązywanego zadania optymalizacyjnego [9].

2.3.2. Metoda hierarchiczna

Metoda ta polega, tak jak poprzednia, na doprowadzeniu optymalizacji wielokryterialnej do optymalizacji jednokryterialnej, która będzie stosowana po kolei wobec wszystkich kryteriów. Należy postępować wg poniższej instrukcji:

1. posegregować kryteria od najważniejszego (f_1) do mało ważnego (f_m),
2. poszukać rozwiązania optymalnego X_1 dla kryterium f_1 z pierwotnymi ograniczeniami,
3. znaleźć optymalne rozwiązania X_i , gdzie $i = 2, 3, \dots, M$ dla reszty kryteriów, uwzględniając dodatkowe ograniczenia:

$$f_{i-1}(X) \leq (1 \pm \varepsilon_{i-1}) * f_{i-1}(x_{i-1}) \quad (4)$$

gdzie: ε to wartość wariancji (wyrażona w procentach), która jest dozwolona dla funkcji f_i . Określa ona jak „ważne” jest wyznaczone w poprzednim punkcie optimum. Jeżeli ta wariancja jest równa zero, to metoda optymalizacyjna określana jest jako metoda leksykograficzna.

2.3.3. Metoda ograniczonych kryteriów

Polega na ustaleniu skali ważności pojedynczych kryteriów, dzięki czemu ilość możliwych rozwiązań problemu jest redukowana. W tej metodzie optymalizacja wielokryterialna zostaje zmieniona na optymalizację określonego kryterium f_r , z powiększoną o $M - 1$ ilością ograniczeń - efekt uwzględnienia pozostałych kryteriów. Operacje te można zapisać w ten sposób:

$$f_r(X) \rightarrow MIN$$

$$f_i(X) \leq \varepsilon_i, i = 1, \dots, M, i \neq r$$

$$g_k(X) \leq 0, k = 1, \dots, K$$

$$h_j(X) = 0, j = 1, \dots, J \quad (5)$$

gdzie: ε_i to wartość ograniczająca poszczególne kryteria, $g_k(X)$ to ograniczenia nierównościowe, a $h_j(X)$ - ograniczenia równościowe.

2.3.4. Metoda kryterium globalnego

W tym przypadku zadaniem jest znalezienie przyblizonego rozwiązania $F(X^*)$, w celu określenia kryterium dla optymalizacji jednokryterialnej określonej wzorem:

$$\sum_{i=1}^M \left(\frac{f_i(X^*) - f_i(X)}{f_i(X^*)} \right)^P \rightarrow MIN \quad (6)$$

Muszą być uwzględniane ograniczenia zarówno równościowe jak i nierównościowe. Zwykle P reprezentuje wartość z przedziału od 1 do 2 [10, 11, 12].

2.4. Inspiracja naturą: algorytmy rojowe

Wiele algorytmów swój początek ma w obserwacji zachowań społecznych zwierząt, które wykazują się inteligencją grupową. Inteligencja roju to nauka o systemach obliczeniowych powstały w oparciu o „inteligencję zbiorową”. Taka inteligencja pojawia się przy współpracy wielu członków roju w jednorodnym środowisku. Jako przykład można podać ławice ryb, klucze ptaków i kolonie mrówek. Inteligencja ta jest zdecentralizowana, organizuje się sama i występuje w całym środowisku. W naturze, ten rodzaj zachowania powszechnie wykorzystywany jest do rozwiązywania problemów ze znalezieniem pożywienia, ochroną przed drapieżnikami lub przemieszczania się całej społeczności. Informacje są przekazywane przez różne zachowania grup osobników np. feromony mrówek albo taniec pszczół.

Inspirowane naturą algorytmy wykorzystuje się do rozwiązywania dyskretnych problemów optymalizacji, wykrywania anomalii (nietypowe zachowania programów), rozwiązywania problemów skomplikowanych obliczeniowo [13, 14, 15].

2.4.1. Algorytmy kolonii pszczół

Algorytmy sztucznej kolonii pszczół (ABC, MBO, BCO) opierają się na zachowaniu pszczół miodnych podczas zdobywania pożywienia. Jest to jeden z najinteligentniejszych rojów występujących w naturze.

Dervis Karaboga opracował model roju pszczelego Artificial Bee Colony (ABC) [16] zawierający trzy główne elementy: źródła pożywienia oraz pszczoły zatrudnione i niezatrudnione. Pszczoły zatrudnione mają związek ze źródłem pożywienia. Koniec jedzenie się skończy, wówczas pszczoła robotnica staje się niezatrudnioną. Takie niezatrudnione pszczoły nie posiadają wiedzy o źródłach pożywienia, więc szukają nowych źródeł, by

mówić o pobieraniu z nich pokarmu. Wyróżnia się dwa rodzaje pszczół niezatrudnionych: zwiadowcy i widzowie. Zwiadowcy szukają pożywienia na chybił trafił w okolicach ula. Widzowie natomiast, na podstawie obserwacji tańca wykonywanego przez pszczoły zatrudnione, dokonują wyboru źródła pożywienia. Istotnym elementem jest oczywiście bogactwo danego źródła. W porównaniu z kontekstem optymalizacyjnym, ilość źródeł pożywienia w algorytmie ABC zależy od ilości rozwiązań w populacji. Dodatkowo, miejsce w którym znajduje się pożywienie informuje o współrzędnych rozwiązania problemu, który ma być optymalizowany. Nektar i jego ilość pełni rolę przy ocenianiu jakości rozwiązania, jeśli jest go więcej to oznacza, że rozwiązanie jest lepsze.

W algorytmie ABC wyróżnia się trzy kroki główne w procesie szukania rozwiązania:

- Posyłanie pszczół zatrudnionych do eksploatacji źródeł pożywienia i określanie jakości nektaru,
- Na podstawie informacji przekazanych przez pszczoły zatrudnione, pszczoły widzowie dokonują wyboru źródła pożywienia,
- Wybranie pszczół, które będą zwiadowcami, poszukiwaczami nowych źródeł.

Pełen proces poszukiwania sprowadzić można do postaci schematu blokowego, przedstawionego na rys. 2.4.1. W momencie aktualizacji położenia pszczół, używana jest iteracja, w postaci procesu, znajdującego się na rys. 2.4.2.

Inicjalizacja pszczelego roju przebiega w następujący sposób. Algorytm ma trzy parametry: P oznacza ilość źródeł pożywienia, M to ilość testów, prób, które spowodowały, że konkretne źródło pożywienia się skończyło, C_{max} to maksymalna ilość powtórzeń działania algorytmu. W algorytmie Karabogi, liczba pszczół zarówno zatrudnionych, jak i będących widzami jest równa liczbie źródeł pożywienia. Najpierw określone są współrzędne źródeł pożywienia wg wzoru:

$$x_{ij} = x_{\min j} + rand[0, 1](x_{\max j} - x_{\min j}) \quad (7)$$

gdzie: $x_i (i = 1, \dots, P)$ to następne położenie źródła pożywienia znajdującego się w D-wymiarowej przestrzeni ($j = 1, \dots, D$), $rand[0, 1]$ generuje liczby losowe z zakresu $[0, 1]$.

Otrzymane współrzędne należą do populacji potencjalnych rozwiązań, a populacja ta będzie na bieżąco aktualizowana przez pszczoły.

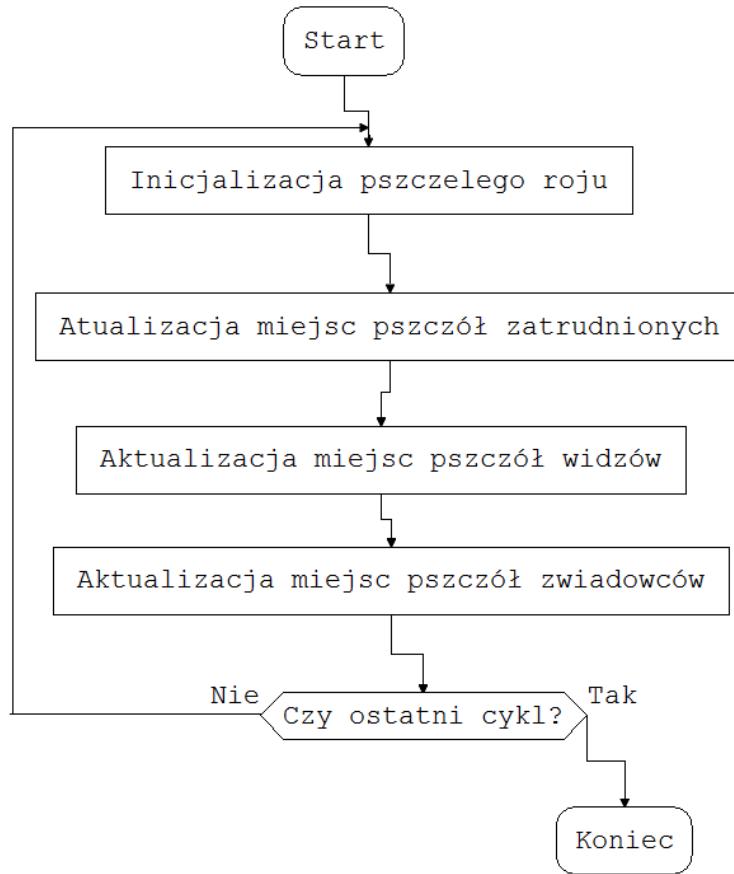
Zwiadowcą pszczoła zostaje, wtedy kiedy po M próbach testowych modyfikacji współrzędnych miejsca w którym jest pożywienie, nie została poprawiona jakość rozwiązania.

Dla pszczół zatrudnionych pierwsze wykonywane działanie to sprawdzanie ilości nektaru $F(x_i)$, w oparciu o współrzędne wyliczone ze wzoru (7). Potem współrzędne źródeł jedzenia zostają zaktualizowane, opierając się o współrzędne miejsca, w którym znajduje się pszczoła i współrzędne reszty robotnic, z wykorzystaniem takiej zależności:

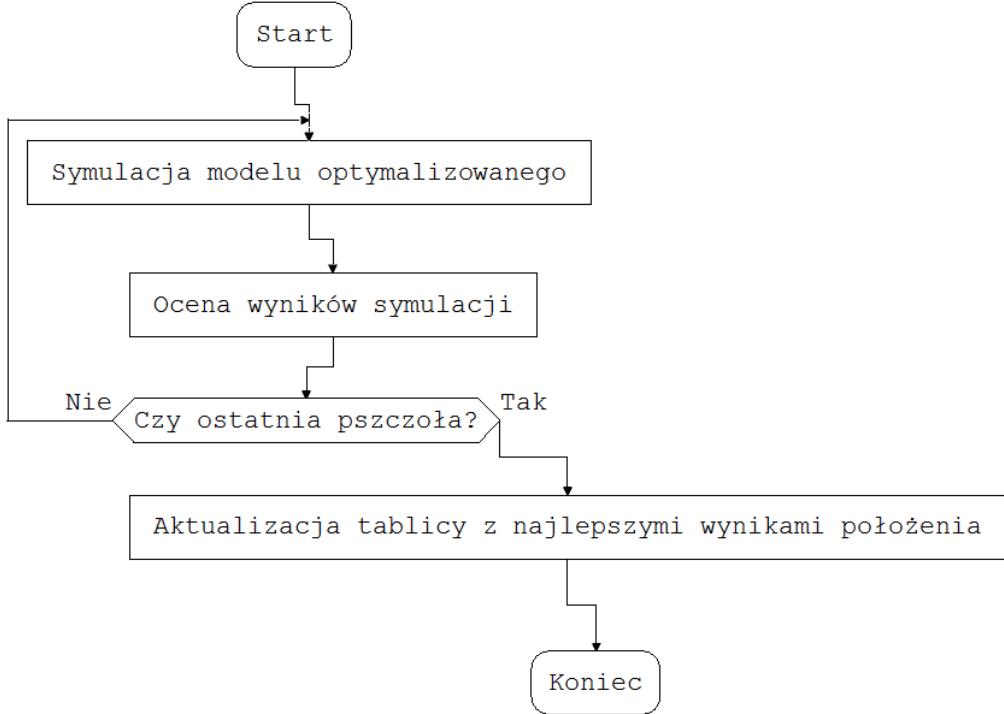
$$v_{ij} = x_{ij} + \text{rand}[-1, 1](x_{ij} - x_{kj}), i = 1, \dots, P, \quad k \neq i \quad (8)$$

gdzie: $j \in \{1, 2, \dots, D\}$, $k \in \{1, 2, \dots, P\}$ to dwa indeksy losowo wyznaczone

Za każdym razem, kiedy wyznaczone zostaną nowe współrzędne ze źródłami pożywienia V_i trzeba sprawdzić ilość nektaru $F(V_i)$. Porównywana jest ilość nektaru źródeł. W wypadku, kiedy więcej jest go w nowym źródle $F(V_i) > F(x_i)$, to współrzędne źródła pożywienia dla i -tej pszczoły zostają zmienione ($x_i = V_i$).



Rysunek 2.4.1: Schemat blokowy z kolejnymi fazami algorytmu kolonii pszczół



Rysunek 2.4.2: Schemat blokowy wykonywany dla wszystkich faz algorytmu kolonii pszczół

Źródła pożywienia dla pszczół widzów mają taką samą liczebność, co źródła przeznaczone dla pszczół zatrudnionych. Na tym etapie obliczeniowym psy zatrudnione wracając do ula, przekazują pszczołom widzom informacje dotyczące nektaru i jego ilości $F(x_i)$. Na wybór źródła pożywienia przez psy widzów ma wpływ to, jak dużo w źródle tym jest nektaru $F(X_i)$. Wraz ze wzrostem ilości nektaru w danym źródle, rośnie prawdopodobieństwo na to, że psy widzowie wybiorą właśnie to konkretne źródło pożywienia. Wzór na prawdopodobieństwo wyboru prezentuje się następująco:

$$p_i = \frac{F(x_i)}{\sum_{i=1}^P F(x_i)}, \quad i = 1, \dots, P \quad (9)$$

Pszkoły widzowe oglądają taniec robotnic, a kiedy dobiegnie on końca, opuszczają ul i lecą w kierunku źródła pożywienia x_i , wybranego po analizie prawdopodobieństwa. Zwykle psy chcą uwzględnić własne obserwacje i zatrzymują się w okolicy obranego źródła pożywienia. Inaczej mówiąc, zwiadowcy dokonują wyboru, na podstawie porównywania źródeł pożywienia x_i . Korzystając ze wzoru (8) obliczane są współrzędne źródła będącego w okolicy. W przypadku, kiedy w nowym źródle jest więcej nektaru $F(V_i) > F(x_i)$, to wtedy dane o pozycji źródła zostają uaktualnione dla i -tego pszczelego widza ($x_i = V_i$).

Rozważeniu należy jeszcze poddać przypadek, w którym źródło pożywienia nie jest zmieniane po predefiniowanej ilości cykli (M). Wtedy przyjmuje się, że należy porzucić dane źródło pokarmu i przejść do fazy zwiadowcy. Podczas tego etapu pszczoła zrywa relację ze starym źródłem pożywienia i zostaje zwiadowcą, którego nowe źródło pokarmu zostaje wybrane z puli źródeł znajdujących się w danym, przeszukiwanym polu rozważań. Wspomniana predefiniowana ilość cykli (M) to ważna wartość występująca w algorytmie sztucznej kolonii pszczół, parametr ten określany jest jako tzw. granica odrzucenia. Pszczoły zwiadowcy porzucają stare źródło x_i , zastępując je innym, do którego wyznaczenia należy użyć wzoru[16]:

$$x_{ij} = x_{\min j} + \text{rand}[0, 1](x_{\max j} - x_{\min j}) \quad (10)$$

gdzie: $j = 1, \dots, D$, natomiast $\text{rand}[0, 1]$ to generator liczb losowych z zakresu [0,1].

2.4.2. Algorytmy optymalizacji rojem częstek

Algorytm optymalizacyjny roju częstek (PSO-Particle Swarm Optimization) to technika obliczeniowa, która została opracowana na wzór zachowania, które zaobserwowano u ptaków i ryb w ławicach. Zachowania te polegają na tym, że poszczególni członkowie w stadzie starają się tak dostosować prędkość ruchu, aby utrzymywać określony, najkorzystniejszy dystans do sąsiednich osobników. Korzyść z tego modelu zachowania jest taka, że wszyscy członkowie reagują jednocześnie, co zapobiega kolizjom, umożliwia szybkie zmiananie kierunku ruchu całej grupy i sprawniejsze przemieszczanie się, szczególnie wtedy, gdy zachodzi potrzeba wykonania np. zwrotu, który wymaga reorganizacji układu całego ”oddziału”.

Propozycja algorytmu PSO autorstwa Ebercharta i Kennedy'ego [17], została opracowana na wzór zachowań zwierząt, które mają poprawić bezpieczeństwo stada, ułatwić mu poszukiwanie jedzenia i poprawić jego mobilność. Z poziomu algorytmu, rój znajduje się w przestrzeni posiadającej D wymiarów, poruszając się z losowo określonymi pozycjami i prędkościami, jednak wiadoma jest ich wartość najkorzystniejsza.

Można teraz zastanowić się nad pozycją i -tej częsteczki $X_{i,m}$, przemieszczającej się wewnątrz D wymiarowej przestrzeni. Zapisywana jest jako $P_{best,i,m}$ najkorzystniejsza i najlepsza pozycja i -tej częstki. Najlepsza spośród częstek w populacji jest określana jako $g_{best,i,m}$ i zapisywana, zaś najlepsza z częstek występujących w najbliższym sąsiedztwie to

$Lbest_{i,m}$. Prędkość poruszania się poszczególnych cząsteczek znajdujących się w przestrzeni rozważań zapisywana jest jako $V_{i,m}$. Prędkości oraz pozycje aktualizowane są zależnie od obliczeń do których wykorzystywane są pozycje i prędkości bieżące [18, 19].

Miejsce, w przestrzeni x_i o D wymiarach, w którym znajduje się cząstka jest opisywane w sposób następujący:

$$x_i = (X_{i,1}, X_{i,2}, \dots, X_{i,D}), \quad i = 1, \dots, N \quad (11)$$

gdzie: N to ilość cząsteczek w roju

Zapamiętywanie najkorzystniejszej pozycji i -tej cząsteczki $Pbest_i$:

$$Pbest_i = (Pbest_1, Pbest_2, \dots, Pbest_D) \quad (12)$$

Najlepsza cząstka w populacji, czyli o najkorzystniejszym wskaźniku $Pbest_i$, po zapisaniu określana jest jako $gbest$. V_i , czyli prędkość cząstki jest zapisywana w postaci:

$$V_i = (V_{i,1}, V_{i,2}, \dots, V_{i,D}) \quad (13)$$

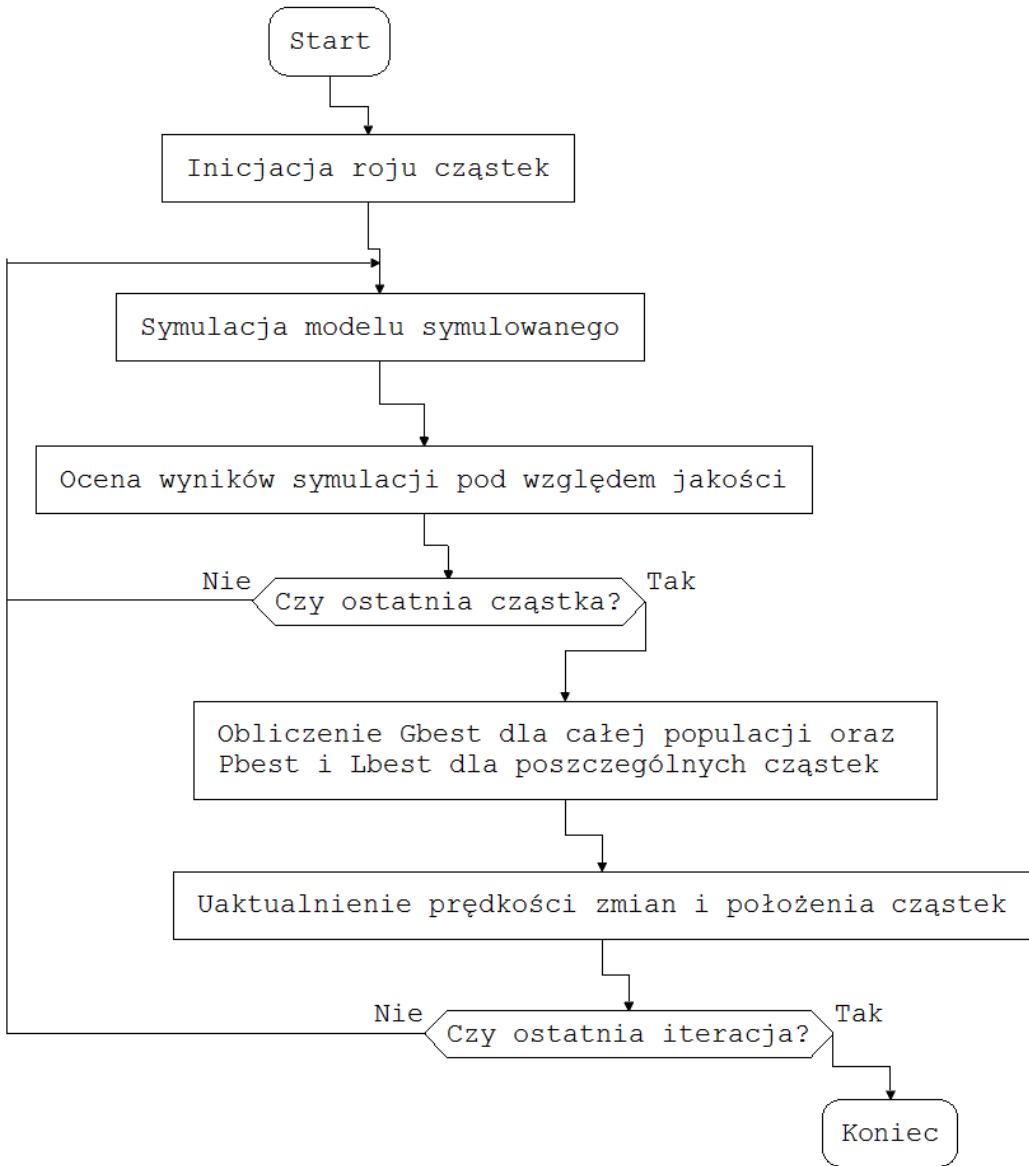
Wykorzystując różnicę odległości pozycji i -tej cząsteczki x_i od współrzędnych $Pbest_i$ oraz $Lbest$, uaktualniana jest pozycja i prędkość każdej kolejnej cząstki, odbywa się to z wykorzystaniem wzorów:

$$V_{i,m}^{(t+1)} = w * V_{i,m}^{(t)} + c_1 rand[0, 1] * (Pbest_{i,m} - x_{i,m}^{(t)}) + c_2 rand[0, 1] * (Lbest_m - x_{i,m}^{(t)}) \quad (14)$$

$$x_{i,m}^{(t+1)} = x_{i,m}^{(t)} + V_{i,m}^{(t+1)}, \quad i = 1, \dots, N; \quad m = 1, \dots, D \quad (15)$$

gdzie: w to wagowy współczynnik inercji, c_1, c_2 to stałe przyspieszenia, $rand[0, 1]$ to generator liczb losowych z zakresu $[0,1]$.

Proces wyznaczania pozycji cząsteczki przedstawić można na układzie o dwóch wymiarach. Na początku określany jest nowy wektor V^{k+1} (wektor prędkości) dla cząstki x^k . Jest to obliczane z wykorzystaniem bieżącej pozycji tej cząstki i pozycję $Pbest$ oraz $Lbest$. Wyznaczony wektor jest niezbędny do określenia nowych współrzędnych w kolejnej instrukcji danej iteracji algorytmu x^{k+1} . Na rysunku 2.4.3 przedstawiono schemat blokowy omawianego algorytmu optymalizacji rojem cząstek [20].



Rysunek 2.4.3: Algorytm PSO na schemacie blokowym

2.4.3. Algorytm świetlików

Algorytm GSO (Glowworm Swarm Optimization), czyli algorytm świetlików został wymyślony przez Xin-She Yang'a w Cambridge University w 2007 roku. Algorytm ten jak sama nazwa wskazuje wzoruje się swym działaniem na zachowaniu robaczków świętojańskich. "Świecenie" tych owadów ma za zadanie wabić ofiary, przestrzegać wrogich osobników przed zbliżaniem się oraz jest wykorzystywane w zalotach. Elementem, który wykorzystuje się w omawianych algorytmach są zmiany w natężeniu światła emitowanego przez świetliki, co określa jaki jest cel wysyłania sygnału świetlnego. Jeśli jakiś z owadów świeci jaśniej, reszta, o mniej intensywnym sygnale będzie zbliżała się do niego, a to z kolei

umożliwia wydajniejsze zbadanie przez algorytm przestrzeni poszukiwań [21, 22].

W algorytmie GSO obowiązują następujące zasady [21]:

- zarówno osobniki żeńskie jak i męskie są uważane za atrakcyjne,
- to jak bardzo dany osobnik jest atrakcyjny zależy od blasku emitowanego przez niego światła, im odległość między osobnikami większa, tym intensywność świecenia coraz mniejsza, kiedy osobniki są jednakowo atrakcyjne, przemieszczają się w losowych kierunkach,
- funkcja celu determinuje swą wartością, jakie jest natężenie wysyłanego przez świetnika sygnału świetlnego.

Atrakcyjność jest cechą właściwą dla każdego świetnika, istnieją jednak różnice w poziomie atrakcyjności poszczególnych osobników. Atrakcyjność określa funkcja dystansu między wybranymi dwoma owadami:

$$\beta(r) = \beta_0 e^{-\gamma r^m}, \quad m \geq 1, \quad (16)$$

gdzie: β_0 to atrakcyjność kiedy $r = 0$, γ to wartość współczynnika absorpcji promieniowania świetlnego.

Wspomniany wcześniej dystans pomiędzy wybraną parą świetników (i, j) , zajmujących pozycje x_i oraz x_j można obliczyć za pomocą wzoru:

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^d (x_{i,k} - x_{j,k})^2}, \quad (17)$$

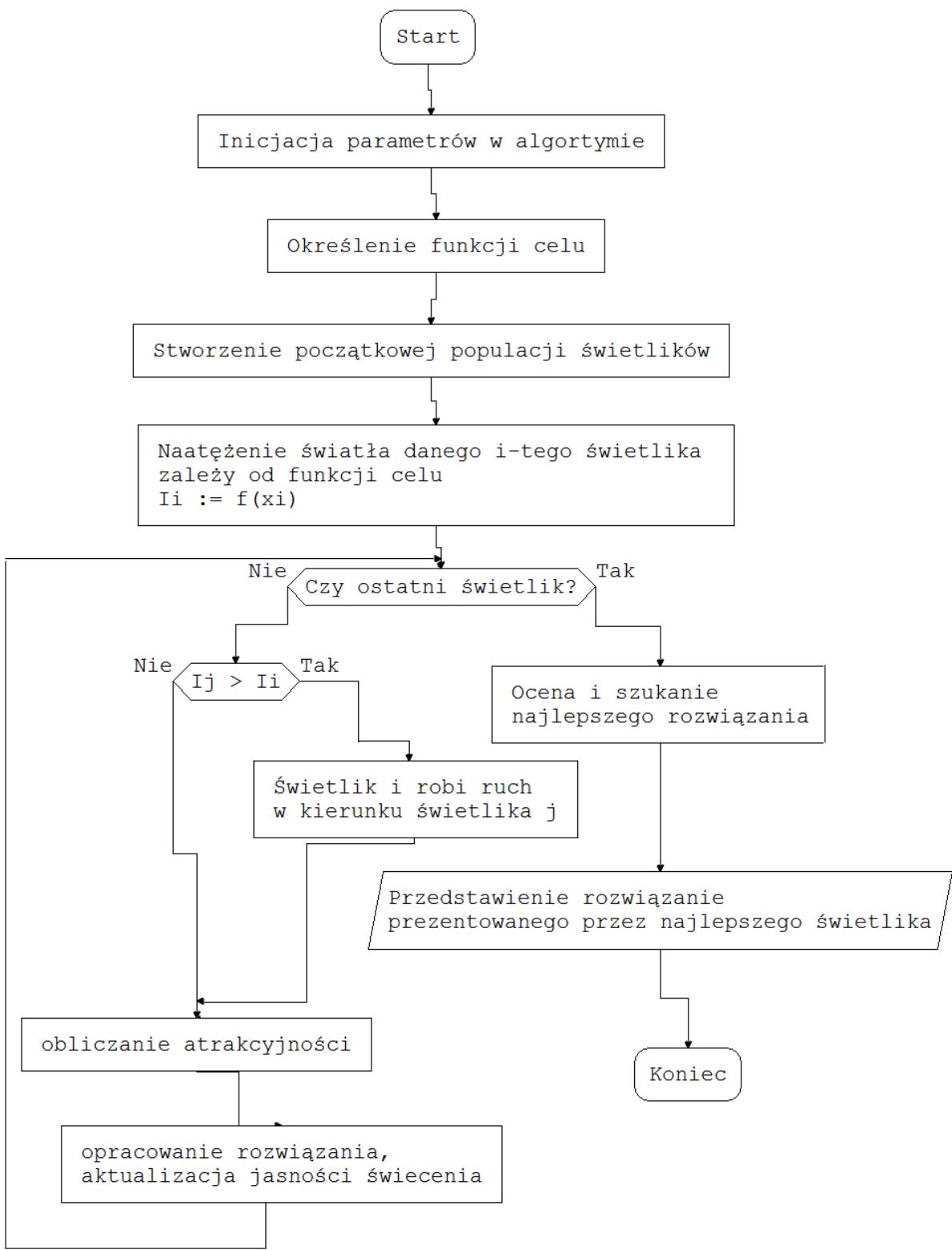
gdzie: d to ilość wymiarów.

Świetlik i porusza się w sposób określony następującym wzorem:

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2} (x_j - x_i) + \alpha (rand - 0, 5), \quad (18)$$

gdzie: x_i to aktualne współrzędne świetnika i , drugi element sumy stanowi o atrakcyjności, a ostatni wykorzystywany jest, jeżeli występuje losowa zmiana położenia; $rand$ to losowo generowana wartość $[0, 1]$, natomiast $\alpha \in (0, 1)$. Zwykle β_0 i γ przyjmują wartość 1.

Budowa algorytmu GSO została przedstawiona na schemacie blokowym (rys. 2.4.4) [23, 21]:



Rysunek 2.4.4: Algorytm świetlika na schemacie blokowym

2.4.4. Algorytm nietoperza

Bat algorithm (BA) czyli algorytm nietoperza to metoda metaheurystyczna, zaproponowana przez Yang'a w 2010 roku [21, 24]. Zdolność echolokacji nietoperzy to fascynująca rzecz, ponieważ pomaga ona znaleźć nietoperzom zdobycz oraz rozpoznawać różne rodzaje owadów w zupełnej ciemności [25]. Oparty o echolokację nietoperzy algorytm, prowadzi proces poszukiwań za pomocą sztucznych odpowiedników nietoperzy, które wysyłają impulsy o odpowiedniej częstotliwości i głośności, podobnie jak to ma miejsce w naturze. Kiedy zwierzęta te gonią swą zdobycz, natężenie impulsów jest zmniejszane, a rośnie ich częstotliwość.

Algorytm nietoperza jest wydajny w przypadku optymalizacji danych o małej komplikacji parametrów [26, 27, 28], szeroko się go używa w optymalizacji inżynierijnej [29] i wieloobiektowej [30]. Jednak ze względu na niską różnorodność populacji, traci na wydajności przez konwergencję w przypadku optymalizacji problemu trudnego [31]. Powstały różne warianty algorytmu nietoperza, starające się zwiększyć różnorodność populacji, żeby uniknąć uwięzienia w optimum lokalnym.

Echolokacja jest istotną cechą charakteryzującą nietoperze. Yang odwzorował ich charakterystykę w swym algorytmie. Nietoperze latają z użyciem echolokacji aby uniknąć przeszkód i zlokalizować pożywienie. W celu przekształcenia zachowania zwierząt na działanie algorytmu, trzeba dokonać pewnych uproszczeń i zastosować wyidealizowane reguły [24].

- Wszystkie nietoperze używają echolokacji do określania dystansu od obiektu i potrafią rozróżnić, czy konkretny obiekt jest przeszkodą, czy potencjalnym pożywieniem.
- Nietoperze latają losowo z prędkością v_i , znajdując się w miejscu x_i , emitują fale o stałej częstotliwości f_{min} , różej długości λ i głośności A_0 , żeby szukać zdobyczy. Mogą automatycznie dostosowywać długość fali lub częstotliwość emitowanych impulsów, a także regulować szybkość emisji sygnałów $r \in [0, 1]$, w zależności od bliskości celu.
- Mimo, iż poziom głośności może być różny pod wieloma względami, zakłada się, że głośność może przyjąć wartości od dużej (dodatniej) A_0 do minimalnej stałej A_{min} .

W algorytmie BA, dla i -tych nietoperzy roju, jest określana pozycja (rozwiążanie) x_i , prędkość v_i i częstotliwość f_i , każdy nietoperz przemieszcza się w kierunku najlepszej ak-

tualnej pozycji (rozwiązań), a jego pozycja, prędkość oraz częstotliwość są aktualizowane podczas kolejnych iteracji następująco:

$$f_i = f_{min} + (f_{max} - f_{min})\beta$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x_g^{t-1})f_i \quad (19)$$

$$x_i^t = x_i^{t-1} + v_i^t$$

gdzie: β jest liczbą losową równomiernego rozmieszczenia o wartości $[0,1]$, a x_g^{t-1} reprezentuje aktualnie najlepsze globalne rozwiązanie (pozycję) po porównaniu wszystkich rozwiązań (pozycji) spośród wszystkich n nietoperzy. Te równania mogą zagwarantować zdolności poszukiwawcze algorytmu BA.

Podczas szukania lokalnego, kiedy rozwiązanie jest wybierane ze zbioru najlepszych, może zostać wygenerowane nowe rozwiązanie kandydujące, wg wzoru:

$$x_{new} = x_{old} + \varepsilon \bar{A}^t \quad (20)$$

gdzie: ε jest liczbą losową z zakresu $[0,1]$ i określa nowe rozwiązanie, które jest odmienne lub zbliżone do aktualnego najlepszego rozwiązania, a \bar{A}^t to średnia wartość głośności sygnałów wszystkich nietoperzy.

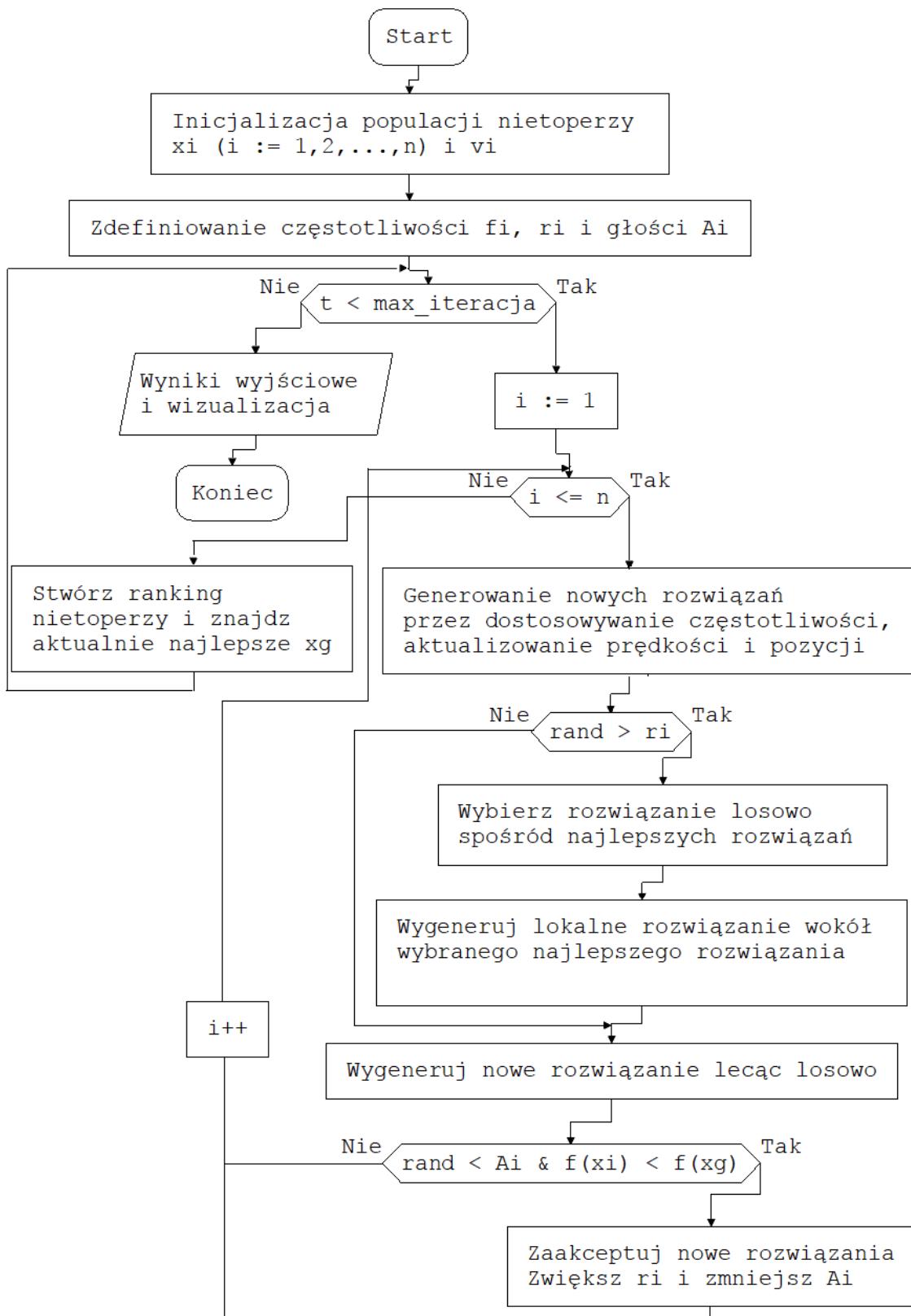
Gdy nietoperz znajdzie cel, stopniowo zmniejsza głośność impulsów i zwiększa szybkość ich emisji, żeby śledzić zdobyty i pochwycić ją. Poziom głośności oraz częstości występowania sygnałów jest aktualizowany w czasie wykonywania procesu, przebiegającego iteracyjnie:

$$A_i^t = \alpha A_i^{t-1}$$

$$r_i^t = r_i^0 + (1 - \exp(-\gamma t)) \quad (21)$$

gdzie: α i λ są stałe. Parametr α kontroluje zbieżność algorytmu.

Podstawowe etapy algorytmu BA można przedstawić w postaci schematu blokowego przedstawionego na rys. 2.4.5 [32].



Rysunek 2.4.5: Schemat blokowy Bat algorithm

2.5. Inspiracja naturą: algorytmy ewolucyjne

Algorytmy ewolucyjne wywodzą się od procesów zachodzących naturalnie, gdzie procesy szukania rozwiązania zadania są związane z teorią Darwina o selekcji naturalnej. Opis działania algorytmów ewolucyjnych oraz pojęcia związane z tymi algorytmami są ścisłe powiązane z ewolucją i genetyką. Uogólniając, algorytm ewolucyjny działa na populacji składającej się z P osobników, z których każdy zawiera chromosom, będący określonym sposobem rozwiązania zadania. Algorytm ewolucyjny działa w przestrzeni, czy środowisku, które jest determinowane przez rodzaj problemu, który ma być przez ten algorytm rozwiązany. Im bardziej określony osobnik jest dostosowany do danego środowiska (ma większe prawdopodobieństwo przeżycia w tym środowisku), tym jakość sposobu rozwiązania problemu, który prezentuje, jest wyższa i otrzymuje lepszą "ocenę". Tak przydzielona ocena jest zwana przystosowaniem osobnika. Wybrany osobnik posiada genotyp, reprezentujący dane w postaci kodu. Z genotypu, dzięki zawartej w nim instrukcji, jest tworzony fenotyp, czyli już odkodowane, możliwe rozwiązanie zadania. Fenotypy również muszą być oceniane w środowisku. Odbywa się zatem kodowanie fenotypu wykonywane przez genotyp (czasem, w niektórych algorytmach ewolucyjnych pojęcie fenotypu jest tożsame z genotypem). Streszczając, fenotyp to punkt znajdujący się w przestrzeni zawierającej rozwiązania problemu, a genotyp to punkt w przestrzeni zawierającej kody. Wzorcowego osobnika o binarnej reprezentacji genotypu i jego fenotyp przedstawiono na rys. 2.5.6.

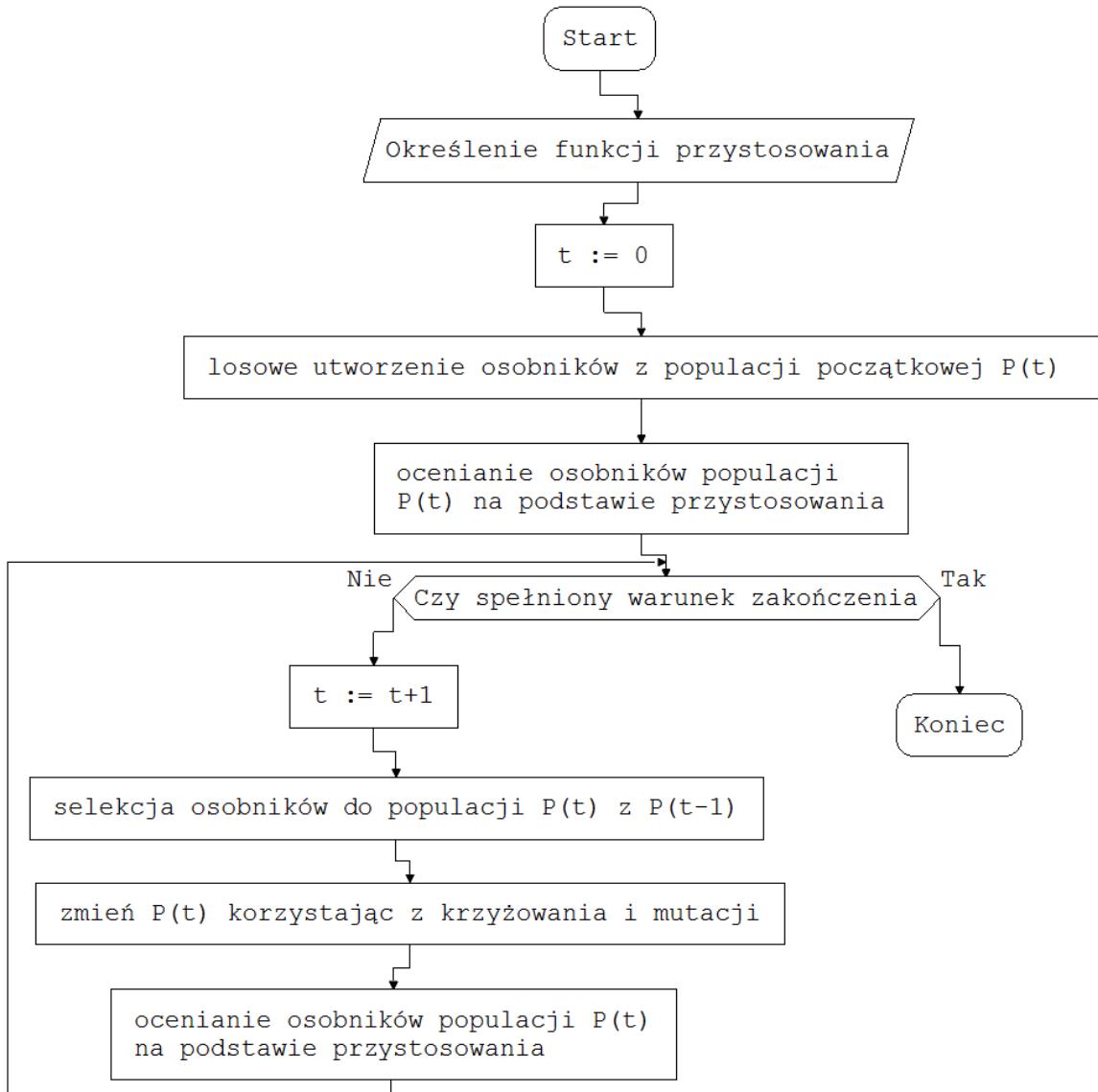
Genotyp	Fenotyp
0 1 1 0 1 0 0 1	105

Rysunek 2.5.6: Osobnik wzorcowy z genotypem i fenotypem

Środowisko może zostać scharakteryzowane funkcją przystosowania, dzięki której mając na uwadze fenotyp osobnika, przypisywane jest mu przystosowanie. Chromosomy osobników zaś, zbudowane są z częstek nazywanych genami. Wyróżnia się także allele, czyli wartości danego genu. W przedstawionym wyżej przykładzie allelami są 1 i 0.

Na rysunku 2.5.7 przedstawiono schemat blokowy algorytmu ewolucyjnego. Po strukturze tego algorytmu widać, że należy on do algorytmów probabilistycznych, którego

działanie polega na utworzeniu populacji osobników $P(t) = x_1^t, \dots, x_n^t$ dla każdej iteracji t . Poszczególne osobniki zawierają różne sposoby rozwiązania problemu i zwykle występują jako chromosomy jednowarstwowe, czyli struktury S . Aby ocenić poszczególne rozwiązania x_i^t wprowadza się jakąś skalę przystosowania chromosomu. Wobec tego w tzw. fazie selekcji (iteracja $t + 1$) generowana jest nowa populacja wyselekcyjowana z najlepszych osobników.



Rysunek 2.5.7: Algorytm ewolucyjny na schemacie blokowym

W kolejnej fazie, fazie zmiany, niektóre osobniki poddawane są transformacji przez operatory genetyczne, rezultatem czego pojawiają się nowe rozwiązania. Wyróżnia się transformacje jednoargumentowe m_i , polegające na tworzeniu osobników dzięki niewielkiej zmianie jednego osobnika (mutacja) oraz transformacje o wielu argumentach (wieloar-

gumentowe c_j), polegające na tworzeniu osobników (przyjmujących postać jednowarstwowych chromosomów) dzięki złożeniu fragmentów kilku osobników[33]. Program napisany na podstawie algorytmu wykonuje kilka kroków generacji, ilość sensownych rozwiązań maleje, a rozwiązanie prezentowane przez osobniki najlepsze jest bardzo zbliżone do optymalnego.

2.5.1. Algorytmy ewolucyjne: reprezentacja osobników

Konstruowanie algorytmu ewolucyjnego wymaga doboru osobników, reprezentujących pulę dostępnych rozwiązań. Jedne problemy przy użyciu konkretnych reprezentacji osobników będą ciężkie do rozwiązania, inne przy tej samej reprezentacji okażą się być proste [34]. W najbardziej klasycznym algorytmie genetycznym, reprezentacja przyjmuje postać ciągów binarnych o jednej warstwie. Jeżeli optymalizacja jest parametryczna i ma ciągłe dziedziny, to może przyjąć formę liczb rzeczywistych. Ogólnie dokonanie właściwego wyboru reprezentacji, która będzie odpowiednia dla danego problemu jest trudne [35], podobnie jak samo wybranie odpowiedniego algorytmu do rozwikłania określonego problemu. Poza tym stwierdzono [36], że dowolne reprezentacje są tak samo dobre w zastosowaniu do wszystkich problemów. Wniosek z tego taki, że najlepszą metodą znalezienia reprezentacji jest eksperyment, porównujący tę reprezentację z innymi w jakichś ustalonych warunkach.

Najpowszechniej osobniki kodowane są przez liczby rzeczywiste i ciągi binarne. Jeżeli kodowanie jest binarne, istotne staje się by chromosom był odpowiednio długi, tak żeby prezentowane rozwiązanie było wystarczająco dokładne.

2.5.2. Algorytmy ewolucyjne: Funkcja przystosowania

Można powiedzieć, że funkcja przystosowania to spojwo łączące algorytm ewolucyjny z problemem, który ma być przez niego rozpracowany. Zadaniem tej funkcji jest dokonanie porównania poszczególnych osobników populacji i wyselekcjonowanie tych lepszych, żeby miały zwiększone szanse na przetrwanie i dostały się do nowej populacji, co jest równoznaczne z przekazaniem swoich genów potomkom. Trzeba zaznaczyć, że istnieje różnica między funkcją przystosowania i funkcją kosztu. Funkcja przystosowania zwykle powstaje dzięki funkcji kosztu, ale oprócz niej może zawierać różne składniki. Nadmienić można tu o optymalizacji z ograniczeniami wymagającej różnej oceny niedopuszczalnych

rozwiązań. W takim wypadku funkcja przystosowania może wymierzyć karę, oczywiście funkcja posiada potrzebne czynniki, które odpowiadają za kontrolę ograniczeń. W optymalizacji o wielu kryteriach jest generowana funkcja przystosowania w postaci sumy ważonej, ponieważ różne aspekty problemu mają różny priorytet i ważność. Druga opcja przewiduje występowanie procesu wyznaczania rang wszystkim kryteriom zależnych od stopnia niezdominowania.

Rozpatrując wariant pozbawiony ograniczeń, w którym funkcja przystosowania ma taką samą wartość jak funkcja kosztu, trzeba obliczyć funkcję i jej wartość indywidualnie dla poszczególnych osobników. Dokonuje się tego poprzez ustalenie fenotypu na podstawie genotypu. Oto przykład zamiany genotypu w fenotyp: osobnik ma genotyp 1100101, liczba genów (LG) to 7, zmienna $x \in [-1; 1]$ jest kodowane przez osobnika przy dokładności $\alpha = 0,001$, w tym wypadku wzór na konwersje wygląda następująco:

$$fenotyp = x_{min} + \left(\frac{x_{max} - x_{min}}{2^{LG} - 1} \right) \cdot dec(genotyp) \quad (22)$$

gdzie: dec to wartość liczby binarnej zamienionej na dziesiętną.

obliczony wg powyższego wzoru fenotyp:

$$x = -1 + \left(\frac{1 + 1}{2^7 - 1} \right) \cdot dec(1100101) = 0,59055 \quad (23)$$

Otrzymany fenotyp przekazywany jest funkcji przystosowania, decydującej o tym jaką jakość ma rozwiązanie prezentowane przez osobnika.

Sporo problemów lepiej wyraża się w minimalizacji jakiejś funkcji, a nie maksymalizacji, dlatego podjęcie decyzji w wyborze funkcji przystosowania może być pewnym wyzwaniem. Żeby maksymalizację zamienić w operację minimalizacji trzeba przekształcić maksymalną wartość funkcji na minimalną. Uzyskiwane jest to pomnożeniem funkcji kosztu i liczby po odjęciu od niej jedynki. Poza tym, nawet maksymalizacja funkcji nie daje pewności, iż konkretna funkcja będzie dawać wyniki nieujemne po wprowadzeniu dowolnych danych wejściowych, a wynik nieujemny jest konieczny, żeby mogła istnieć funkcja przystosowania.

2.5.3. Algorytmy ewolucyjne: Selekcja osobników

Selekcja to proces poszukiwania wewnętrz populacji osobników o pewnych cechach, by można było w algorytmie ewolucyjnym stworzyć kolejne pokolenia. To, czy konkretny

przedstawiciel zostanie wyselekcjonowany zależy od stopnia jego przystosowania. Osobniki o lepszym przystosowaniu mają dużą szansę wyboru do następnego pokolenia, w porównaniu do innych. Tworzenie nowych pokoleń jest zależne od dwóch aspektów: powinna występować różnorodność w populacji i nacisk selekcyjny, czyli poziom wymagań jakie powinien spełnić osobnik, aby brać udział w reprodukcji. Jedno kryterium w jakiś sposób zawsze wpływa na drugie i na odwrót, postawienie wyższych wymagań osobnikom obniży różnorodność genetyczną populacji, co przyspiesza pojawienie się zbieżności, która może spowodować zatrzymanie się programu w ekstremum lokalnym. Z kolei zbyt niski nacisk selekcji zwiększa coraz bardziej losowość poszukiwań rozwiązania. Selekcja ma za zadanie dbać o zrównoważenie omawianych czynników. Spośród dostępnych metod selekcji największą popularnością cieszy się selekcja ruletkowa. Polega na wprowadzeniu zależności prawdopodobieństwa wybrania danego osobnika od tego jak dobrze jest przystosowany. Pojedyncze osobniki mają przydzielone sektory na kole, wielkością odpowiadającą funkcji przystosowania osobnika, która zostaje podzielona przez wartość przystosowania wszystkich osobników razem wziętych. Obracając ruletkę selekcjonuje się przedstawicieli wartych przeniesienia do następnej populacji.

Nacisk selektywny jest sukcesywnie obniżany w kolejnych pokoleniach, za sprawą tego, że rośnie liczba nowych osobników o podobnym przystosowaniu, przez to algorytm ma szansę na zbieżność. Istnieje także sporo innych metod selekcji, np. reprodukcja rangowa, selekcja histogramowa lub wachlarzowa.

Wspomniane metody selekcji, a zwłaszcza selekcja ruletkowa, opcjonalnie można uzupełnić funkcjami dbającymi o to, aby najlepiej dostosowane osobniki były zachowane w przyszłych generacjach. Najpopularniejsza metoda to model elitarystyczny, przenoszący najlepszego przedstawiciela bez poddawania go selekcji. Ma to miejsce, kiedy najlepiej przystosowany osobnik nie zostaje w nowej populacji, wtedy dzięki modelowi elitarystycznemu, najgorszy osobnik w nowej populacji jest usuwany, a jego miejsce zajmuje ten najlepszy, który został przeniesiony z pominięciem selekcji.

2.5.4. Algorytmy ewolucyjne: Krzyżowanie i mutacja

Selekcja jest pierwszym działaniem, zmierzającym w kierunku utworzenia kolejnej generacji populacji. Konieczne są jednak zmiany w chromosomach osobników, by w nowym pokoleniu nie było wyłącznie klonów z poprzedniego. W tym celu używa się operatorów

genetycznych, żeby zmieniły w jakiś sposób wyselekcjonowanych przedstawicieli. Istnieją dwie grupy operatorów genetycznych - jednoargumentowe i wielogargumentowe. Te pierwsze są nazywane mutacjami, co w biologicznym znaczeniu oznacza modyfikacje standardowego kodu DNA osobnika, pod wpływem czynników zewnętrznych albo pomyłkami w procesie replikacji DNA. Wyróżnia się dwa rodzaje mutacji: punktowe, czyli dotyczące jednego genu i mutacje większe, modyfikujące całe fragmenty kodu genetycznego. Jako większe mutacje warto wspomnieć o delecjach - usunięciu części DNA, insercjach - wprowadzenie w jakieś miejsce części kodu pochodzącej z innego obszaru DNA, rearanżacjach - odcinek sekwencji jest odwracany, geny znajdujące się z przodu są przenoszone na tył i na odwrotnie, geny pierwotnie znajdujące się z tyłu na przód. W algorytmach ewolucyjnych naśladowanych jest wiele przykładów mutacji obserwowanych w przyrodzie. Wymienić można mutację brzegową, równomierną, nierównomierną. Te sposoby używane są wobec genów z wartością prawdopodobieństwa $PMut \in [0, 1]$, a wartość tego prawdopodobieństwa to parametr algorytmu.

2.5.5. Algorytmy ewolucyjne: Algorytmy genetyczne

Genetyczne algorytmy, których autorem jest John Holland są najpopularniejsze spośród algorytmów, które powstały na wzór procesów zachodzących w naturze. Holland poprzez algorytmy genetyczne starał się zrozumieć i rozjaśnić, w jaki sposób organizmy dostosowują się do zmian zachodzących w środowisku.

2.5.6. Algorytmy ewolucyjne: Programowanie genetyczne

Programowanie genetyczne bazuje na bardzo ściśle określonych rozwiązańach, które powstały na podstawie specjalnych wersji operatorów genetycznych. Jest to rodzaj algorytmu genetycznego ukierunkowanego na pracę z problemami z jakiegoś konkretnego obszaru. Twórcą tego programowania jest John Koza, który wynalazł je podczas badań nad automatem, który pisałby programy samodzielnie, mając na uwadze podane warunki jakie ten program ma spełniać, aby działał prawidłowo. Podstawowym językiem został LISP, a program napisany w nim przyjmuje postać drzewa. W algorytmach genetycznych popularne było binarne kodowanie, a zostało zmienione na drzewiaste. Algorytmy genetyczne są blisko spokrewnione z programowaniem genetycznym, w związku z tym zasada działania obu jest zbliżona, a polega na tworzeniu kolejnych populacji osobników

reprezentujących rozwiązania problemu i znajdowaniu najlepiej dostosowanych.

2.5.7. Algorytmy ewolucyjne: Strategie ewolucyjne

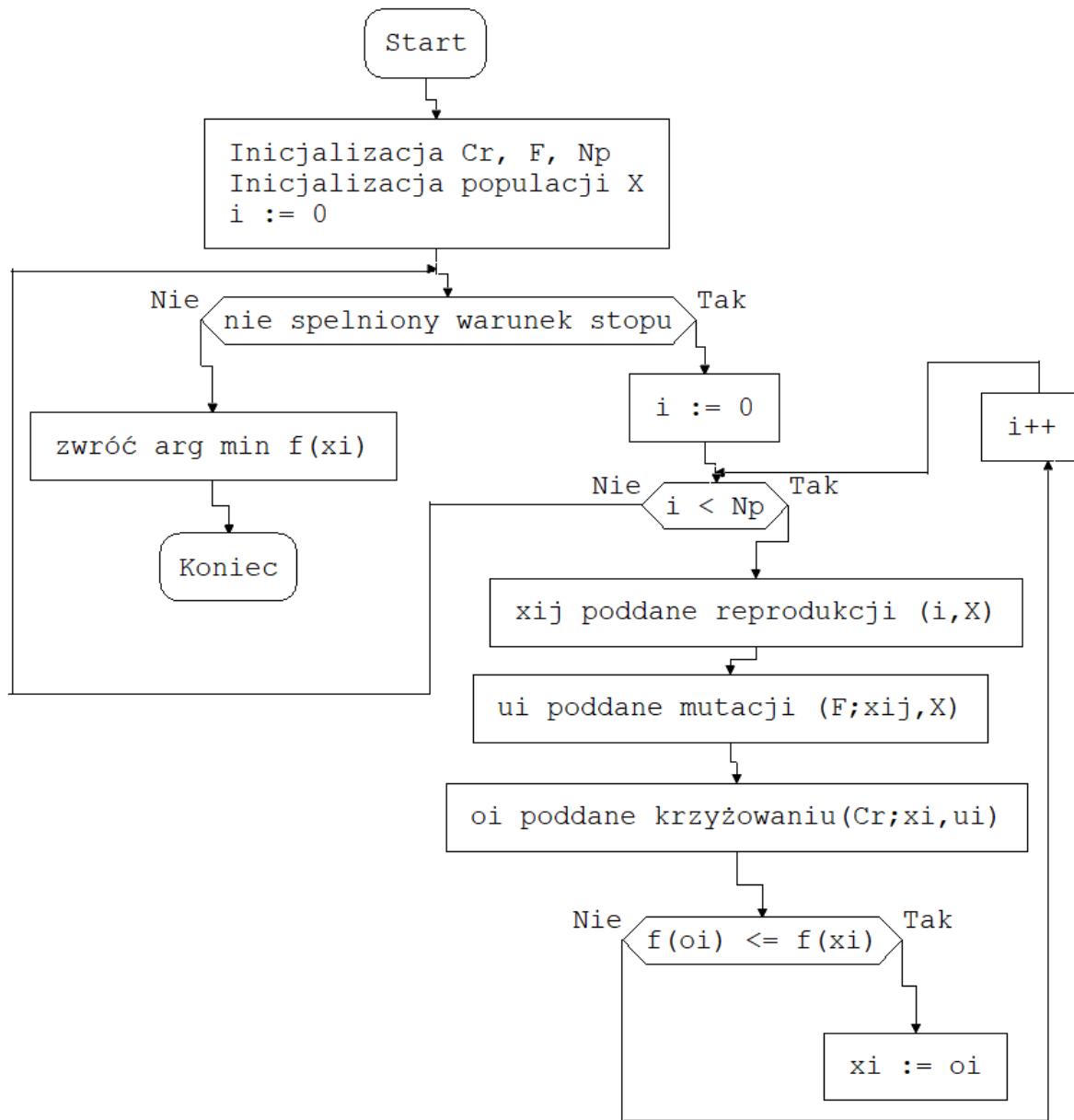
Wynalazcami strategii ewolucyjnych zostali w latach sześćdziesiątych Schwefel, Rechenberg i Bienert. Projektowanie ewolucyjne należy do nauk, w których strategie te zaczęły być używane na samym początku swego istnienia. Zadanie do których je wykorzystało polegało na optymalizacji kształtu rury. Początkowe badania nad strategiami ewolucyjnymi przeprowadzano bez użycia komputerów, algorytm na bazie omawianych strategii pojawił się w 1965 roku, a jego twórcą był Schwefel. W algorytmie tym wszystkie osobniki z kolejnych populacji są poddawane mutacji. Autorzy określili algorytm jako strategię $(1+1)$. Jej działanie polega na tworzeniu jednego przedstawiciela nowej populacji, który powstaje z osobnika starej populacji, potem nowa populacja jest ograniczana, tak żeby ilość osobników z powrotem wynosiła jeden, następnie działanie algorytmu jest powtarzane. Ten rodzaj strategii ma wadę, którą jest słaba odporność na minima lokalne, opracowano więc różne wersje algorytmu, np. $(\mu + \lambda)$ albo $(1 + \lambda)$. W wersji $(1 + \lambda)$, tworzeni są potomkowie w liczbie $\lambda > 1$. W przypadku strategii $(\mu + \lambda)$ w populacji występuje μ osobników, na których wzór wygenerowanych zostaje λ potomków. Za sprawą selekcji tymczasowa populacja $(\mu + \lambda)$ coraz bardziej maleje, a kiedy zmaleje do pojedynczych osobników, kroki strategii są powtarzane od początku. Istnieje jeszcze strategia (μ, λ) , a jej działanie polega na wygenerowaniu λ nowych osobników na bazie μ osobników starszego pokolenia, gdzie λ musi być większa od μ , a dodatkowo selekcja powoduje wybór kolejnej populacji μ osobników spośród λ osobników potomnych. To sprawia, że czas trwania danego osobnika wynosi jedno pokolenie. Kolejni potomkowie są poddawani mutacji, dotyczy to wszystkich osobników w zbiorze λ [37].

2.5.8. Algorytmy ewolucyjne: Ewolucja różnicowa

Ewolucja różnicowa to algorytm, który charakteryzuje się prostotą i skutecznością. Służy do rozwiązywania problemów związanych z optymalizacją globalną ciągłą. Został odkryty i opisany na łamach artykułów naukowych przez Storn'a i Price'a w roku 1995 i od początku jest popularny w środowisku badaczy oraz jako algorytm optymalizujący.

Działanie algorytmu przedstawiono na rys. 2.5.8. Algorytm pracuje na grupie osobników, których ilość określa się jako N_p , osobniki te są wektorami, składającymi się na popula-

cję $X = \{x_1, x_2, \dots, x_{N_p}\}$. Nie odstępując od normy dotyczącej algorytmów ewolucyjnych, populacja po zainicjowaniu ulega działaniu operatorów genetycznych (krzyżowaniu i mutacjom), a potem operacji selekcji [38].



Rysunek 2.5.8: Schemat blokowy algorytmu ewolucji różnicowej

Algorytm ewolucji różnicowej powstał w kilku wersjach, opracowanych również przez Storn'a i Price'a [39]. Są one opisywane w następującej notacji: $DE/X/Y/Z$, gdzie X to wariant reprodukcji, Y ilość wektorów różnicowych, zaś Z typ operatora krzyżowania.

2.6. Porównanie zwykłych metod optymalizacji z optymalizacją ewolucyjną

Metody szukające ekstremów funkcji celu należą do dwóch różnych grup: numerycznych oraz analitycznych. Za analityczne uważane są metody pośrednie i bezpośrednie. Zaś numeryczne to enumeratywne, losowe i przybliżone.

Działanie metod pośrednich polega na szukaniu ekstremów lokalnych funkcji za pomocą rozwiązywania układu równań, powstałego dzięki przerównaniu do zera gradientu funkcji celu. Natomiast metody bezpośrednie wyszukują ekstrema globalne. Za pomocą metod przeglądowych można dowiedzieć się jaką wartość ma funkcja w dowolnym wybranym punkcie. Wyniki w przybliżeniu, wraz z podaniem stopnia dokładności tego przybliżenia można otrzymać przy zastosowaniu metod przybliżonych. Losowe metody posiadają pewną szansę na rozwiązanie zadania, która rośnie razem ze wzrostem skomplikowania metody, należy jednak zaznaczyć, że nie ma pewności, iż to rozwiązanie lub chociaż jego przybliżenie zostanie ostatecznie znalezione. Mimo tego metody losowe cieszą się sporym zainteresowaniem ze względu na znaczące minusy metod przeglądowych i analitycznych.

Algorytmy ewolucyjne (AE) należą do metod losowych, ale nie oznacza to, że działanie AE polega na znajdywaniu po omacku rozwiązania, ani też na przeszukiwaniu, z trzymaniem w pamięci najbardziej optymalnego rozwiązania. Losowość jest wykorzystywana przez algorytmy ewolucyjne, żeby zwiększyć intensywność procesu znajdywania rozwiązania. Jeśli zrobimy porównanie algorytmów ewolucyjnych z algorytmami losowymi, czyli np. symulowane wyżarzanie albo metoda Monte Carlo, to można stwierdzić że metody ewolucyjne zaczynają działanie od rozwiązań początkowych, reprezentowanych przez grupę punktów i szukanie optymalnego rozwiązania jest przeprowadzany z użyciem operacji krzyżowania, mutacji oraz poddawania selekcji. Odnosząc się do metody Monte Carlo dzięki losowaniom można wyznaczyć i zapamiętać najlepszy punkt z populacji. Zaś w odniesieniu do symulowanego wyżarzania punkt v_n , który został utworzony, będzie zapamiętany w przypadku kiedy zawiera doskonalsze rozwiązanie od obecnego. Jeśli tak nie jest to punkt zostaje zapamiętany w zależności od prawdopodobieństwa, na które wpływa różnica funkcji celu poprzedniego punktu (v_c) i nowego (v_n). Oprócz tego wpływ ma również parametr T czyli tzw. temperatura, wg wzoru:

$$\text{if } \text{random}[0; 1] < \exp\left\{\frac{(f(v_c) - f(v_n))}{T}\right\} \text{then } v_c := v_n \quad (24)$$

Rzecz jasna, algorytm podlega wyżarzaniu i przez to wartość T maleje po każdym wykonanym kroku, tak długo aż zostanie osiągnięta bardzo mała wartość T , co będzie oznaczało, że żadne zmiany już nie mogą być przeprowadzone, gdyż układ jest „zamrożony”.

Bardzo ważną kwestią, którą trzeba odpowiednio ustalić w symulowanym wyżarzaniu to właściwe ustalenie wartości spadku temperatury w kolejnych populacjach. Jeżeli temperatura spadać będzie za wolno, to obliczenia zajmą dużo czasu, jeśli natomiast za szybko, to metoda straci na dokładności [38].

Metod optymalizacji jest wiele, co sugeruje, że każda z nich nadaje się do rozwiązywania problemów z dość wąskiej dziedziny. Nie można jednak metodą zoptymalizować wszystkich lub większości zadań. Poszczególne metody różnią się między sobą działaniem, niektóre metody w pewnych sytuacjach mogą wykazywać właściwości, które można by uznać za wady, a w innych zastosowaniach są to zalety. Wydajność i przydatność metody w konkretnej sytuacji jest uzależniona od właściwości funkcji celu, zatem w tej dziedzinie nie ma metod uniwersalnych, nadających się do wykorzystania w każdej sytuacji. Wybór metody powinien być uzależniony od typu zadania optymalizacyjnego jej powierzonego.

Algorytmy ewolucyjne są skuteczniejsze od innych w przypadku, kiedy ilość informacji o rozwiązywanym problemie jest niewielka albo wręcz prawie żadna, w takiej sytuacji można nawet powiedzieć, że algorytm ewolucyjny jako jedyny nadaje się do pracy z tym problemem. Natomiast jeżeli informacje na temat problemu są wystarczające, żeby można było opracować dla niego specjalne rozwiązanie, to będzie ono lepsze od metod ewolucyjnych.

Wyróżnia się cztery cechy używane w algorytmach ewolucyjnych: operacje na populacjach, kodowanie parametrów, operacje losowe i używanie znikomej porcji informacji. Dzięki tym właściwościom, algorytmy ewolucyjne nie są podatne na niektóre pułapki, których zwykłe algorytmy są nieodporne, i w rezultacie czego pozwalają uzyskać korzystniejsze wyniki od zwykłych metod [37].

3. Eksperymenty

3.1. Metody i środowisko badawcze wykorzystane w trakcie eksperymentów

Podczas przeprowadzonych badań, w celu porównania wydajności algorytmów wykorzystano metody napisane w języku *R* [40]. W przypadku większości algorytmów Skorzystano z pakietów w repozytorium *CRAN*, które uzupełniono o mechanizmy umożliwiające kontrolę pracy algorytmu np. pomiar czasu. Jako środowisko programistyczne wykorzystano IDE *RStudio* [41]. Wybrano następujące metody, poszukujące minimum funkcji, w ograniczonym zakresie wartości zmiennych:

1. Algorytmy rojowe:

- Particle Swarm Optimization (pakiet *psoptim* [42]),
- Bat algorithm (pakiet *microbats* [43]).

2. Algorytmy ewolucyjne:

- Genetic Algorithm (pakiet *GA* [44]),
- Differential Evolution (pakiet *DEoptim* [45]).

W związku z dostępnym oprogramowaniem eksperyment ograniczono do 13 funkcji z wieloma ekstremami lokalnymi i jednym ekstremum globalnym. Dla każdej z tych funkcji i dla każdej liczebności zbioru poszukującego rozwiązania wykonano po 50 prób optymalizacji każdym algorytmem($10 \times 5 \times 50 \times 5$). Obliczono błędy średniokwadratowe znalezionych minimów i ich odchylenie standardowe. Na podstawie analizy prób uzyskano wyniki średnie, które umieszczone w tabelach. Dodano także wykresy procesu poszukiwania rozwiązania dla każdej funkcji oraz wykresy znajdowanych minimów w stosunku do iteracji.

3.2. Opis wywołań metod i funkcji rysujących w języku R użytych w badaniach

Zostanie przedstawiony opis poszczególnych argumentów funkcji wywołujących metody optymalizacyjne w R oraz przykład wyniku działania tych metod na przykładzie funkcji *Ackley*. Przedstawiono też fragmenty kodu, odpowiedzialne za generowanie wykresów 2D i 3D.

3.2.1. PSO

Optymalizację przeprowadzono z użyciem pakietu *psoptim*. Oto przykład formuły wywołującej tę metodę:

```
library(pso)
ptm <- proc.time()
psoptim(FUN=ackley.pso, n <- 200, max.loop<- 100, w<- 0.95, c1<-
0.2, c2<- 0.2,
        xmin <- c(-32, -32), xmax<- c(32, 32), vmax<- c(4, 4),
        seed=rnorm(1, min=0, max=1000), anim=FALSE)
czas<- proc.time() - ptm
print(czas[3])
```

gdzie: FUN-nazwa funkcji, n-liczba częstek w roju, max.loop-maksymalna liczba iteracji, w-współczynnik bezwładności, c2- współczynnik własnego „zaufania”, c1-współczynnik „zaufania” do roju, xmin-wektor określający dolne ograniczenia wartości zmiennych, xmax-wektor określający górne ograniczenia wartości zmiennych, vmax-wektor ograniczeń prędkości w każdym kierunku, seed-liczba określająca tzw. ziarno dla generatora liczb pseudolosowych, anim-wartość logiczna informująca o tym czy wykresy 2D mają być generowane, ptm,czas-wartości odpowiadające za pomiar czasu wykonania programu

Wynik działania metody na funkcji Ackley:

```
$sol
      x1          x2
[1,] 0.0009598709 0.0006415018
$val
[1] -0.003300919
$loop
```

```
[1] 76
```

```
elapsed
```

```
0.26
```

gdzie: sol-znalezione wartości niewiadomych x1,x2, val-wartość minimum funkcji, loop-ilość przebiegów, elapsed-czas wykonania

3.2.2. BAT

Optymalizację przeprowadzono z użyciem pakietu *microbats*. Przykład formuły wywołującej:

```
library(microbats)
set.seed(runif(1,min=0,max=1000))
ptm <- proc.time()
fit <- bat_optim(D = 2, NP = 200, N_Gen = 100, A = 0.5, r = 0.5,
                  Qmin = 0, Qmax = 2, Lower = -32, Upper = 32,
                  FUN = ackley.bat, anim=FALSE)
czas<- proc.time() - ptm
fit
print(czas[3])
```

gdzie: set.seed-liczba określająca tzw. ziarno dla generatora liczb pseudolosowych, D-ilość niewiadomych funkcji, NP-liczba nietoperzy, N_GEN-maksymalna liczba iteracji, A-„głośność” nietoperzy, r-szybkość impulsów, Qmin-minimalna częstotliwość, Qmax-maksymalna częstotliwość, Lower-wektor określający dolne ograniczenia wartości zmiennych,Upper-wektor określający górne ograniczenia wartości zmiennych, anim-wartość logiczna informująca o tym czy wykresy 2D mają być generowane, ptm,czas-wartości odpowiadające za pomiar czasu wykonania programu

Wynik działania metody na funkcji Ackley:

```
$best_solution
```

```
[ ,1] [ ,2]
```

```
[1,] 0.9521676 2.17323e-06
```

```
$min_fitness
```

```
[1] 2.579928
```

```
$iter
```

```
[1] 57
```

```
elapsed
```

```
0.18
```

gdzie: best_solution-znalezione wartości niewiadomych x1,x2, min_fitness-wartość minimum funkcji, iter-ilość przebiegów, elapsed-czas wykonania

3.2.3. GA

Przykład bieżący jest oparty o metodę z pakietu *GA*. Formuła wywołująca działanie metody:

```
library(GA)
```

```
ptm <- proc.time()
```

```
GA <- ga(type = "real-valued",
```

```
    fitness = function(x) -ackley.ga(x[1], x[2]),
```

```
    min = c(-32, -32), max = c(32, 32),
```

```
    popSize = 200, maxiter = 100, run = 50,
```

```
    pmutation = ga_pmutation, anim = FALSE, maxFitness = 0)
```

```
czas<-proc.time() - ptm
```

```
summary(GA)
```

```
print(czas[3])
```

gdzie: type-typ algorytmu genetycznego, który będzie uruchamiany w zależności od decyzji, fitness-funkcja do optymalizacji(ponieważ metoda szuka domyślnie maksimum funkcji to występuje odejmowanie), min,max-wektory dolnego i górnego ograniczenia wartości szukanych niewiadomych, popSize-wielkość populacji, maxiter-maks. liczba iteracji, run-minimalna ilość iteracji przed zatrzymaniem, pmutation-prawdopodobieństwo mutacji na parach chromosomów, anim-wartość logiczna informująca o tym czy wykresy 2D mają być generowane, ptm,czas-wartości odpowiadające za pomiar czasu wykonania programu

Wynik działania metody na funkcji Ackley:

```
GA results:
```

```
Iterations = 100
```

```
Fitness function value = -3.967519e-06
```

```
Solution =
```

x1	x2
[1 ,] 1.153305e-06	-7.984272e-07
elapsed	
0.58	

gdzie: Fitness function value-znaleziona wartość minimum funkcji, Solution-znalezione wartości niewiadomych x1,x2, Iterations-ilość przebiegów, elapsed-czas wykonania

3.2.4. DE

Przykład dotyczy metod z pakietu *DEoptim*. Kod wywołujący działanie metody jest następujący:

```
library(DEoptim)
lower <- c(-32, -32)
upper <- -lower
set.seed(runif(1,min=0,max=1000))

ptm <- proc.time()
outDEoptim <- DEoptim(ackley.de, lower, upper,
                      DEoptim.control(itermax = 100,
                                     storepopfrom = 1,
                                     storepopfreq = 1, NP
                                     =200,trace=FALSE, anim=
                                     FALSE) )
summary(outDEoptim)
czas<- proc.time() - ptm
print(czas [3])
```

gdzie: lower,upper-wektory dolnego i górnego ograniczenia wartości szukanych niewiadomych, set.seed-liczba określająca tzw. ziarno dla generatora liczb pseudolosowych, ackley.de-nazwa funkcji do optymalizacji, NP-wielkość populacji, itermax-maks. liczba iteracji, storepopfrom-liczba określająca, od której iteracji ma być zachowana populacja w pamięci, storepopfreq-z jaką częstotliwością mają być zachowywane populacje, anim-wartość logiczna informująca o tym czy wykresy 2D mają być generowane, ptm,czas-wartości odpowiadające za pomiar czasu wykonania programu

Wynik działania metody na funkcji Ackley:

```
***** summary of DEoptim object *****  
best member      : 0 0  
best value       : 0  
after           : 100 generations  
fn evaluated    : 20200 times  
*****  
elapsed  
0.14
```

gdzie: best member-znaleziona wartość minimum funkcji, best value-znalezione wartości niewiadomych x_1, x_2 , after-ilość pokoleń po których znaleziono rozwiązanie, elapsed-czas wykonania

3.2.5. Wykresy 2D z pozycjami osobników

W przypadku PSO wykresy 2D zostały wygenerowane za pomocą funkcji *contour* i *points*. *contour* odpowiada za wyświetlanie tła, przedstawiającego zarys optymalizowanej funkcji rzutowanej z góry, zaś *points* rysuje pozycje osobników:

```
contour(x_image[,1], x_image[,2], z, nlevels=20, xlab="x1", ylab="x2", col="darkgray", main=paste(loop))  
points(x, xlim=c(xmin[1], xmax[1]), ylim=c(xmin[2], xmax[2]), pch=21, bg="cadetblue")
```

gdzie: $x_image[1]$, $x_image[2]$ - macierze zawierające informacje na temat rzutowanego obrazu

W pozostałych metodach wykorzystano funkcję *plot* do rysowania pozycji członków populacji:

```
plot(Sol, main=t, type = "p", pch=19, col="darkred", cex = 2, xlab="x1", ylab="x2", xlim=range(Lower:Upper), ylim=range(Lower:Upper))
```

gdzie: Sol(różne nazwy w zależności od metody) - macierz zawierająca współrzędne członków populacji

Funkcje tworzenia wykresów 2D znajdują się wewnętrz metod optymalizacyjnych i były wywoływanie poprzez wartość logiczną *anim = TRUE*.

3.2.6. Wykresy 2D zależności znalezioneego minimum od iteracji

Wykresy 2D przedstawiające wartość minimum w z biegiem iteracji są generowane dzięki funkcji *plot* przedstawionej w poniższym fragmencie kodu:

```
plot(wynik, type = "o", pch=19, col="darkblue", xlab="Iteracje",
      ylab="Znalezione_minimum_funkcji")
```

gdzie: wynik(różne nazwy w zależności od metody) - wektor zawierający wyniki najlepszego znalezioneego minimum w kolejnych iteracjach

3.2.7. Wykresy 3D funkcji

Wykresy 3D wygenerowano za pomocą funkcji *persp* należącej do pakietu *GA*. Poniżej zaprezentowano kod wywołujący tę funkcję rysującą:

```
x1 <- x2 <- seq(xmin, xmax, by = 0.1)
f <- outer(x1, x2, nazwa)
persp3D(x1, x2, f, theta = 50, phi = 20)
```

gdzie: x1,x2 - wektory liczb od minimum przedziału do maksimum, skok co 0.1, xmin,xmax-minimalna wartość ograniczeń wartości zmiennych oraz wartość maksymalna, nazwa-nazwa funkcji

3.3. Opis funkcji użytych w eksperymencie

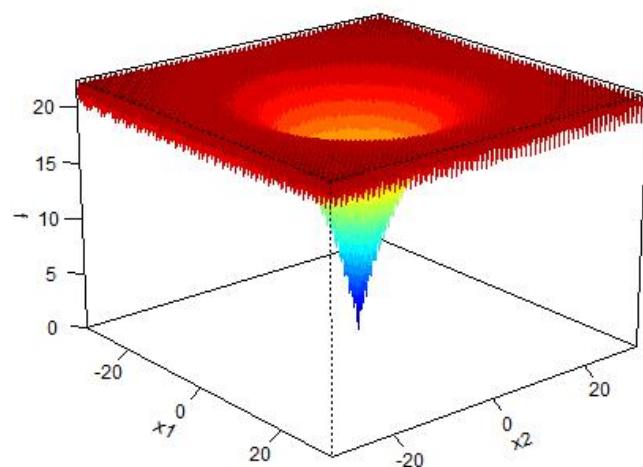
Podczas badań optymalizowano 13 funkcji o dwu niewiadomych [46, 47] szukając ich minimum globalnego w określonym przedziale wartości zmiennych. W tabeli 3.3.1 umieszczone nazwy wykorzystanych funkcji, ograniczenia wartości zmiennych, poszukiwane wartości niewiadomych i poszukiwana wartość minimum globalnego.

W kolejnych podsekcjach przedstawiono poszczególne funkcje, ich wykresy 3D (rysunki 3.3.9 - 3.3.21), postać matematyczną oraz kod źródłowy w języku R.

Tabela 3.3.1: Lista funkcji użytych do eksperymentu

Lp.	Nazwa funkcji	Zakres wartości x1 i x2	Poszukiwane wartości x1 i x2	Poszukiwana wartość f(x1,x2)
1	Ackley	[-32, 32]	(0, 0)	0
2	Beale	[-4.5, 4.5]	(3, 0.5)	0
3	Goldstein	[-2, 2]	(0, -1)	3
4	Bartels Conn	[-500, 500]	(0, 0)	1
5	Leon	[-1.2, 1.2]	(1, 1)	0
6	Eggholder	[-512, 512]	(512, 404)	-959
7	Venter	[-50, 50]	(0, 0)	-400
8	Matyas	[-10, 10]	(0, 0)	0
9	Zirilli	[-10, 10]	(-1.04 ,0)	-0,35
10	Easom	[-100, 100]	(3.14, 3.14)	-1
11	Rastrigin	[-5.12, 5.12]	(0, 0)	0
12	Levy N.13	[-10, 10]	(1, 1)	0
13	Drop Wave	[-5.12, 5.12]	(0, 0)	-1

3.3.1. Ackley



Rysunek 3.3.9: Wykres 3D funkcji Ackley

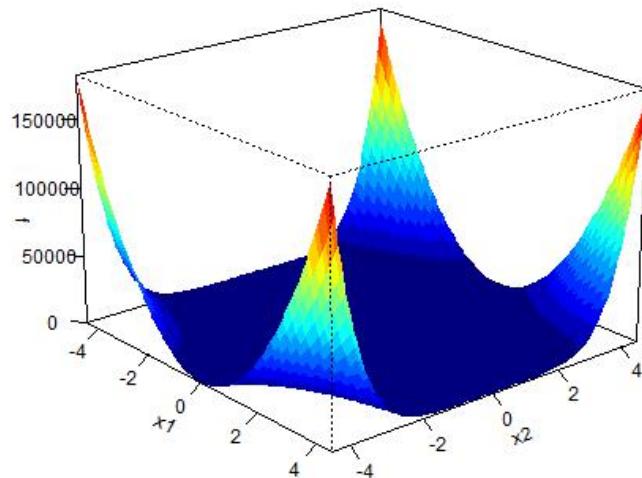
Postać matematyczna funkcji:

$$f(x) = -20\exp(-0.2 \cdot \sqrt{0.5 \cdot (x_1^2 + x_2^2)}) - \exp(0.5 \cdot (\cos(2\pi x_1) + \cos(2\pi x_2))) + e + 20 \quad (25)$$

Kod źródłowy:

```
ackley = function(x1, x2)
-20*exp(-0.2*sqrt(0.5*(x1^2+x2^2)))-exp(0.5*(cos(2*pi*x1)+cos(2*pi*x2)))+ exp(1) + 20
```

3.3.2. Beale



Rysunek 3.3.10: Wykres 3D funkcji Beale

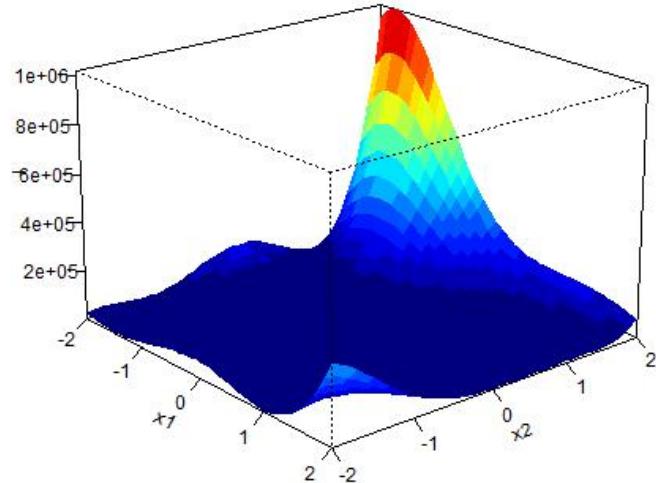
Postać matematyczna funkcji:

$$f(x) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2 \quad (26)$$

Kod źródłowy:

```
beale = function(x1, x2)
(1.5-x1+x1*x2)^2 + (2.25-x1+x1*x2^2)^2 +(2.625-x1+x1*x2^3)^2
```

3.3.3. Goldstein



Rysunek 3.3.11: Wykres 3D funkcji Goldstein

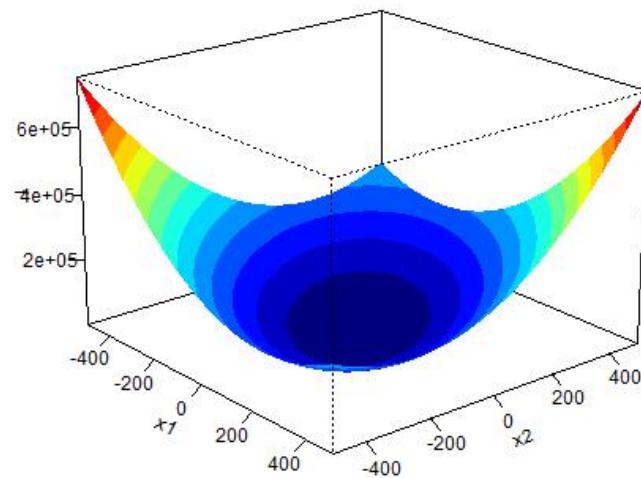
Postać matematyczna funkcji:

$$f(x) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \\ (30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)) \quad (27)$$

Kod źródłowy:

```
goldstein = function(x1, x2)
(1+(x1+x2+1)^2 * (19-14*x1+3*x1^2 - 14*x2+6*x1*x2+3*x2^2)) *
(30+(2*x1-3*x2)^2 * (18-32*x1+12*x1^2 + 48*x2-36*x1*x2+27*
x2^2))
```

3.3.4. Bartels Conn



Rysunek 3.3.12: Wykres 3D funkcji Bartels Conn

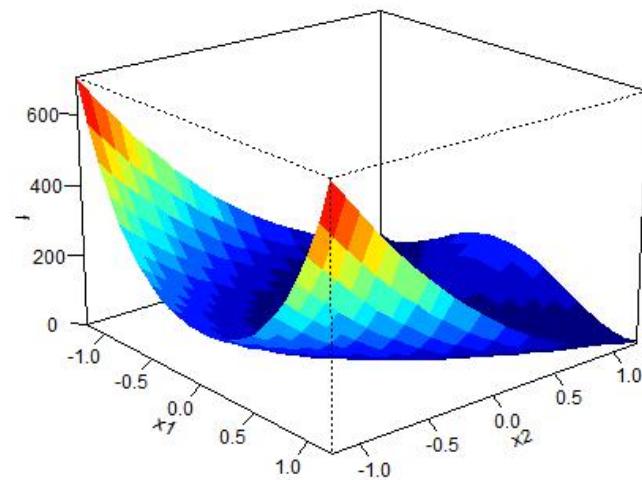
Postać matematyczna funkcji:

$$f(x) = |x_1^2 + x_2^2 + x_1 x_2| + |\sin(x_1)| + |\cos(x_2)| \quad (28)$$

Kod źródłowy:

```
bartels.conn = function(x1, x2)
  abs(x1^2 + x2^2 + x1*x2) + abs(sin(x1)) + abs(cos(x2))
```

3.3.5. Leon



Rysunek 3.3.13: Wykres 3D funkcji Leon

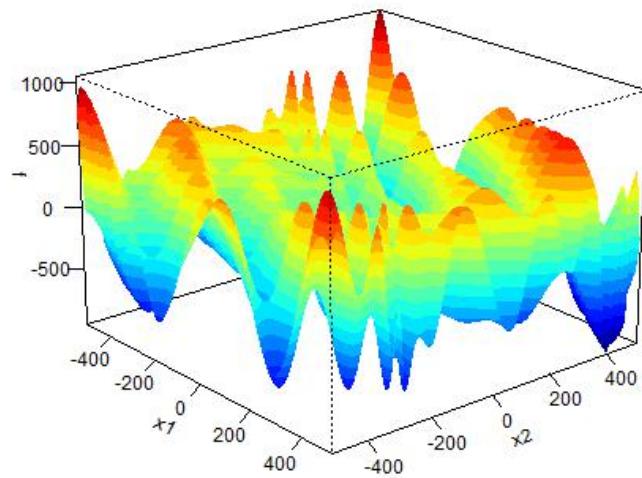
Postać matematyczna funkcji:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad (29)$$

Kod źródłowy:

```
leon = function(x1, x2)
  100*(x2-x1^2)^2 + (1-x1)^2
```

3.3.6. Eggholder



Rysunek 3.3.14: Wykres 3D funkcji Eggholder

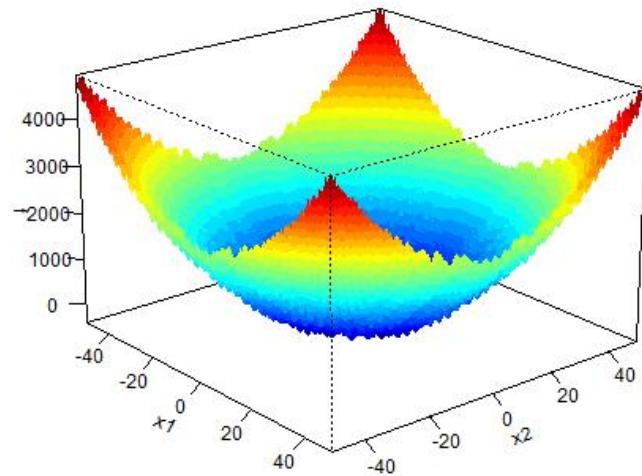
Postać matematyczna funkcji:

$$f(x) = -(x_2 + 47)\sin\sqrt{\left|\frac{x_1}{2} + (x_2 + 47)\right|} - x_1\sin\sqrt{\left|x_1 - (x_2 + 47)\right|} \quad (30)$$

Kod źródłowy:

```
eggholder = function(x1, x2)
  -(x2+47)*sin(sqrt(abs(x2+x1/2+47))) - x1*sin(sqrt(abs(x1-(x2+47))))
```

3.3.7. Venter



Rysunek 3.3.15: Wykres 3D funkcji Venter

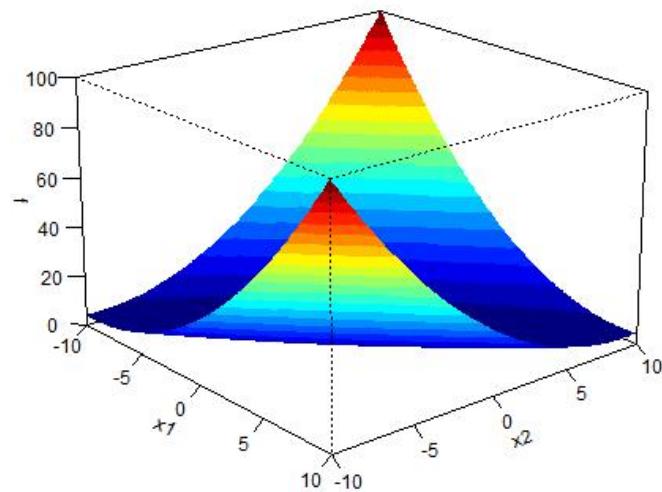
Postać matematyczna funkcji:

$$f(x) = x_1^2 - 100\cos(x_1)^2 - 100\cos(x_1^2/30) + x_2^2 - 100\cos(x_2)^2 - 100\cos(x_2^2/30) \quad (31)$$

Kod źródłowy:

```
venter = function(x1, x2)
  x1^2 - 100*cos(x1)^2 - 100*cos(x1^2/30) + x2^2 - 100*cos(x2)^2 - 100*
  cos(x2^2/30)
```

3.3.8. Matyas



Rysunek 3.3.16: Wykres 3D funkcji Matyas

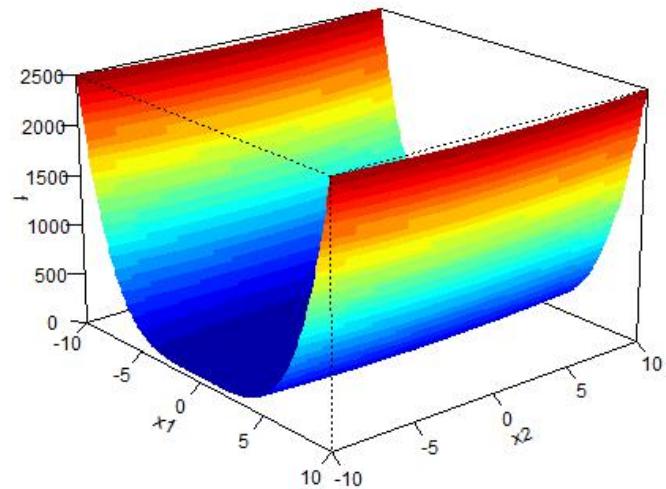
Postać matematyczna funkcji:

$$f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2 \quad (32)$$

Kod źródłowy:

```
matyas = function(x1, x2)
  0.26 * (x1^2 + x2^2) - 0.48 * x1 * x2
```

3.3.9. Zirilli



Rysunek 3.3.17: Wykres 3D funkcji Zirilli

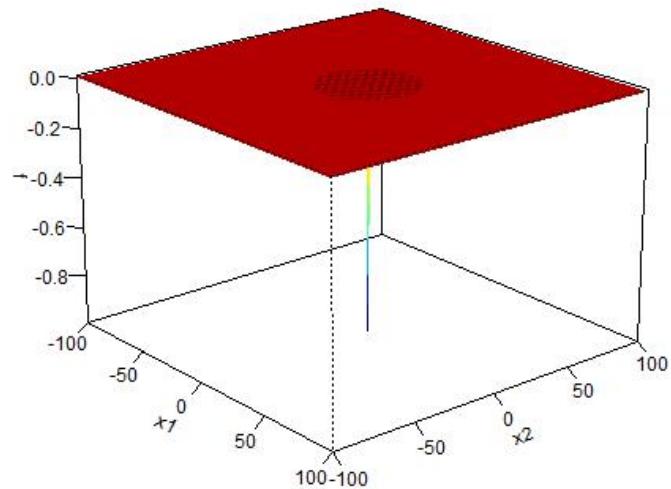
Postać matematyczna funkcji:

$$f(x) = 0.25x_1^4 - 0.5x_1^2 + 0.1x_1 + 0.5x_2^2 \quad (33)$$

Kod źródłowy:

```
zirilli = function(x1,x2)
  0.25*x1^4 - 0.5*x1^2 + 0.1*x1 + 0.5*x2^2
```

3.3.10. Easom



Rysunek 3.3.18: Wykres 3D funkcji Easom

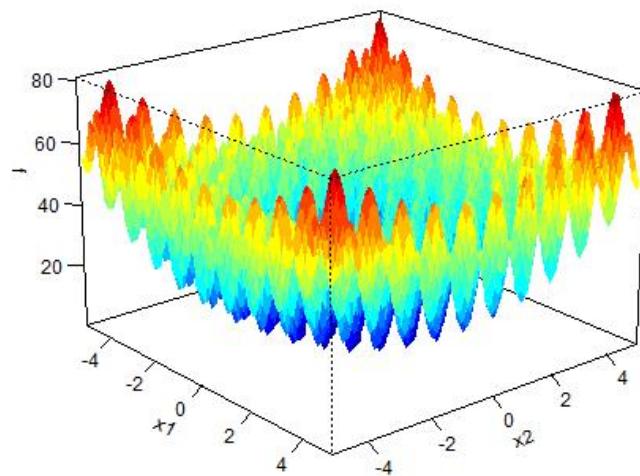
Postać matematyczna funkcji:

$$f(x) = -\cos(x_1)\cos(x_2)\exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2)) \quad (34)$$

Kod źródłowy:

```
easom = function(x1, x2)
-cos(x1)*cos(x2)*exp(-((x1-pi)^2 + (x2-pi)^2))
```

3.3.11. Rastrigin



Rysunek 3.3.19: Wykres 3D funkcji Rastrigin

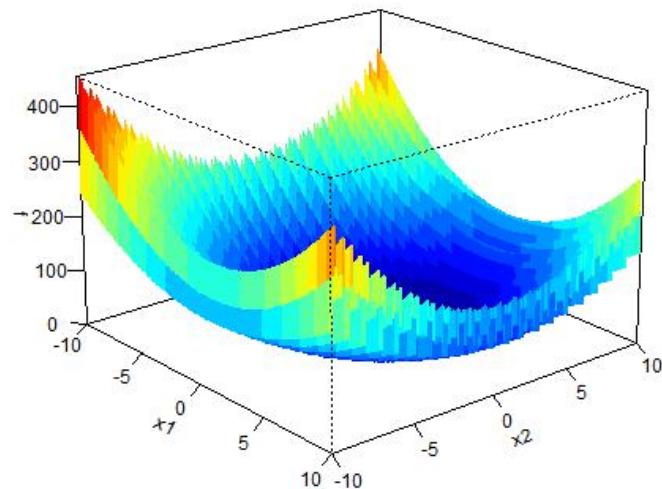
Postać matematyczna funkcji:

$$f(x) = -\cos(x_1)\cos(x_2)\exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2)) \quad (35)$$

Kod źródłowy:

```
rastrigin <- function(x1, x2)
20 + x1^2 + x2^2 - 10*(cos(2*pi*x1) + cos(2*pi*x2))
```

3.3.12. Levy N.13



Rysunek 3.3.20: Wykres 3D funkcji Levy N.13

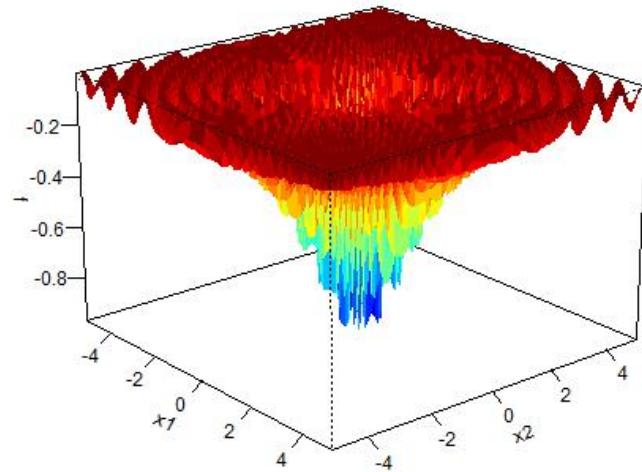
Postać matematyczna funkcji:

$$f(x) = \sin^2(3\pi x_1) + (x_1 - 1)^2(1 + \sin^2(3\pi x_2)) + (x_2 - 1)^2(1 + \sin^2(2\pi x_2)) \quad (36)$$

Kod źródłowy:

```
levin13 = function(x1, x2)
  sin(3*pi*x1)^2 + (x1 - 1)^2 * (1 + sin(3*pi*x2)^2) + (x2 - 1)^2 * (1 + sin(2*pi*x2)^2)
```

3.3.13. Drop Wave



Rysunek 3.3.21: Wykres 3D funkcji Drop Wave

Postać matematyczna funkcji:

$$f(x) = -\frac{1 + \cos(12\sqrt{x_1^2 + x_2^2})}{0.5(x_1^2 + x_2^2) + 2} \quad (37)$$

Kod źródłowy:

```
drop_wave = function(x1, x2)
(-(1+cos(12*sqrt(x1^2+x2^2)))/(0.5*(x1^2+x2^2)+2))
```

3.4. Optymalizacja rojem częstek

Eksperyment przeprowadzono z wykorzystaniem pakietu *psoptim*. Wartości parametrów wywołania metody optymalizacyjnej były następujące:

- nazwa funkcji
- liczba częstek w roju: [20,40,70,100,200]
- maksymalna liczba iteracji: 100
- liczba powtórzeń o identycznym wyniku zatrzymująca pracę programu: 20
- współczynnik bezwładności: 0,95
- współczynnik własnego „zaufania”: 0,2
- współczynnik „zaufania” do roju: 0,2
- wektor określający dolne ograniczenia wartości zmiennych
- wektor określający górne ograniczenia wartości zmiennych
- wektor ograniczeń prędkości w każdym kierunku: (4,4)
- liczba określająca tzw. ziarno dla generatora liczb pseudolosowych argument stosowany w celu uzyskania powtarzalności otrzymywanych wyników: rand(0:1000)

Na podstawie 50 prób dla każdej funkcji i liczebności zbioru wygenerowano wyniki średnie, które zamieszczono w tabelach 3.4.2 - 3.4.6.

Tabela 3.4.2: Średnie wyniki optymalizacji PSO

Ackley								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-0,021	0,025	0,454	0,27	0,25	63,16	0,0142
2	40	-0,011	-0,007	0,572	0,72	0,64	52,16	0,0236
3	70	-0,002	-0,011	0,260	0,16	0,31	63,32	0,054
4	100	-0,001	-0,014	0,236	0,10	0,22	52,9	0,0694
5	200	0,001	-0,004	0,108	0,03	0,12	60	0,1716
Wartości szukane								

Beale								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	3,160	0,524	0,022	0,00	0,03	45,02	0,0132
2	40	3,016	0,503	0,002	0,00	0,00	51,46	0,0202
3	70	3,006	0,500	0,002	0,00	0,00	56,8	0,048
4	100	3,004	0,503	0,001	0,00	0,00	61,68	0,0968
5	200	3,001	0,500	0,000	0,00	0,00	56,16	0,1564
Wartości szukane								

Goldstein								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,007	-0,995	3,233	0,22	0,41	59,1	0,018
2	40	0,005	-0,998	3,164	0,25	0,48	54,5	0,0316
3	70	0,000	-0,999	3,095	0,07	0,26	57,78	0,059
4	100	0,001	-1,001	3,034	0,01	0,09	55,78	0,0722
5	200	0,001	-1,000	3,005	0,00	0,01	70,72	0,2138
Wartości szukane								

Tabela 3.4.3: cd. Średnie wyniki optymalizacji PSO

Bartels Conn								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-7,161	-0,073	849,971	2253397,61	1250,57	98,06	0,0254
2	40	-0,596	0,547	24,818	8567,17	90,35	84,66	0,038
3	70	-0,582	0,723	8,476	868,32	28,79	87,84	0,0936
4	100	-0,060	0,147	1,318	0,64	0,74	78,48	0,1362
5	200	-0,172	0,303	2,602	106,55	10,30	82	0,2252
		Wartości szukane	0	0	1			

Leon								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,951	0,915	0,018	0,00	0,03	52,34	0,0162
2	40	1,014	1,029	0,002	0,00	0,00	50,78	0,0224
3	70	0,999	1,000	0,003	0,00	0,01	54,48	0,0402
4	100	1,000	1,002	0,002	0,00	0,00	56,86	0,072
5	200	0,999	0,999	0,000	0,00	0,00	58,88	0,1712
		Wartości szukane	1	1	0			

Eggholder								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	353,201	332,005	-896,709	11860,21	90,24	80,96	0,0208
2	40	442,033	406,984	-935,571	2506,96	44,70	75,12	0,0344
3	70	478,452	413,605	-948,901	665,04	23,97	62	0,063
4	100	502,098	411,507	-956,303	61,08	7,41	84,96	0,1382
5	200	504,944	411,175	-958,521	3,07	1,70	67,02	0,1872
		Wartości szukane	512	404	-959			

Tabela 3.4.4: cd. Średnie wyniki optymalizacji PSO

Venter								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-0,475	-1,651	-387,826	196,69	7,03	50,78	0,0132
2	40	0,105	-0,058	-396,711	47,48	6,12	61,56	0,0286
3	70	-0,061	0,042	-398,554	14,68	3,58	62,18	0,0682
4	100	0,005	-0,003	-399,666	0,49	0,62	63,98	0,0986
5	200	-0,006	-0,002	-399,813	0,21	0,42	63,56	0,1984
Wartości szukane								
0								-400

Matyas								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-0,066	-0,063	0,001	0,00	0,00	57,98	0,014
2	40	0,003	0,003	0,000	0,00	0,00	51,68	0,0224
3	70	0,016	0,011	0,000	0,00	0,00	43,52	0,0322
4	100	-0,014	-0,008	0,000	0,00	0,00	50,34	0,0584
5	200	0,001	0,001	0,000	0,00	0,00	53,68	0,1466
Wartości szukane								
0								0

Zirilli								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-1,045	0,022	-0,352	0,00	0,00	69,96	0,0182
2	40	-1,049	0,002	-0,351	0,00	0,00	54,18	0,023
3	70	-1,047	-0,002	-0,351	0,00	0,00	52,58	0,0518
4	100	-1,046	0,004	-0,352	0,00	0,00	58,3	0,0818
5	200	-1,046	-0,003	-0,352	0,00	0,00	53,38	0,1602
Wartości szukane								-0,35
0								

Tabela 3.4.5: cd. Średnie wyniki optymalizacji PSO

Drop Wave								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,015	-0,059	-0,949	0,00	0,02	40,14	0,011
2	40	0,006	-0,035	-0,970	0,00	0,03	55,46	0,0232
3	70	-0,032	0,009	-0,987	0,00	0,02	65,04	0,0582
4	100	0,003	-0,013	-0,991	0,00	0,02	63,6	0,0844
5	200	0,003	0,010	-0,996	0,00	0,01	68,44	0,1842
	Wartości szukane	0	0	-1				

Levy N.13								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	1,011	0,980	0,059	0,01	0,07	53,06	0,0144
2	40	1,011	1,001	0,052	0,01	0,06	50,38	0,0218
3	70	1,001	1,007	0,013	0,00	0,01	46,22	0,0338
4	100	1,017	1,031	0,015	0,00	0,03	49,76	0,0598
5	200	1,000	0,998	0,001	0,00	0,00	62,46	0,1802
	Wartości szukane	1	1	0				

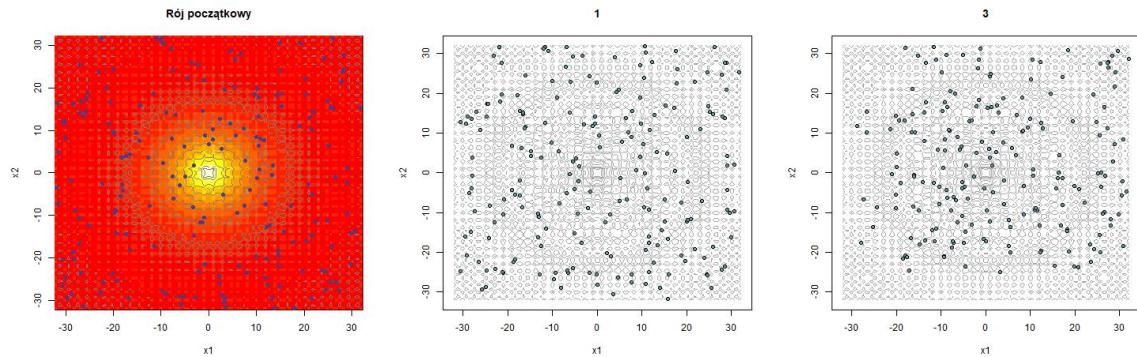
Rastrigin								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-0,018	0,099	0,481	0,62	0,63	63,76	0,0164
2	40	0,023	-0,061	0,427	0,61	0,66	55,44	0,0262
3	70	0,017	-0,021	0,104	0,06	0,23	61,5	0,0486
4	100	0,061	-0,041	0,119	0,10	0,30	60,96	0,0794
5	200	-0,001	-0,021	0,040	0,03	0,16	62,62	0,1698
	Wartości szukane	0	0	0				

Tabela 3.4.6: cd. Średnie wyniki optymalizacji PSO

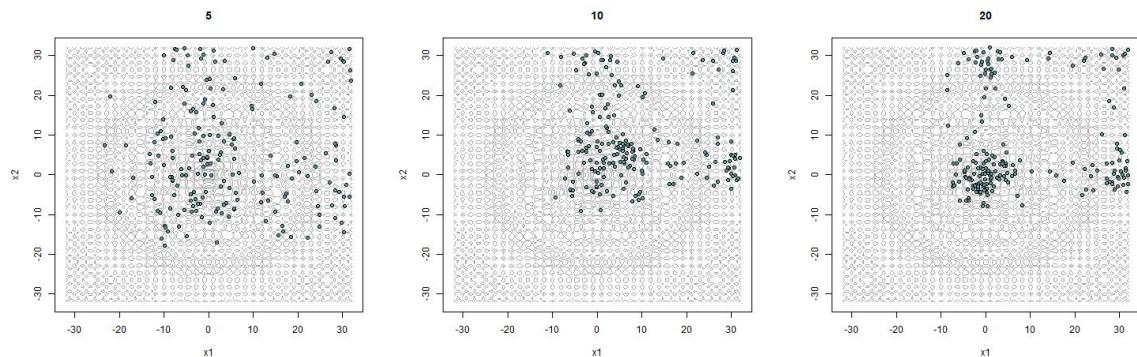
Eason						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe
1	20	11,196	26,280	-0,620	0,37	0,48
2	40	3,135	3,135	-0,991	0,00	0,01
3	70	3,141	3,148	-0,982	0,00	0,06
4	100	3,138	3,134	-0,988	0,00	0,03
5	200	3,139	3,138	-0,995	0,00	0,01
		Wartości szukane	3,14	3,14	-1	

Na rysunkach znajdujących się w następnych podsekcjach przedstawiono wizualizację zainicjowania roju PSO o liczbie 200 i poszukiwania rozwiązań przez ten rój dla funkcji. Na rysunkach 3.4.22 - 3.4.65 przedstawiono początkowe pozycje osobników oraz ich pozycje w dalszych iteracjach.

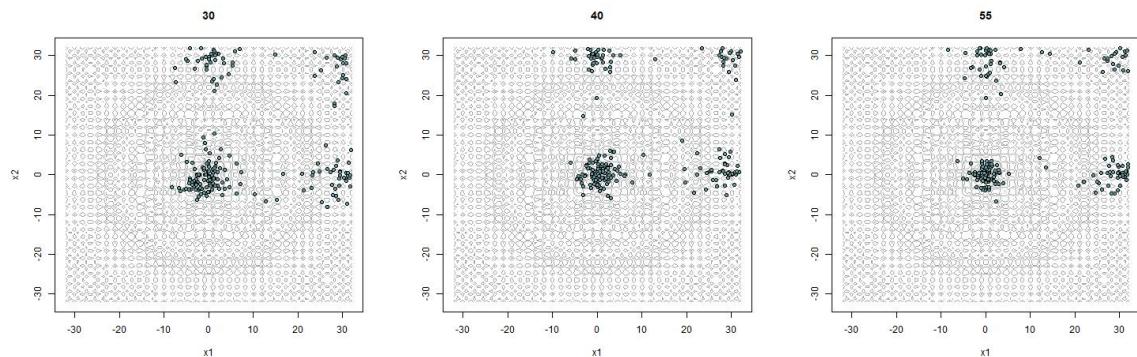
3.4.1. Ackley



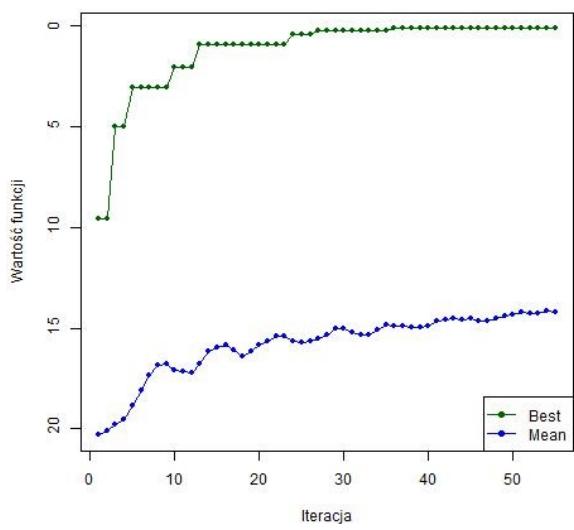
Rysunek 3.4.22: Iteracja 0, 1 i 3



Rysunek 3.4.23: Iteracja 5, 10 i 20

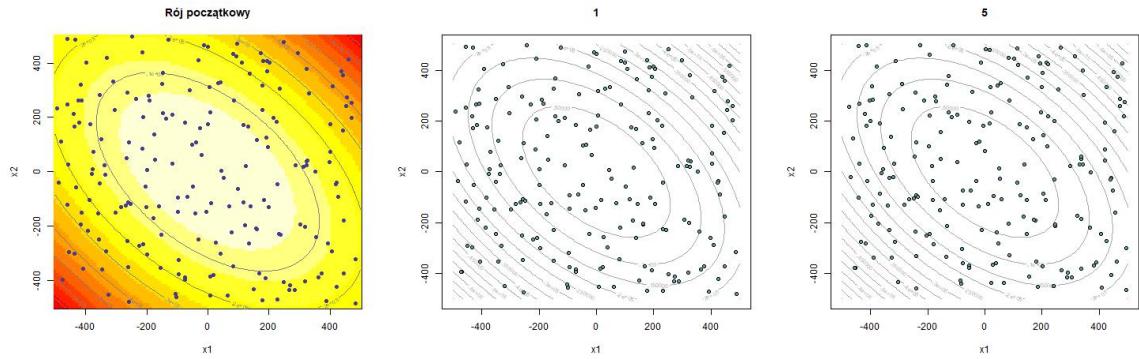


Rysunek 3.4.24: Iteracja 30, 40 i 55

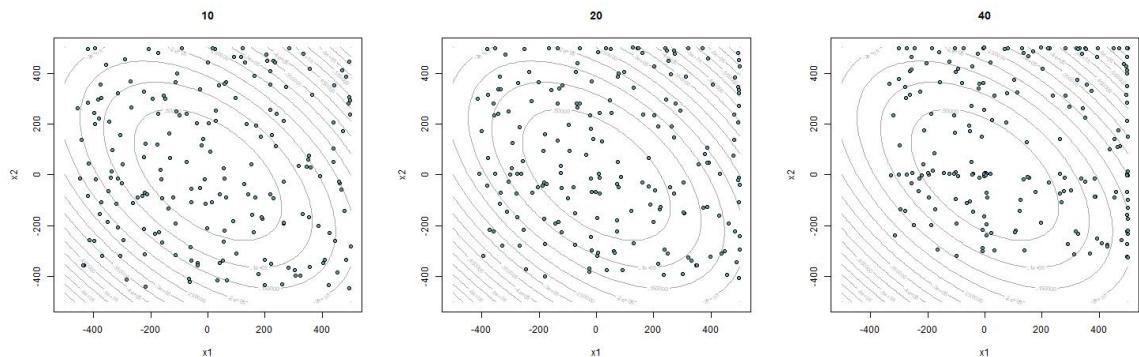


Rysunek 3.4.25: Wartości funkcji z biegiem iteracji

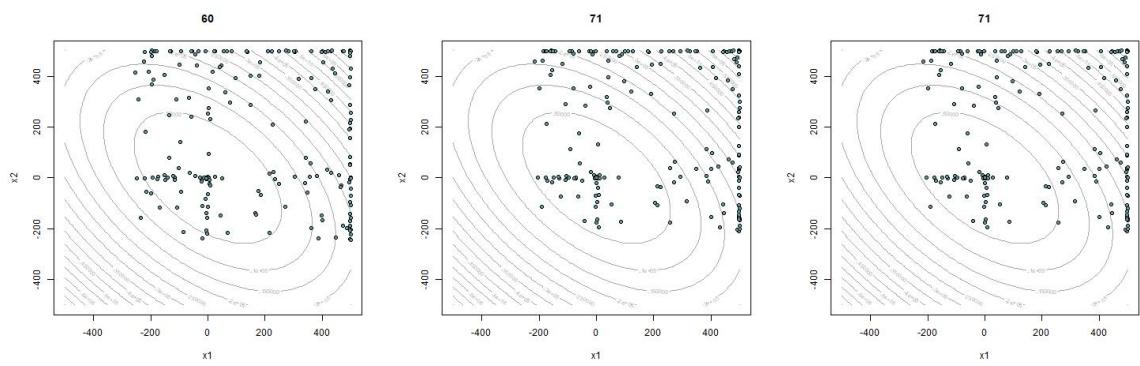
3.4.2. Bartels



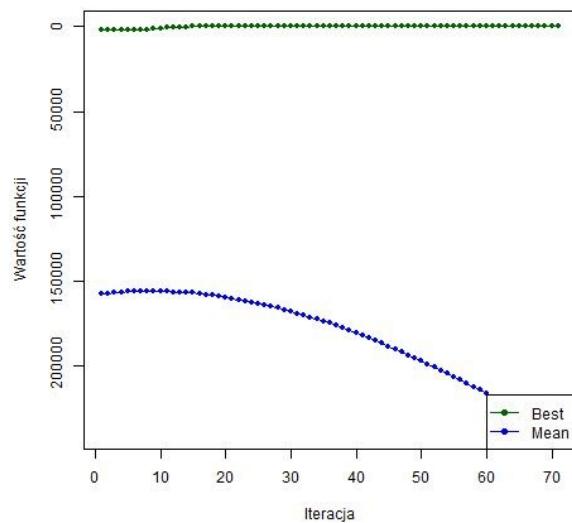
Rysunek 3.4.26: Iteracja 0, 1 i 5



Rysunek 3.4.27: Iteracja 10, 20 i 40

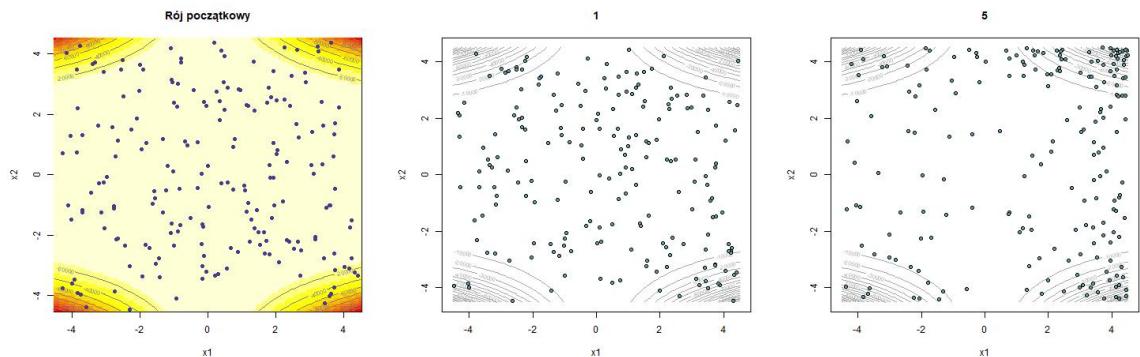


Rysunek 3.4.28: Iteracja 60, 71 i 71

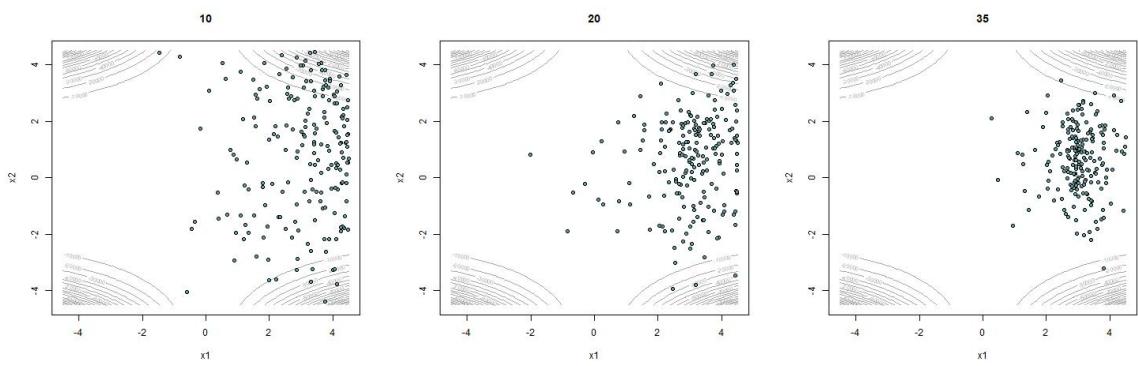


Rysunek 3.4.29: Wartości funkcji z biegiem iteracji

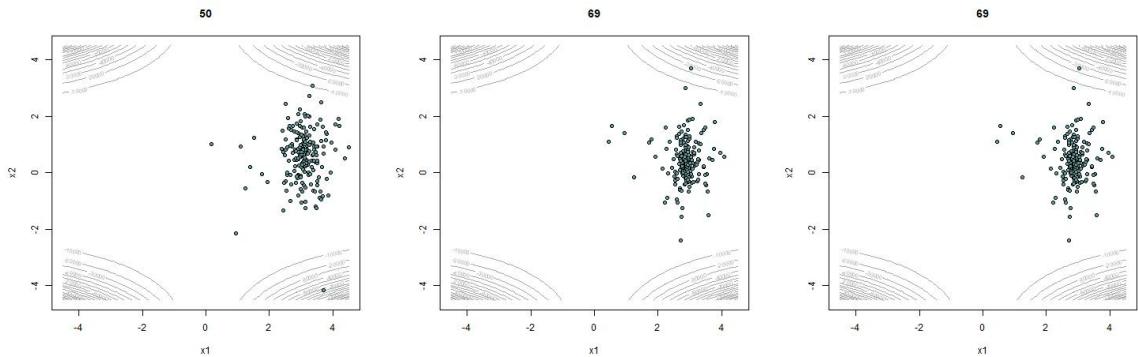
3.4.3. Beale



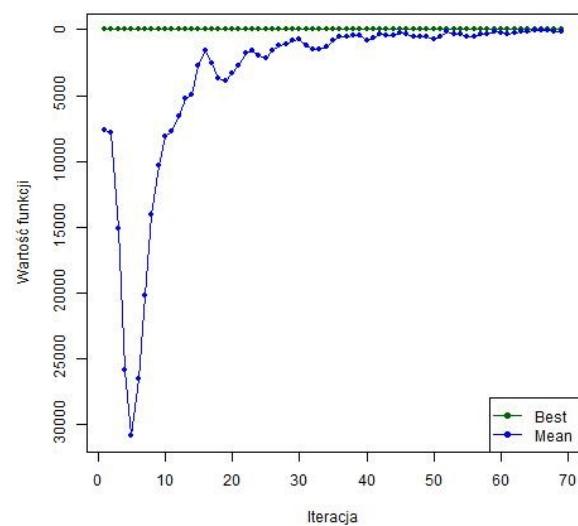
Rysunek 3.4.30: Iteracja 0, 1 i 5



Rysunek 3.4.31: Iteracja 10, 20 i 35

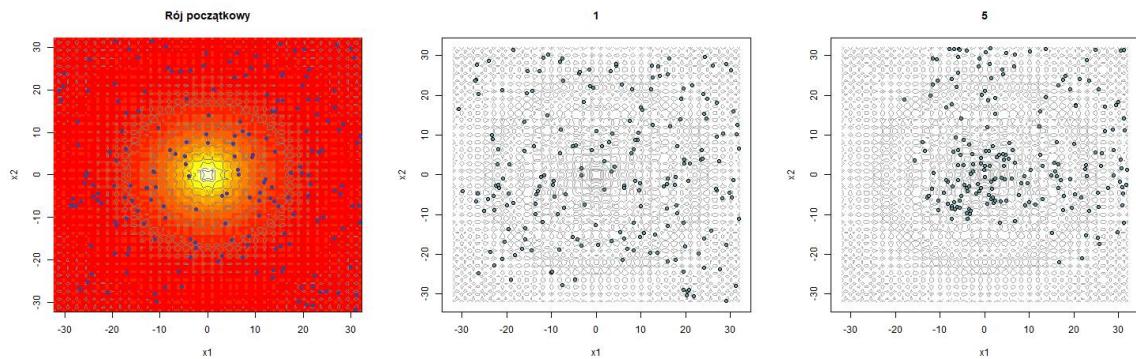


Rysunek 3.4.32: Iteracja 50, 69 i 69

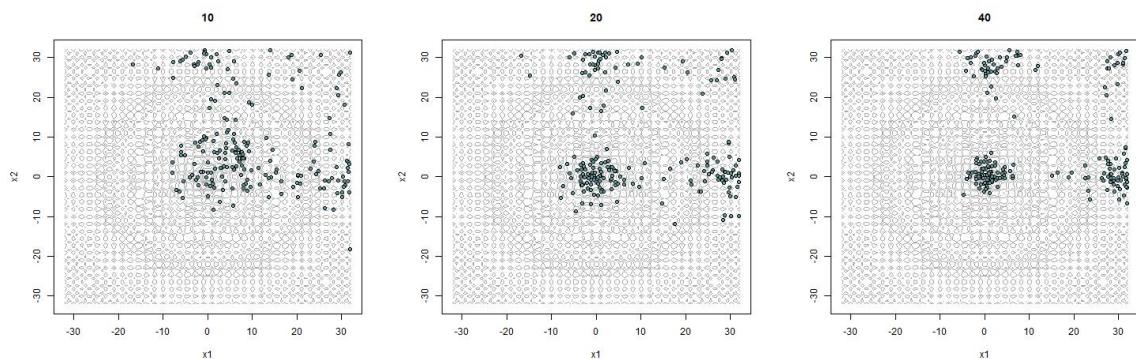


Rysunek 3.4.33: Wartości funkcji z biegiem iteracji

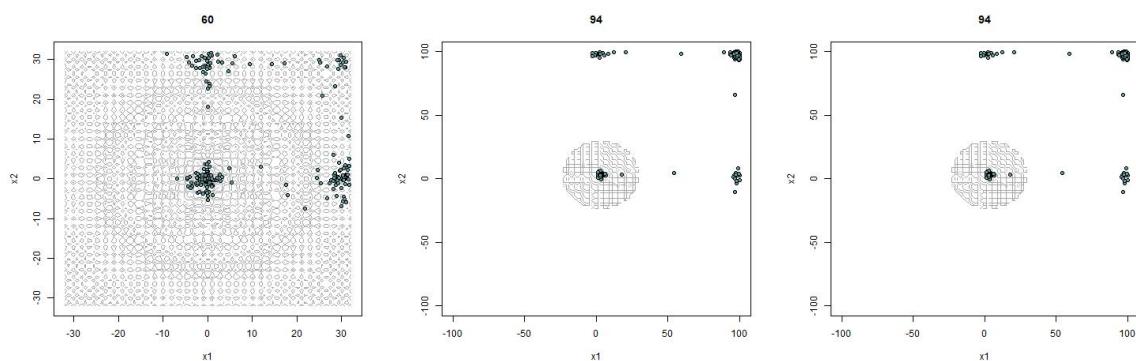
3.4.4. Easom



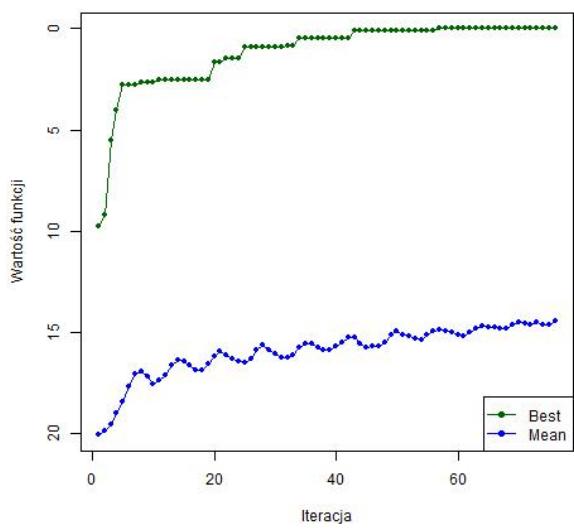
Rysunek 3.4.34: Iteracja 0, 1 i 5



Rysunek 3.4.35: Iteracja 10, 20 i 40

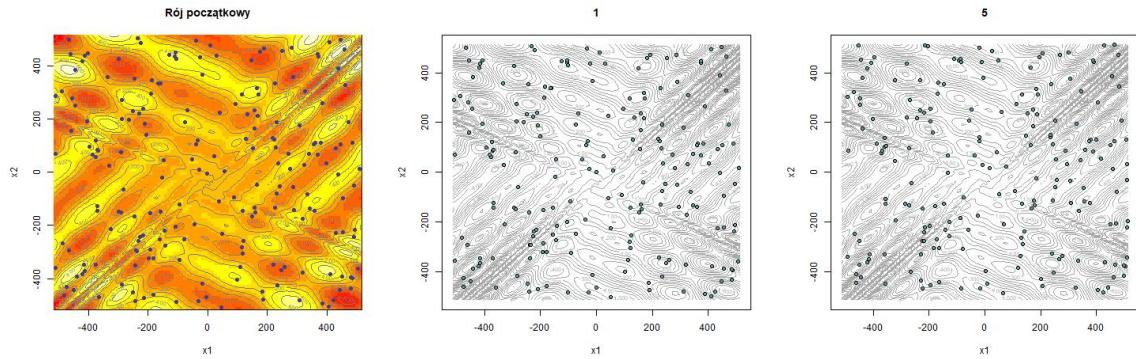


Rysunek 3.4.36: Iteracja 60, 94 i 94

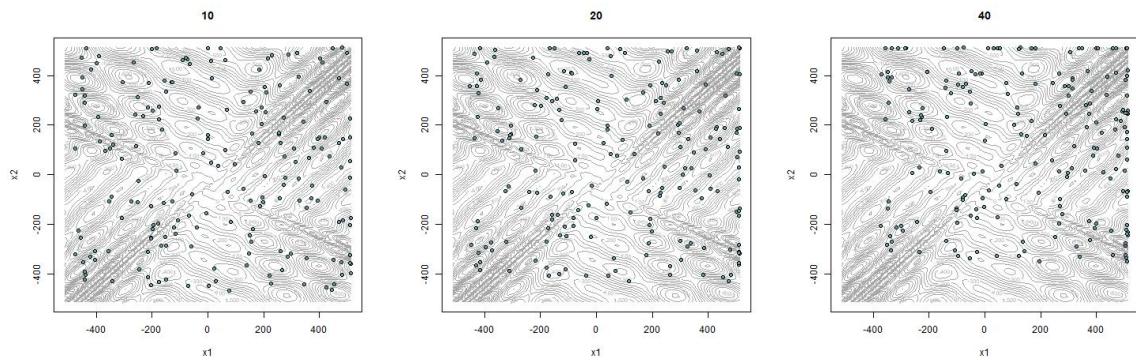


Rysunek 3.4.37: Wartości funkcji z biegiem iteracji

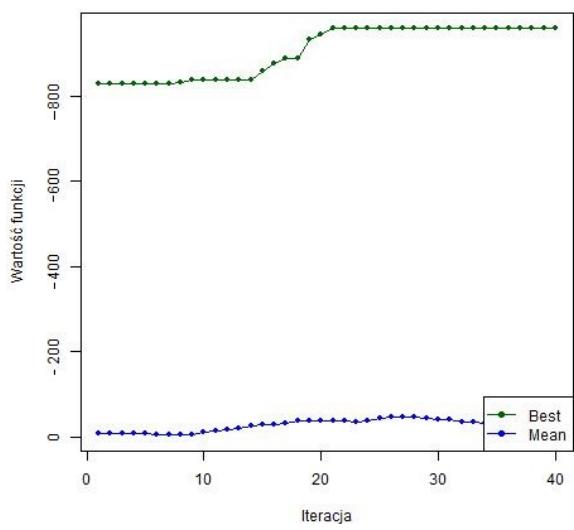
3.4.5. Eggholder



Rysunek 3.4.38: Iteracja 0, 1 i 5

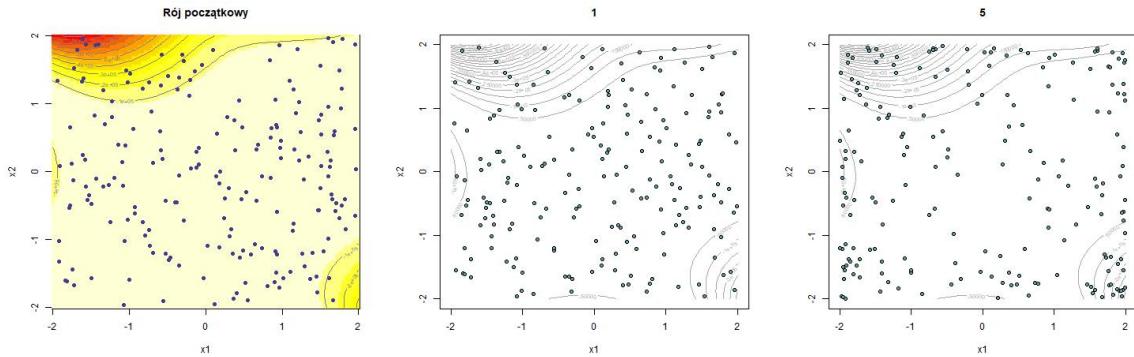


Rysunek 3.4.39: Iteracja 10, 20 i 40

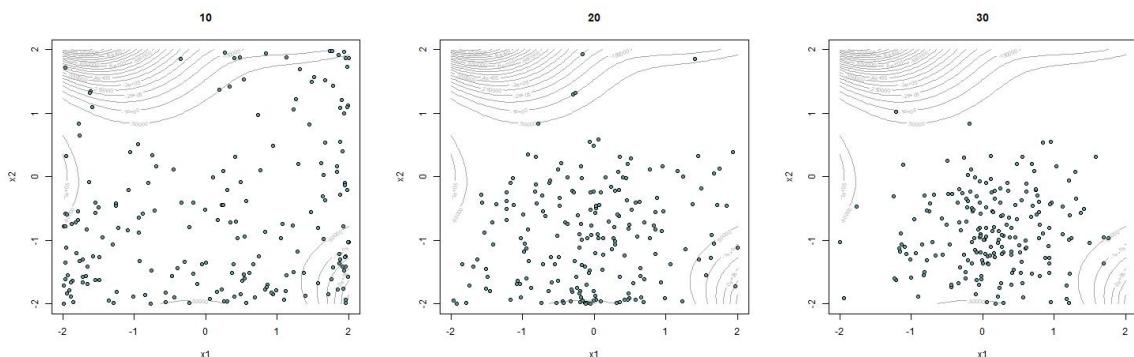


Rysunek 3.4.40: Wartości funkcji z biegiem iteracji

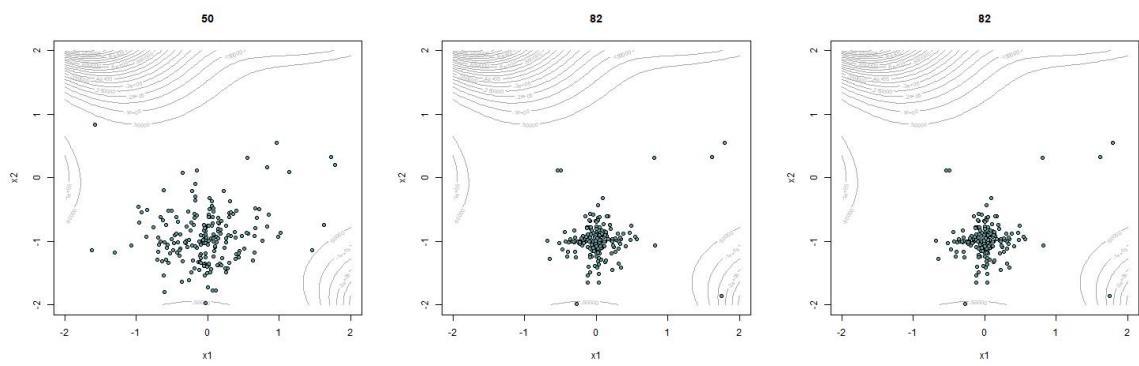
3.4.6. Goldstein



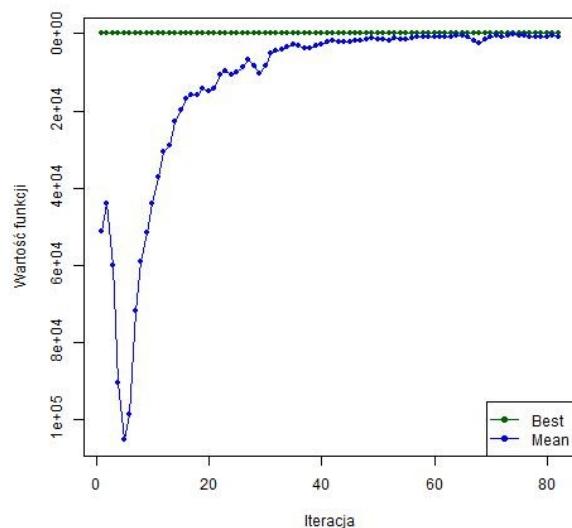
Rysunek 3.4.41: Iteracja 0, 1 i 5



Rysunek 3.4.42: Iteracja 10, 20 i 30

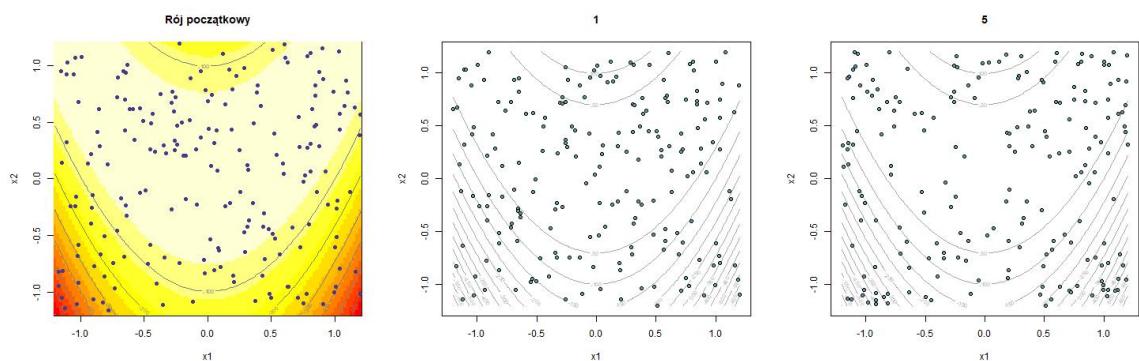


Rysunek 3.4.43: Iteracja 50, 82 i 82

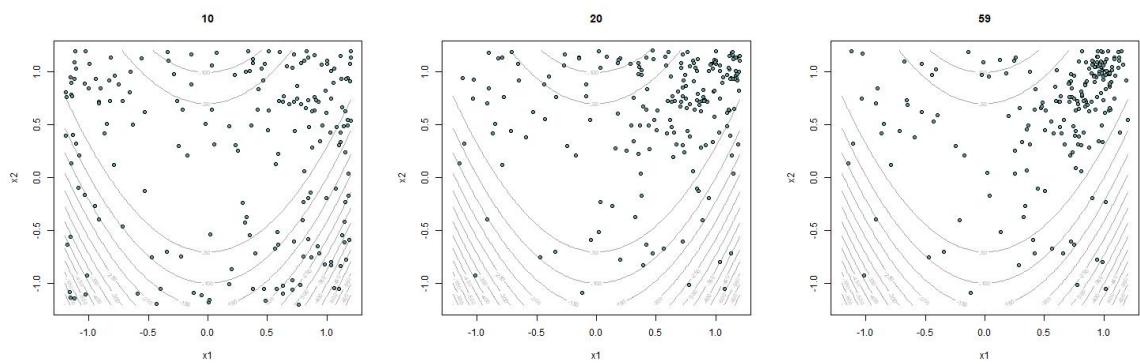


Rysunek 3.4.44: Wartości funkcji z biegiem iteracji

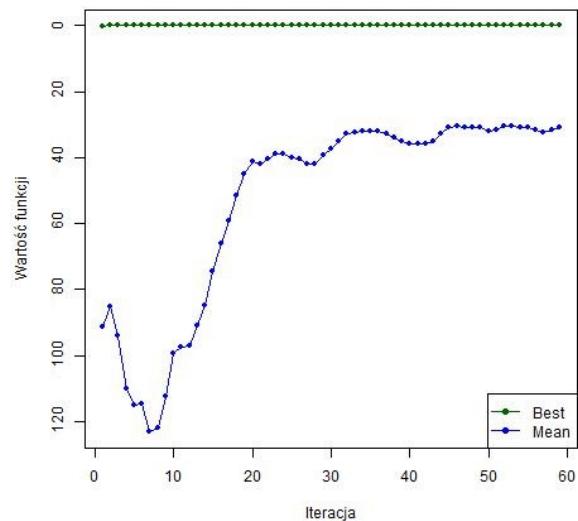
3.4.7. Leon



Rysunek 3.4.45: Iteracja 0, 1 i 5

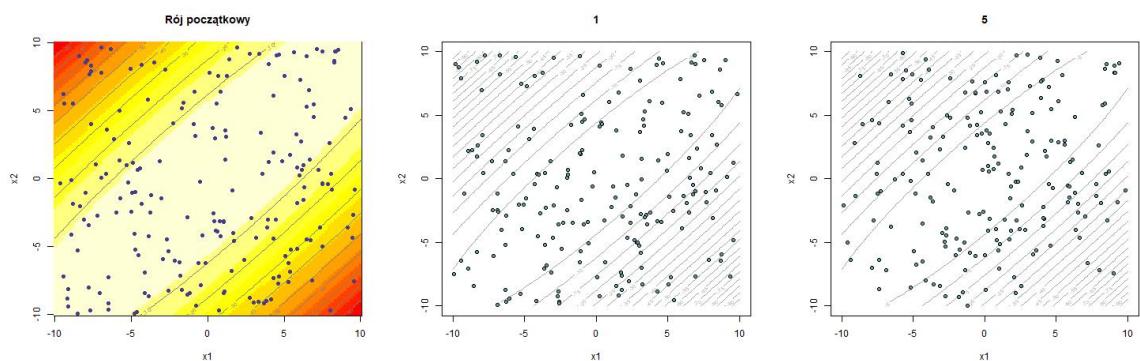


Rysunek 3.4.46: Iteracja 10, 20 i 59

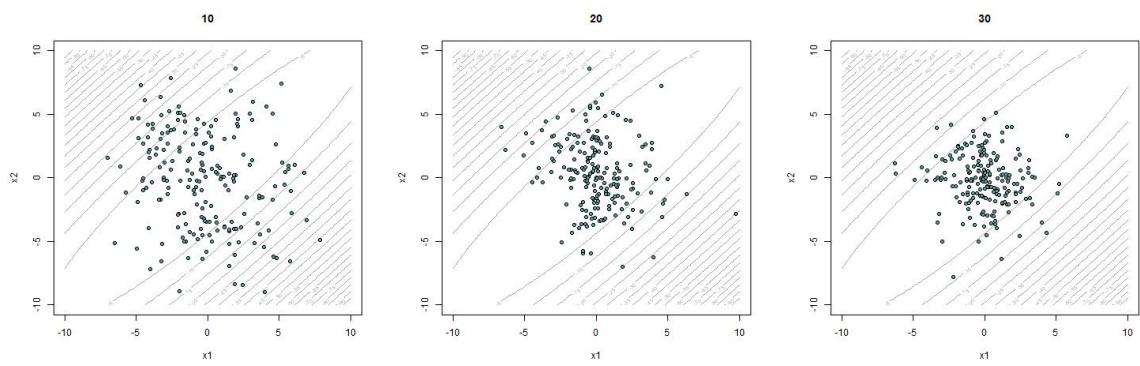


Rysunek 3.4.47: Wartości funkcji z biegiem iteracji

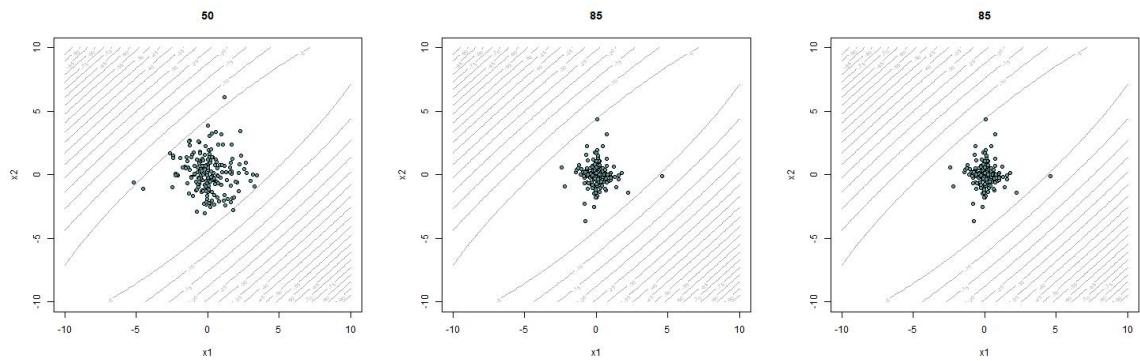
3.4.8. Matyas



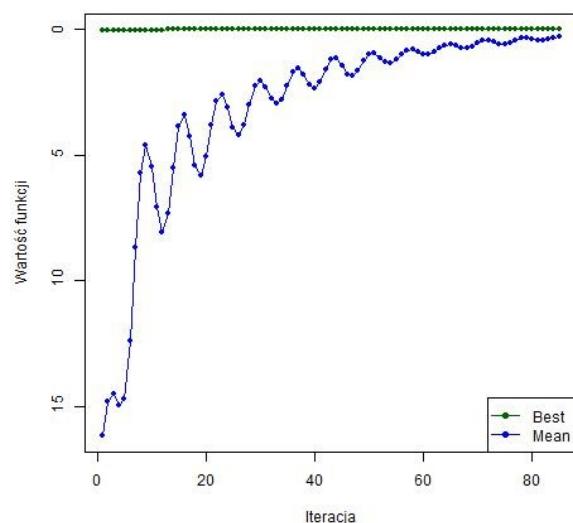
Rysunek 3.4.48: Iteracja 0, 1 i 5



Rysunek 3.4.49: Iteracja 10, 20 i 30

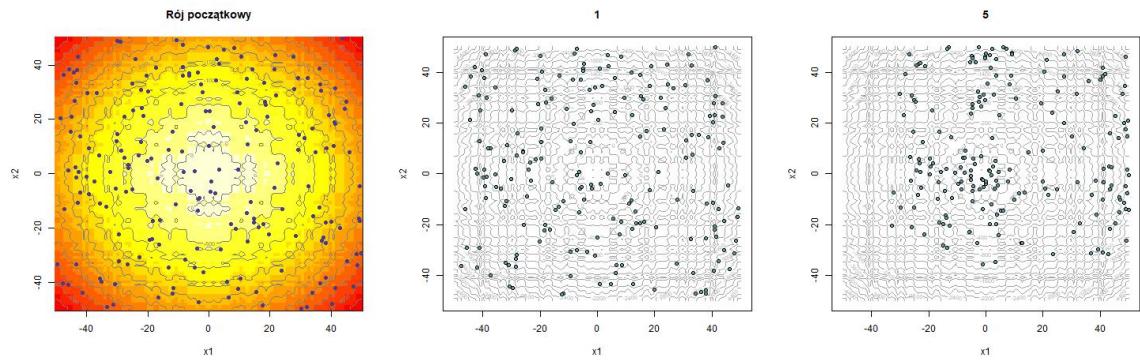


Rysunek 3.4.50: Iteracja 50, 85 i 85

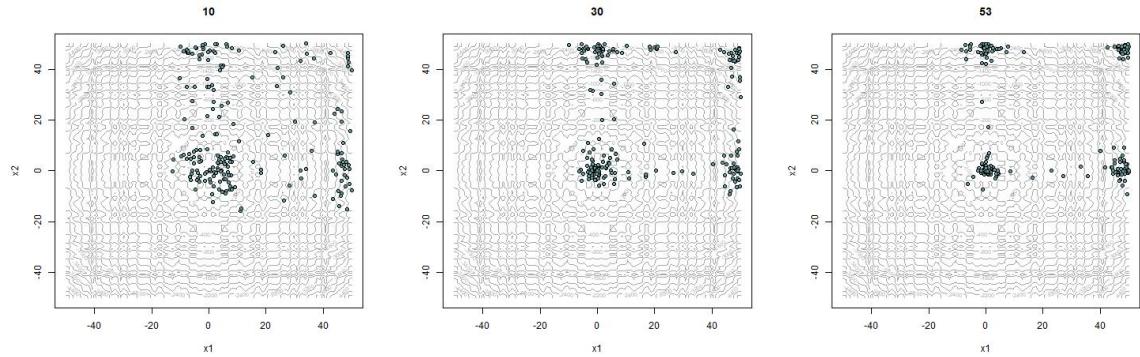


Rysunek 3.4.51: Wartości funkcji z biegiem iteracji

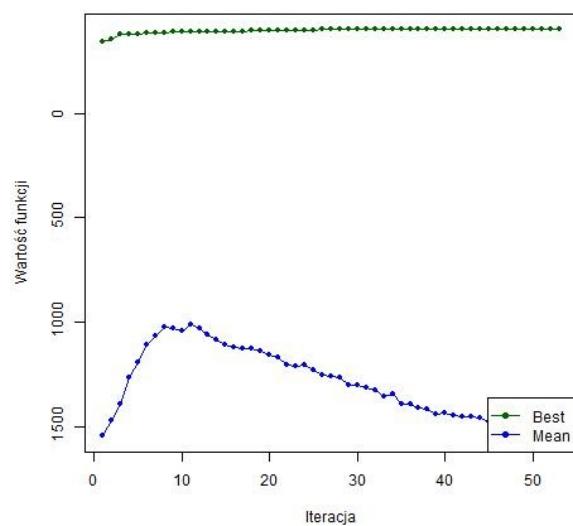
3.4.9. Venter



Rysunek 3.4.52: Iteracja 0, 1 i 5

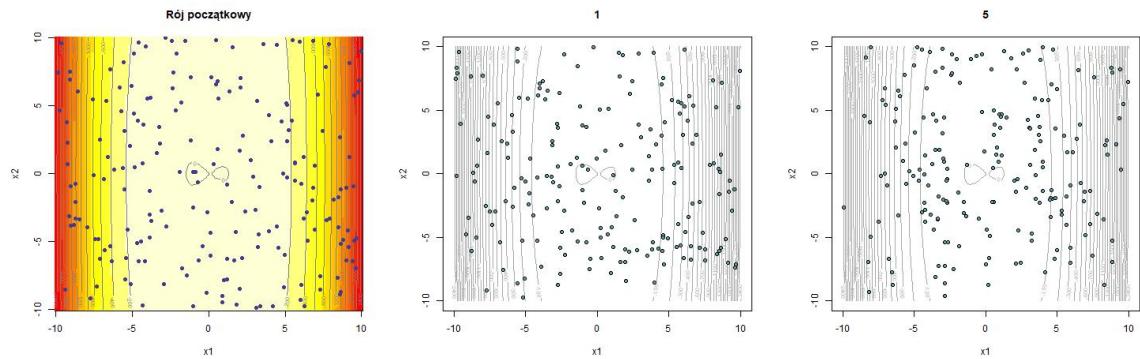


Rysunek 3.4.53: Iteracja 10, 30 i 53

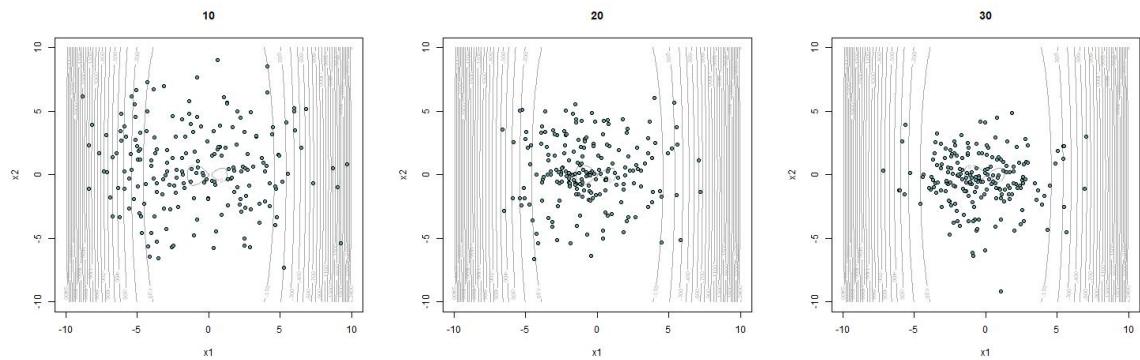


Rysunek 3.4.54: Wartości funkcji z biegiem iteracji

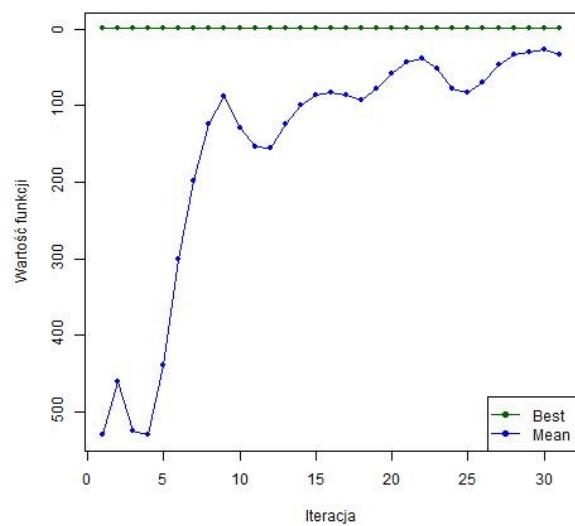
3.4.10. Zirilli



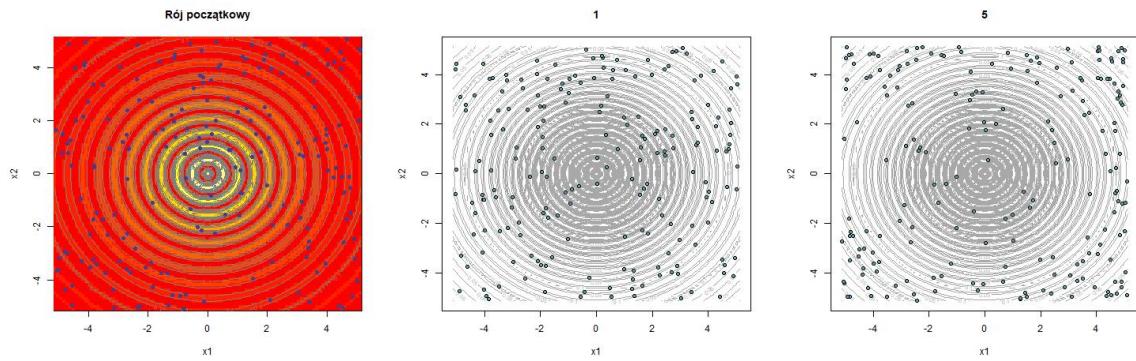
Rysunek 3.4.55: Iteracja 0, 1 i 5



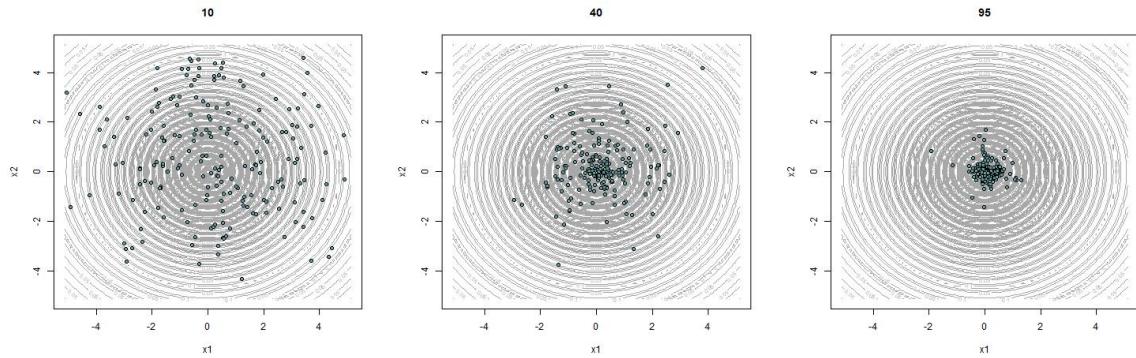
Rysunek 3.4.56: Iteracja 10, 20 i 30



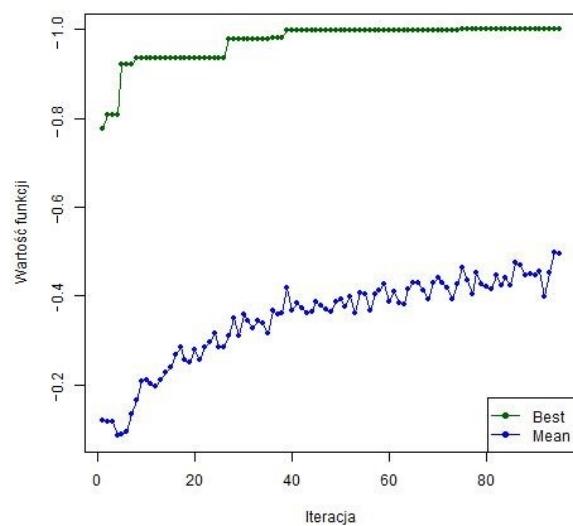
3.4.11. Drop Wave



Rysunek 3.4.57: Iteracja 0, 1 i 5

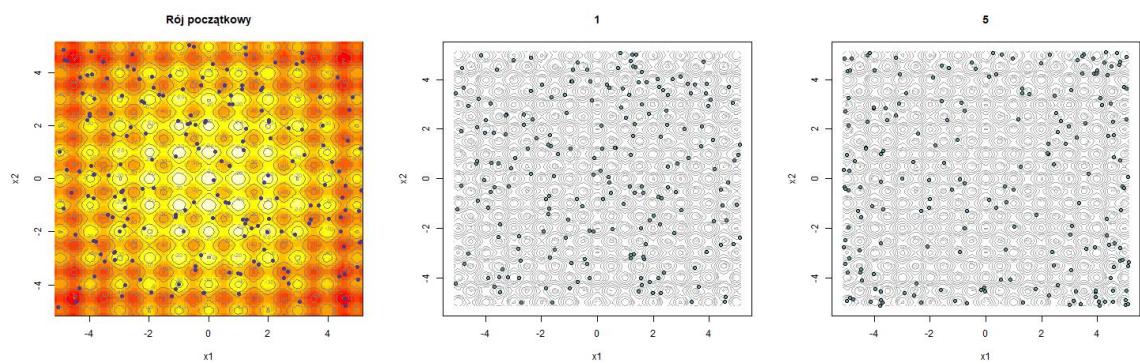


Rysunek 3.4.58: Iteracja 10, 40 i 95

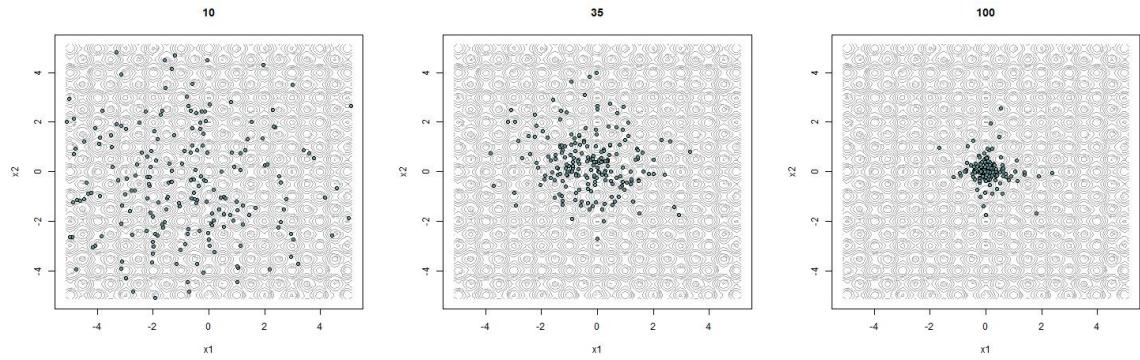


Rysunek 3.4.59: Wartości funkcji z biegiem iteracji

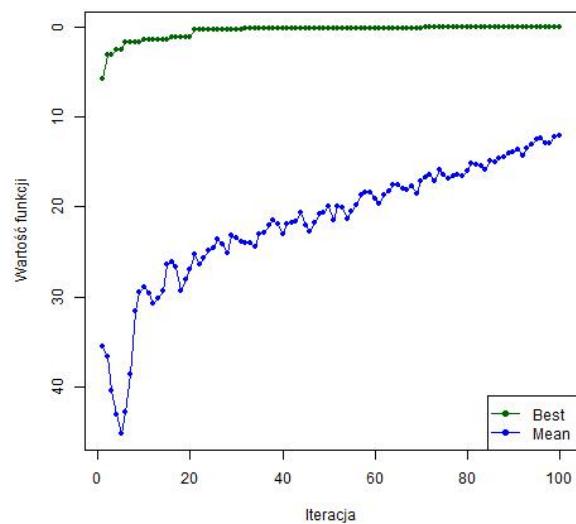
3.4.12. Rastrigin



Rysunek 3.4.60: Iteracja 0, 1 i 5

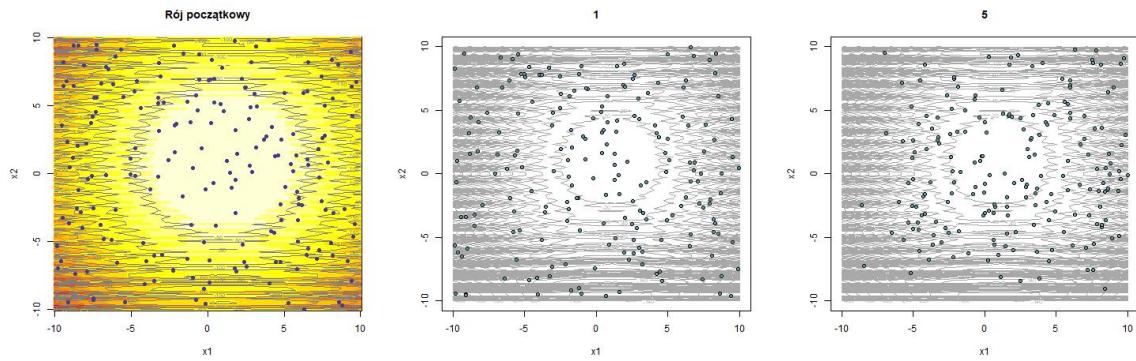


Rysunek 3.4.61: Iteracja 10, 35 i 100

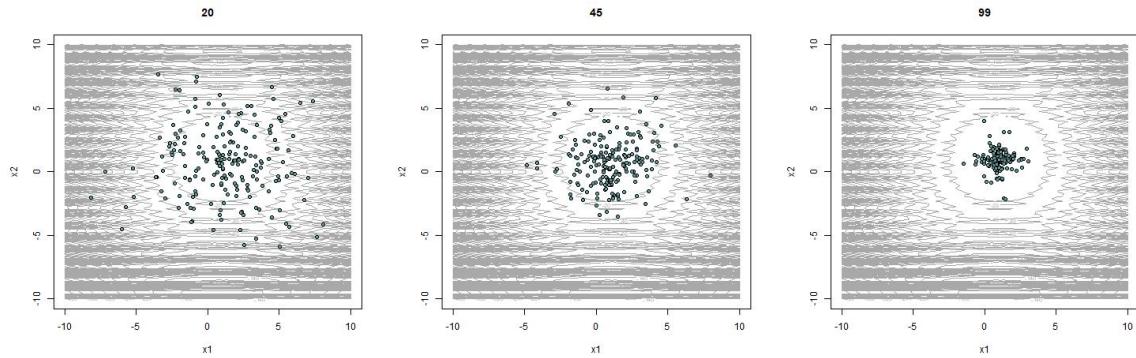


Rysunek 3.4.62: Wartości funkcji z biegiem iteracji

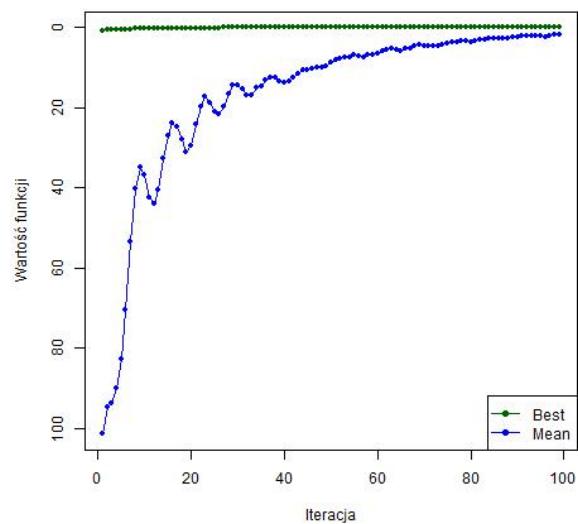
3.4.13. Levy N.13



Rysunek 3.4.63: Iteracja 0, 1 i 5



Rysunek 3.4.64: Iteracja 20, 45 i 99



Rysunek 3.4.65: Wartości funkcji z biegiem iteracji

3.5. Optymalizacja algorytmem nietoperza

Optymalizacja została przeprowadzona algorytmem nietoperza z pakietu *microbats*.

Argumenty funkcji programu wprowadzono następujące:

- nazwa funkcji
- liczba nietoperzy: [20,40,70,100,200]
- maksymalna liczba iteracji: 100
- liczba powtórzeń o identycznym wyniku zatrzymująca pracę programu: 20
- „głośność” nietoperzy: 0,5
- szybkość impulsów: 0,5
- minimalna częstotliwość: 0
- maksymalna częstotliwość: 2
- wektor określający dolne ograniczenia wartości zmiennych
- wektor określający górne ograniczenia wartości zmiennych
- liczba określająca tzw. ziarno dla generatora liczb pseudolosowych argument stosowany w celu uzyskania powtarzalności otrzymywanych wyników: rand(0:1000)

Wykonano zestawy 50 prób badania dla każdej funkcji oraz liczebności zbioru i na ich podstawie wygenerowano wyniki średnie, które zamieszczono w tabelach 3.5.7 - 3.5.11.

Tabela 3.5.7: Uśrednione wyniki optymalizacji algorytmem nietoperza

Ackley								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,266	0,011	5,321	35,02	2,61	86,6	0,0232
2	40	0,039	-0,176	3,190	15,46	2,32	72	0,0352
3	70	-0,101	0,156	2,181	8,74	2,02	61,1	0,0716
4	100	0,041	0,155	1,652	6,23	1,89	47,5	0,0876
5	200	-0,019	0,001	0,872	2,47	1,32	43,64	0,1672
		Wartości szukane		0	0			

Beale								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	1,583	0,766	0,301	0,27	0,43	97,72	0,024
2	40	1,560	0,677	0,183	0,14	0,33	88,86	0,0428
3	70	1,595	0,622	0,134	0,10	0,29	70,4	0,0826
4	100	2,047	0,581	0,088	0,07	0,24	61,18	0,0954
5	200	2,665	0,526	0,029	0,02	0,14	51,92	0,1758
		Wartości szukane		3	0,5	0		

Goldstein								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	-0,008	-0,923	8,200	249,86	15,08	73,86	0,0154
2	40	-0,060	-0,940	5,700	72,90	8,18	47,3	0,0196
3	70	0,000	-1,000	3,000	0,00	0,00	40,54	0,0416
4	100	0,000	-1,000	3,000	0,00	0,00	39,6	0,0542
5	200	0,000	-1,000	3,000	0,00	0,00	34,78	0,1242
		Wartości szukane		0	-1	3		

Tabela 3.5.8: cd. Uśrednione wyniki optymalizacji algorytmem nietoperza

Bartels Conn						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	-6,679	-2,472	1474,761	6024176,81	1982,63
2	40	-0,466	0,656	625,355	1143986,53	877,25
3	70	-1,842	-0,897	246,037	174922,97	342,38
4	100	1,385	-0,423	56,269	9039,46	78,15
5	200	-0,639	0,931	8,054	260,26	14,66
Wartości szukane						
		0	0	1		

Leon						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	0,902	0,908	0,104	0,29	0,54
2	40	1,005	1,011	0,001	0,00	0,0544
3	70	1,000	1,001	0,000	0,00	0,094
4	100	1,000	1,000	0,000	0,00	0,1276
5	200	1,000	1,000	0,000	0,00	0,1884
Wartości szukane						
		1	1	0		

Eggholder						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	55,148	144,065	-755,063	58159,56	130,03
2	40	1,490	79,190	-793,781	36654,19	97,71
3	70	77,336	226,536	-843,395	22014,37	93,95
4	100	116,847	225,280	-854,870	18877,59	90,55
5	200	228,957	363,206	-918,208	5226,82	60,30
Wartości szukane						
		512	404	-959		

Tabela 3.5.9: cd. Uśrednione wyniki optymalizacji algorytmem nietoperza

Ventter						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	1,426	0,721	-299,963	20691,25	104,41
2	40	0,024	-0,479	-365,818	3317,82	46,83
3	70	-0,807	0,185	-379,007	1547,07	33,60
4	100	0,123	0,000	-385,770	298,83	9,91
5	200	0,000	0,431	-386,059	276,85	10,04
		Wartości szukane	0	0	-400	
Matyas						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	-0,081	-0,058	0,096	0,10	0,30
2	40	0,016	0,016	0,003	0,00	0,01
3	70	-0,002	-0,002	0,000	0,00	0,00
4	100	0,000	0,000	0,000	0,00	0,00
5	200	0,000	0,000	0,000	0,00	0,00
		Wartości szukane	0	0	0	
Zirilli						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	-0,072	-0,048	-0,128	0,16	0,33
2	40	-0,250	-0,001	-0,272	0,02	0,10
3	70	-0,808	0,000	-0,328	0,00	0,07
4	100	-0,728	0,000	-0,320	0,01	0,07
5	200	-1,007	0,000	-0,348	0,00	0,03
		Wartości szukane	-1,04	0	-0,35	

Tabela 3.5.10: cd. Uśrednione wyniki optymalizacji algorytmem nietoperza

Drop Wave						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe
1	20	-0,119	0,143	-0,764	0,07	0,14
2	40	-0,010	0,043	-0,885	0,02	0,10
3	70	0,012	0,035	-0,928	0,01	0,06
4	100	0,039	-0,020	-0,930	0,01	0,04
5	200	-0,064	-0,046	-0,940	0,00	0,02
		Wartości szukane	0	0	-1	

Levy N.13						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe
1	20	0,934	1,167	0,989	2,57	1,27
2	40	1,059	0,997	0,339	0,32	0,46
3	70	1,013	0,962	0,135	0,04	0,16
4	100	0,974	0,988	0,122	0,06	0,21
5	200	0,967	1,005	0,050	0,01	0,08
		Wartości szukane	1	1	0	

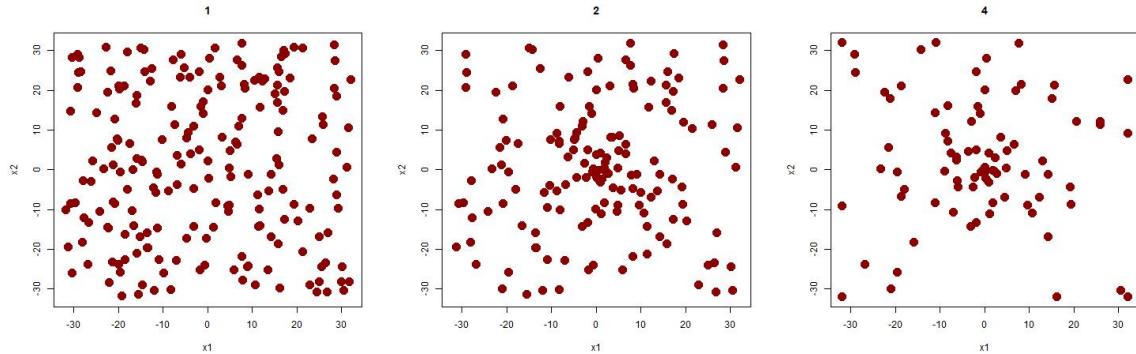
Rastrigin						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe
1	20	-0,020	0,199	3,681	29,36	4,02
2	40	0,080	-0,040	3,025	14,02	2,23
3	70	-0,159	0,040	1,672	6,14	1,85
4	100	0,020	-0,119	1,333	4,97	1,80
5	200	0,040	-0,080	0,995	1,78	0,90
		Wartości szukane	0	0	0	

Tabela 3.5.11: cd. Uśrednione wyniki optymalizacji algorytmem nietoperza

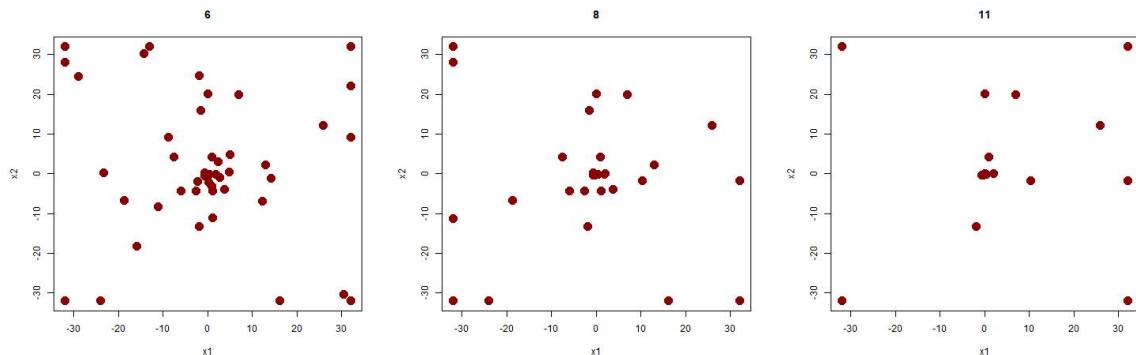
Easom						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe
1	20	-2,436	-1,023	-0,024	0,97	0,12
2	40	-3,063	1,028	-0,104	0,87	0,26
3	70	3,281	3,087	-0,256	0,73	0,42
4	100	2,954	3,453	-0,478	0,52	0,50
5	200	3,031	3,178	-0,860	0,14	0,35
	Wartości szukane	3,14	3,14	-1		

Na rysunkach 3.5.66 - 3.5.108 przedstawiono podobnie jak w przypadku PSO pozycje (populacji 200) osobników w wybranych iteracjach poszukiwania minimów funkcji przez roje.

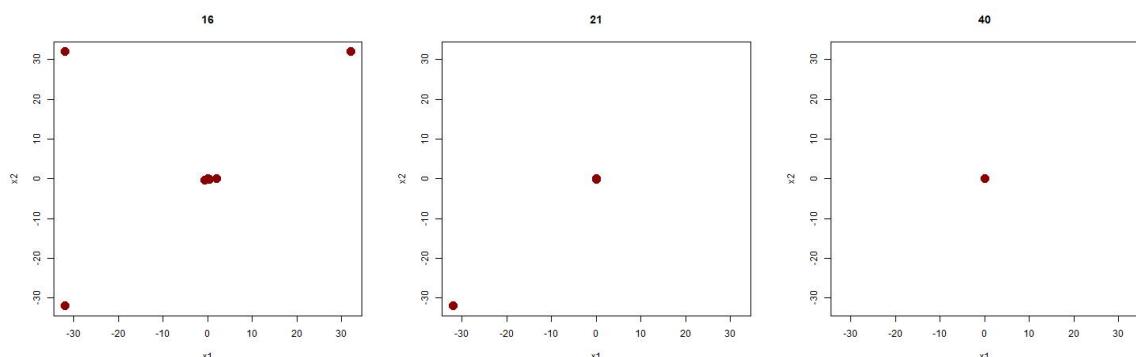
3.5.1. Ackley



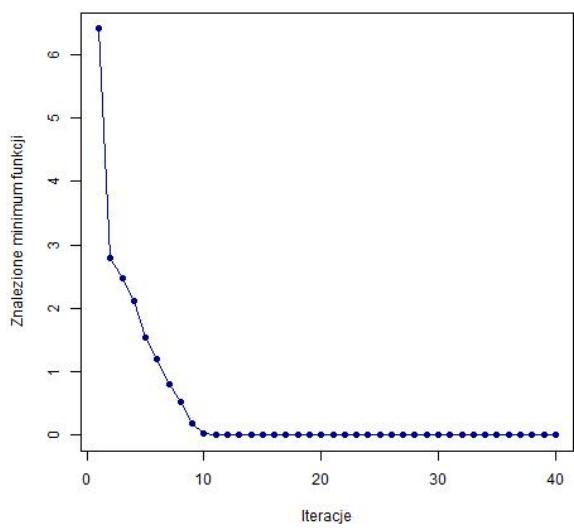
Rysunek 3.5.66: Iteracja 0, 1 i 3



Rysunek 3.5.67: Iteracja 5, 7 i 10

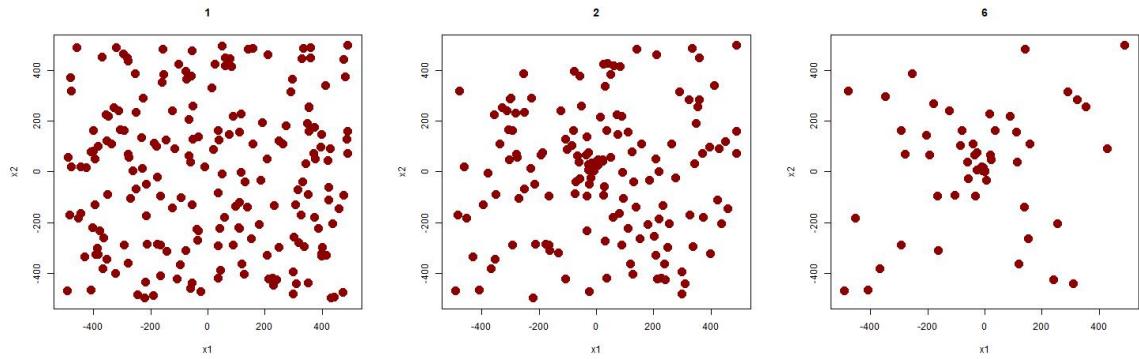


Rysunek 3.5.68: Iteracja 15, 20 i 39

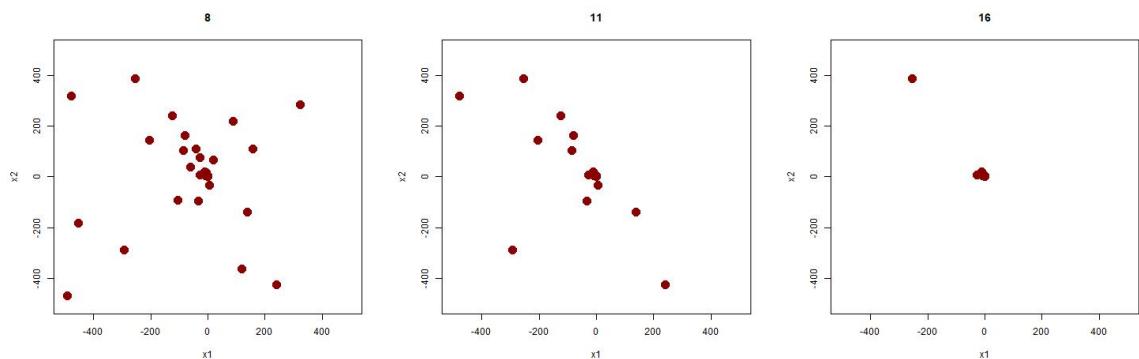


Rysunek 3.5.69: Wartości funkcji z biegiem iteracji

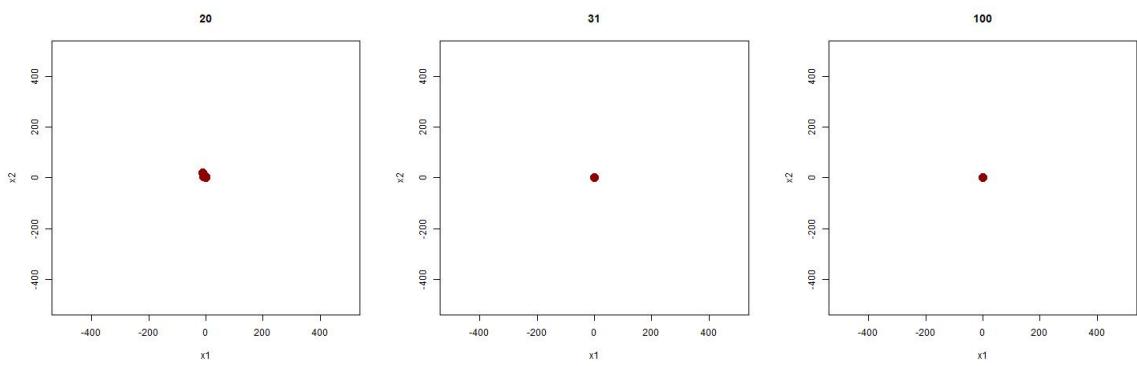
3.5.2. Bartels



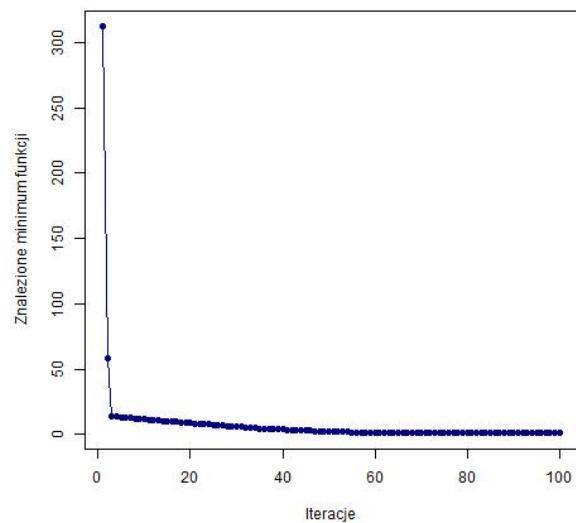
Rysunek 3.5.70: Iteracja 0, 1 i 5



Rysunek 3.5.71: Iteracja 7, 10 i 15

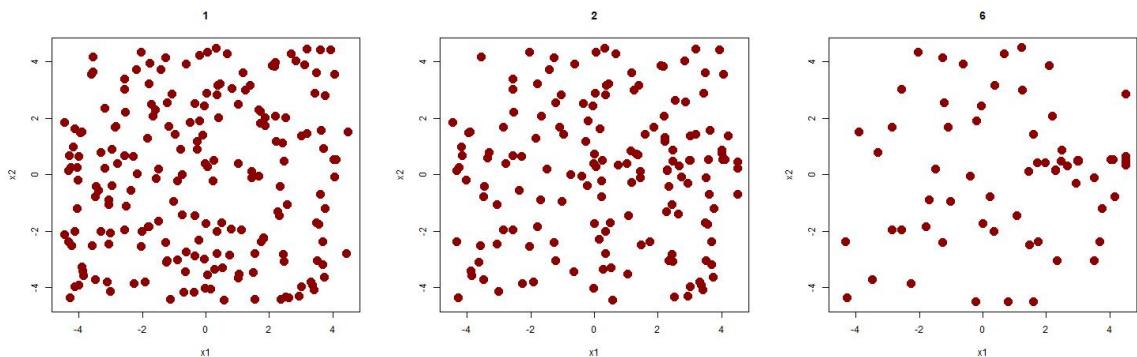


Rysunek 3.5.72: Iteracja 19, 30 i 99

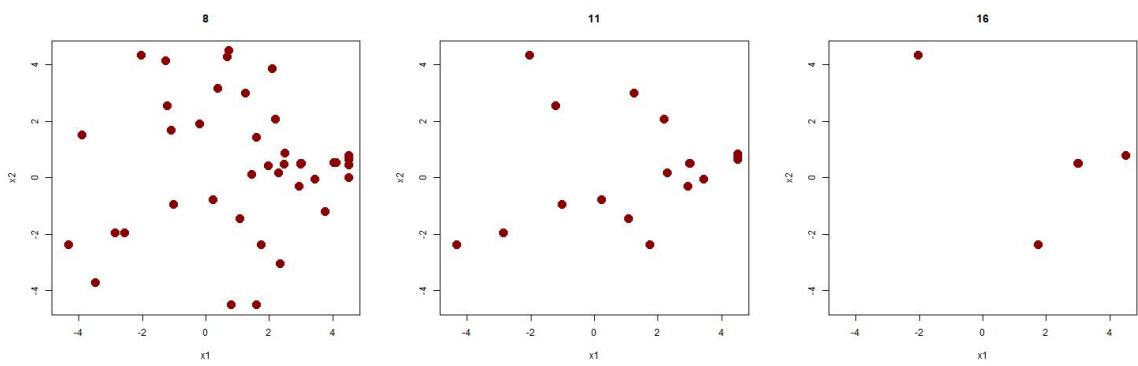


Rysunek 3.5.73: Wartości funkcji z biegiem iteracji

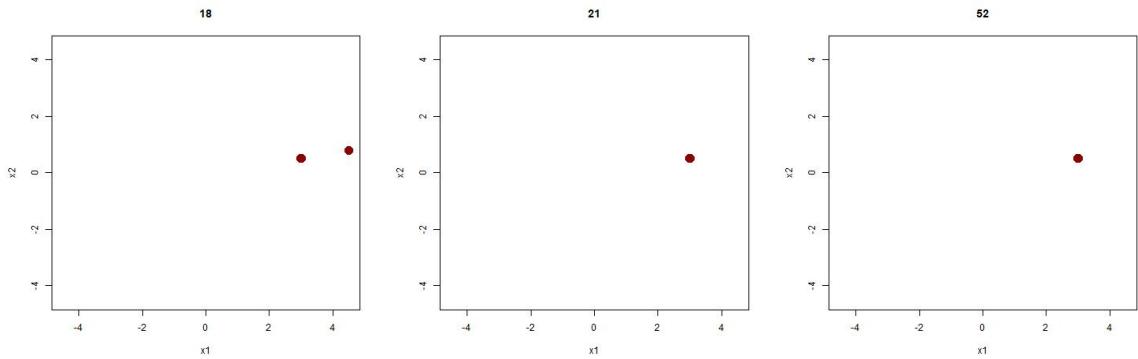
3.5.3. Beale



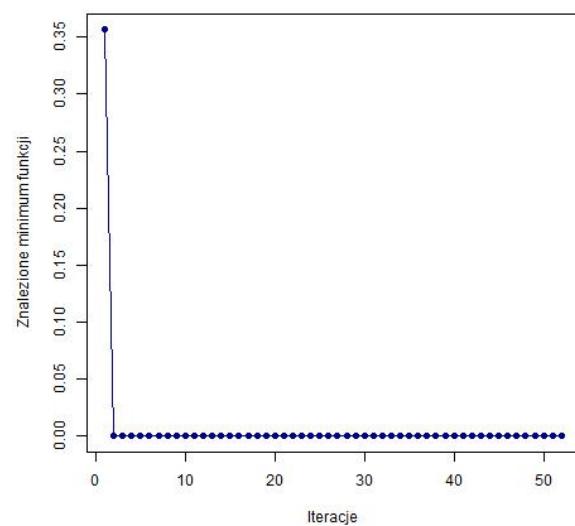
Rysunek 3.5.74: Iteracja 0, 1 i 5



Rysunek 3.5.75: Iteracja 7, 10 i 15

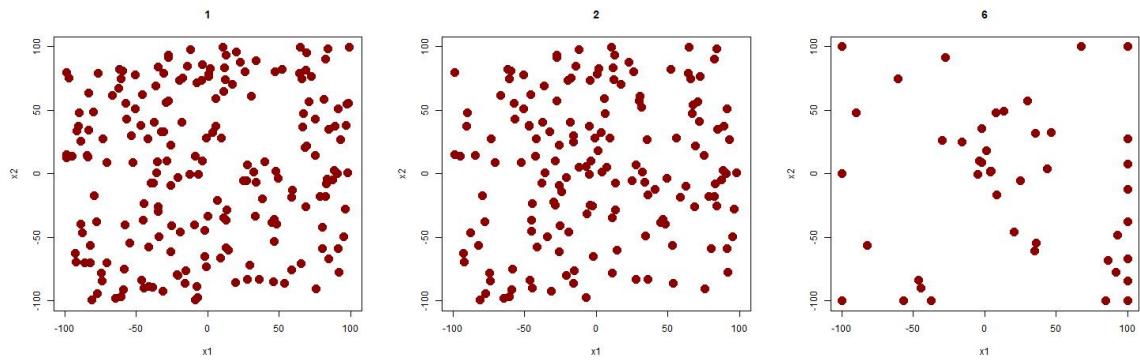


Rysunek 3.5.76: Iteracja 17, 20 i 51

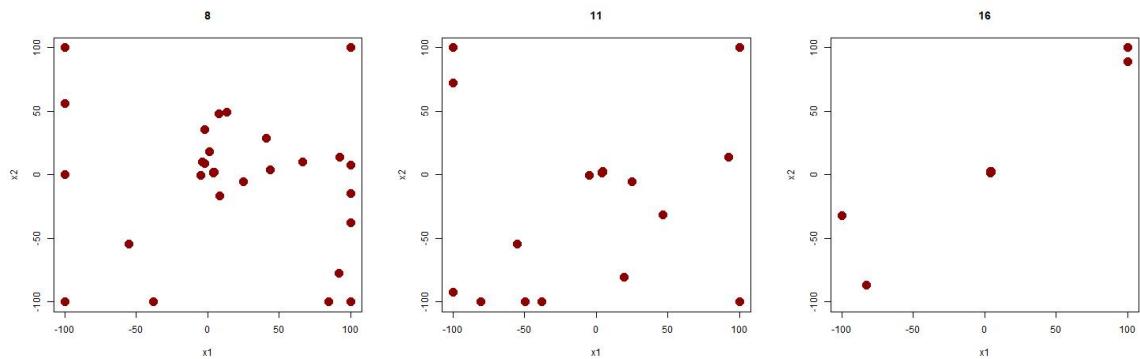


Rysunek 3.5.77: Wartości funkcji z biegiem iteracji

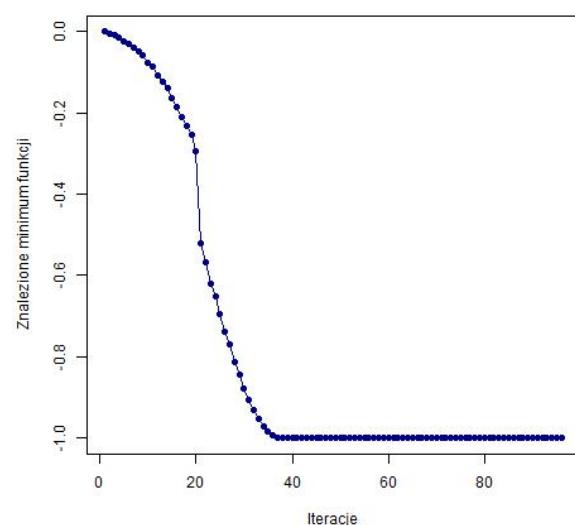
3.5.4. Easom



Rysunek 3.5.78: Iteracja 0, 1 i 5

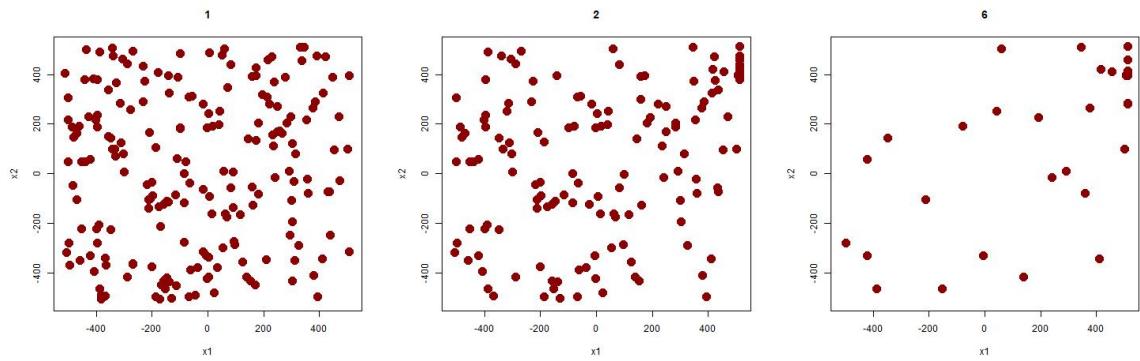


Rysunek 3.5.79: Iteracja 7, 10 i 15

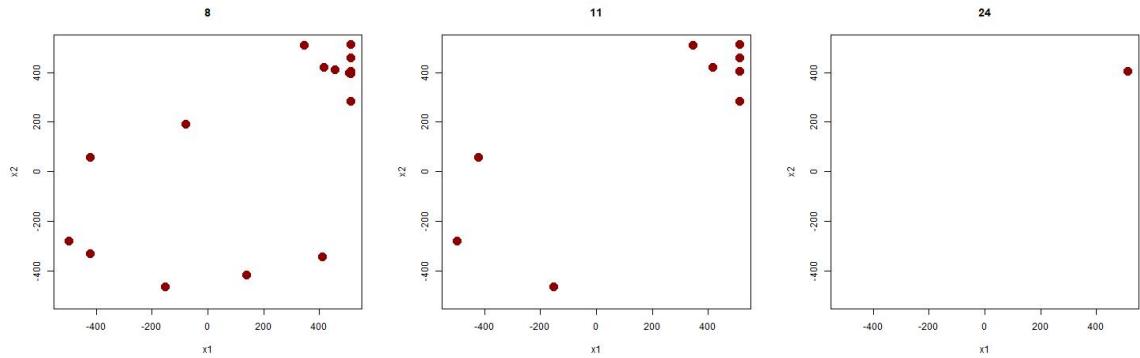


Rysunek 3.5.80: Wartości funkcji z biegiem iteracji

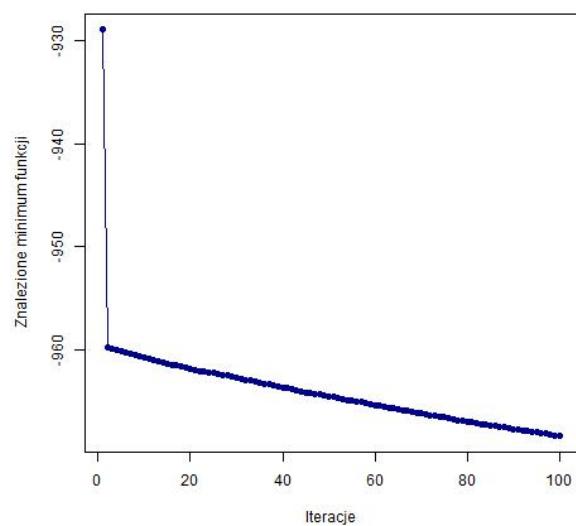
3.5.5. Eggholder



Rysunek 3.5.81: Iteracja 0, 1 i 5

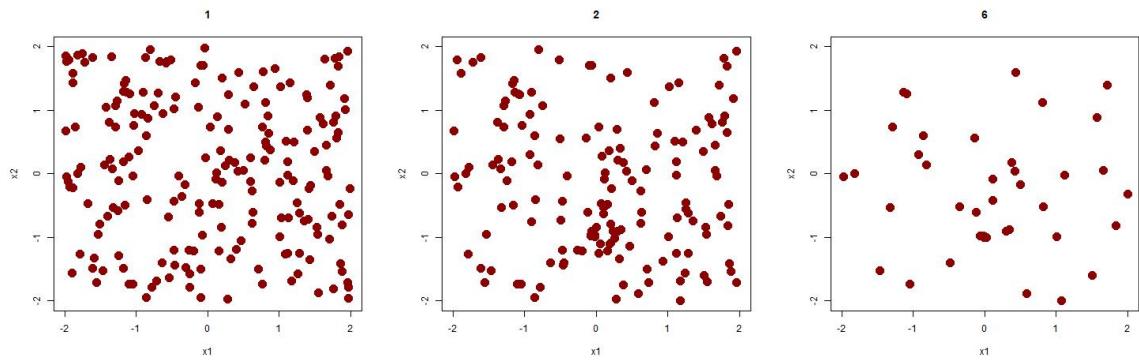


Rysunek 3.5.82: Iteracja 7, 10 i 23

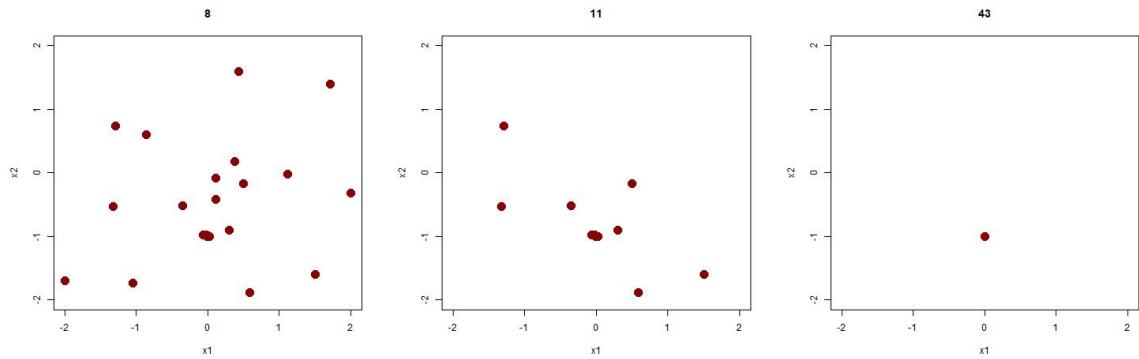


Rysunek 3.5.83: Wartości funkcji z biegiem iteracji

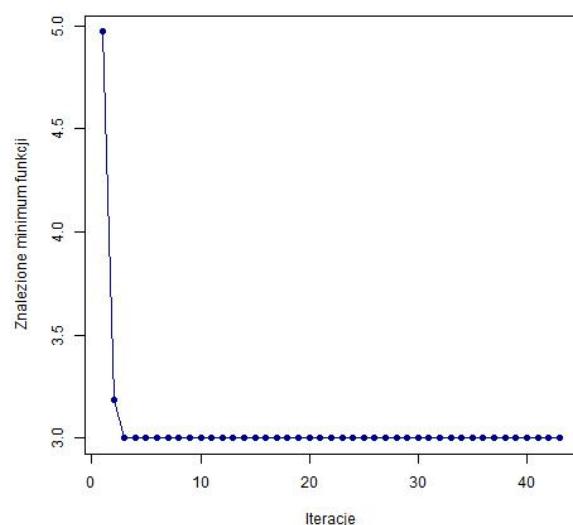
3.5.6. Goldstein



Rysunek 3.5.84: Iteracja 0, 1 i 5

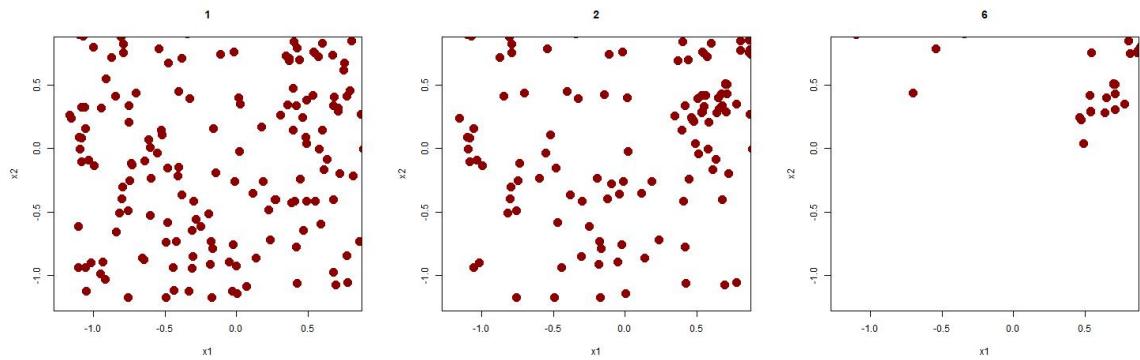


Rysunek 3.5.85: Iteracja 7, 10 i 42

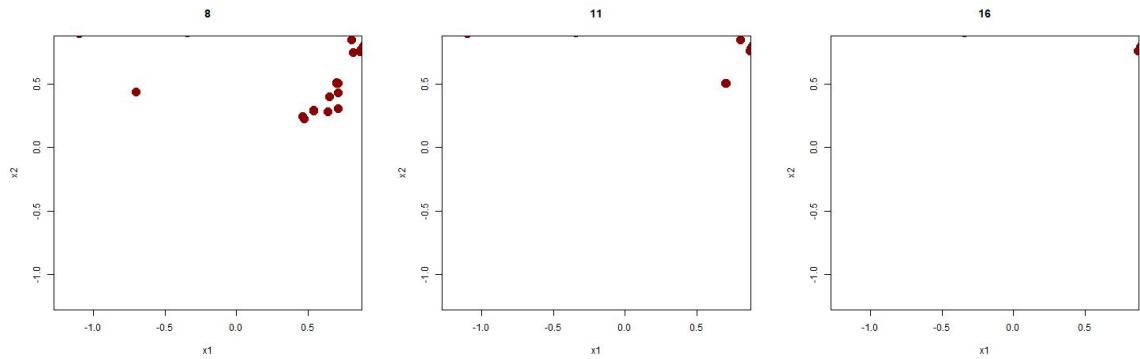


Rysunek 3.5.86: Wartości funkcji z biegiem iteracji

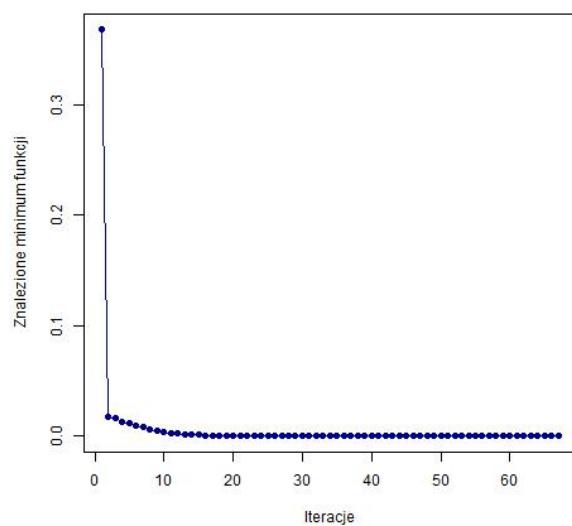
3.5.7. Leon



Rysunek 3.5.87: Iteracja 0, 1 i 5

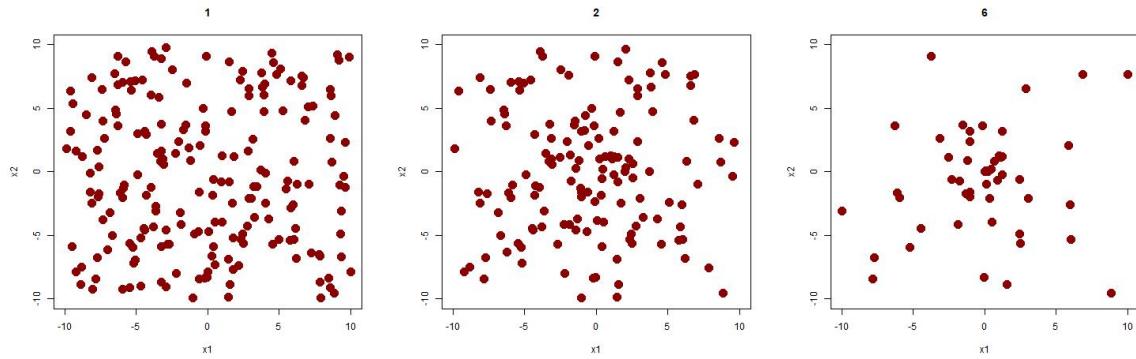


Rysunek 3.5.88: Iteracja 7, 10 i 15

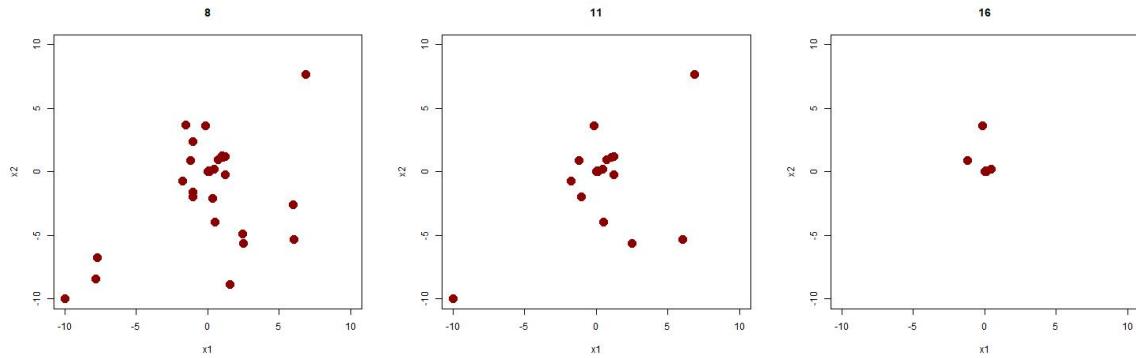


Rysunek 3.5.89: Wartości funkcji z biegiem iteracji

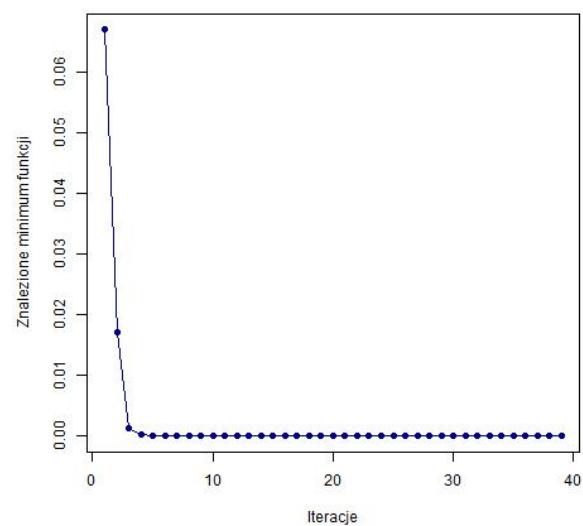
3.5.8. Matyas



Rysunek 3.5.90: Iteracja 0, 1 i 5

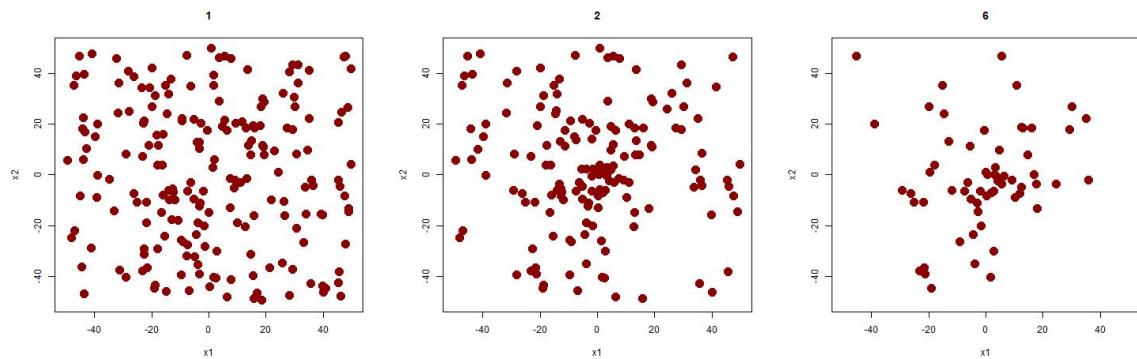


Rysunek 3.5.91: Iteracja 7, 10 i 15

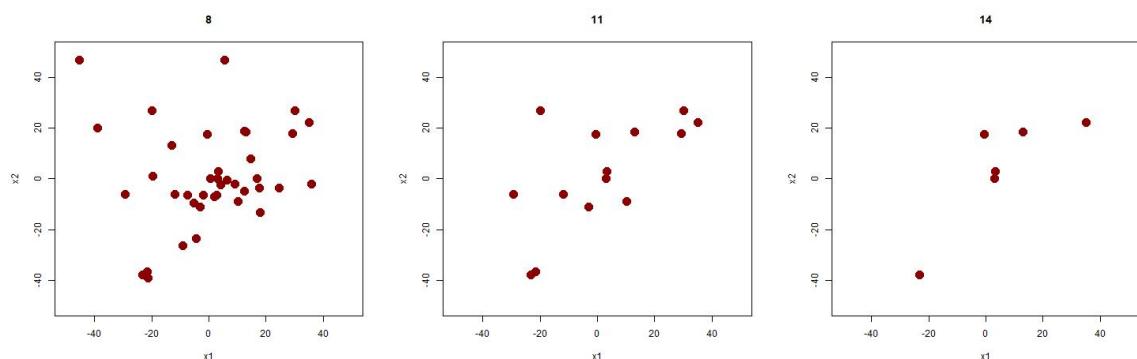


Rysunek 3.5.92: Wartości funkcji z biegiem iteracji

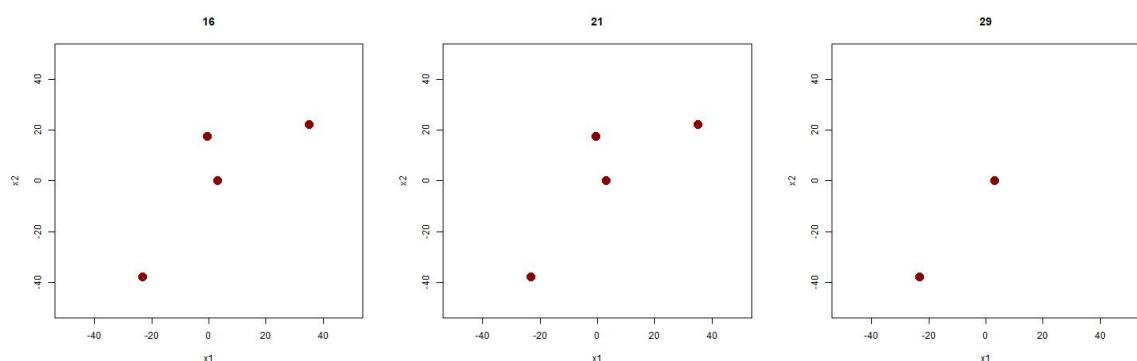
3.5.9. Venter



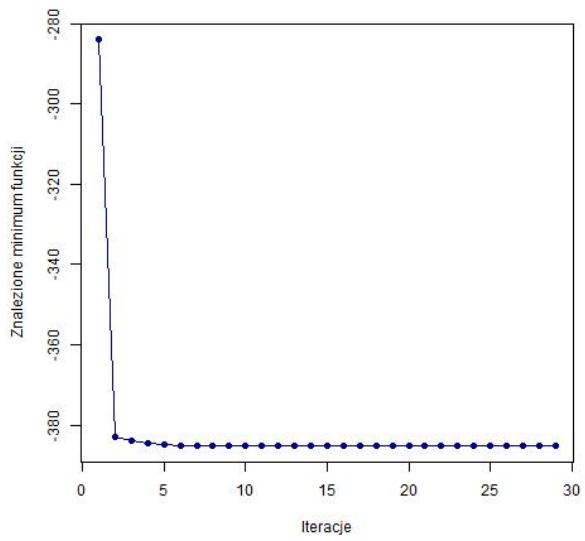
Rysunek 3.5.93: Iteracja 0, 1 i 5



Rysunek 3.5.94: Iteracja 7, 10 i 13

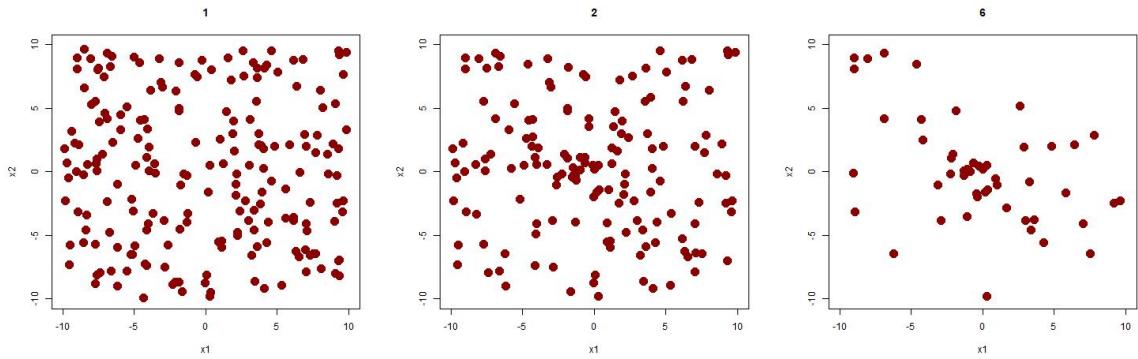


Rysunek 3.5.95: Iteracja 15, 20 i 28

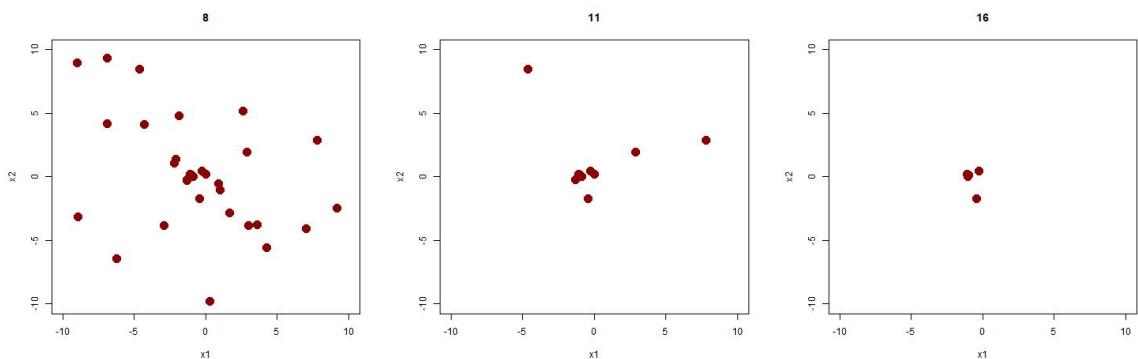


Rysunek 3.5.96: Wartości funkcji z biegiem iteracji

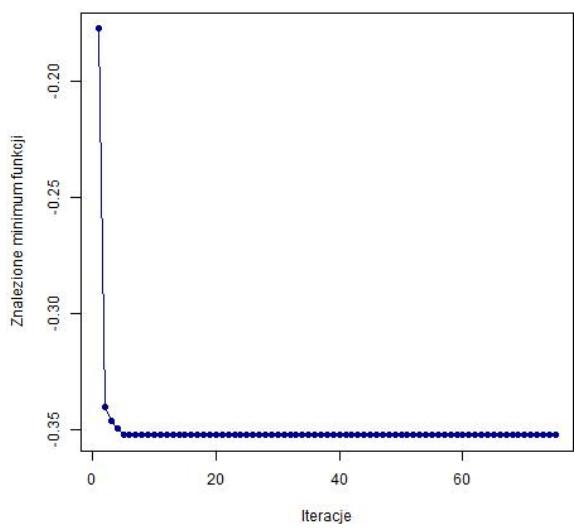
3.5.10. Zirilli



Rysunek 3.5.97: Iteracja 0, 1 i 5

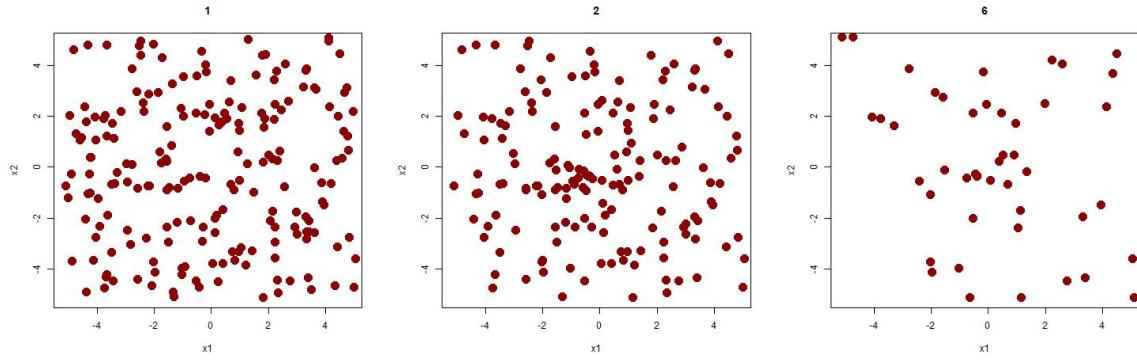


Rysunek 3.5.98: Iteracja 7, 10 i 15

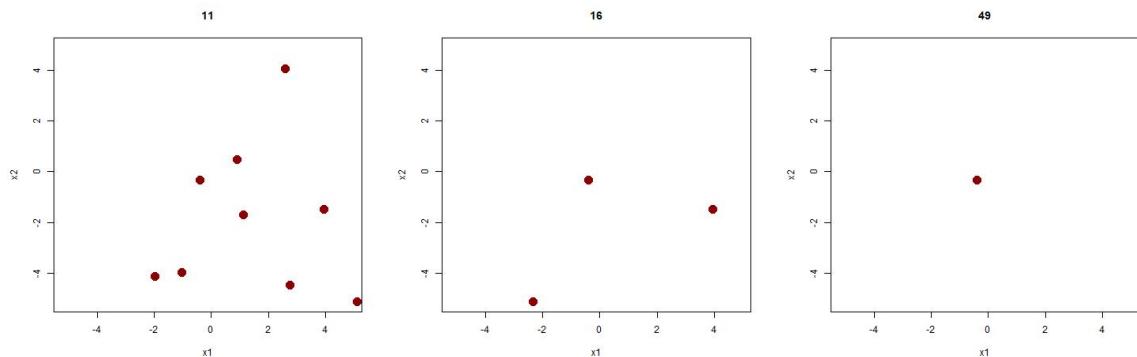


Rysunek 3.5.99: Wartości funkcji z biegiem iteracji

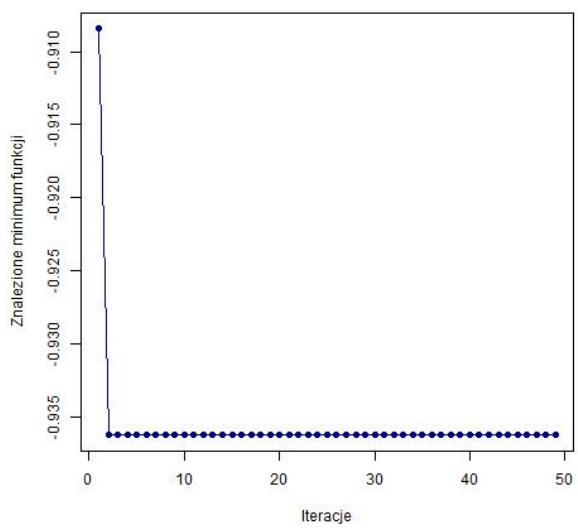
3.5.11. Drop Wave



Rysunek 3.5.100: Iteracja 0, 1 i 5

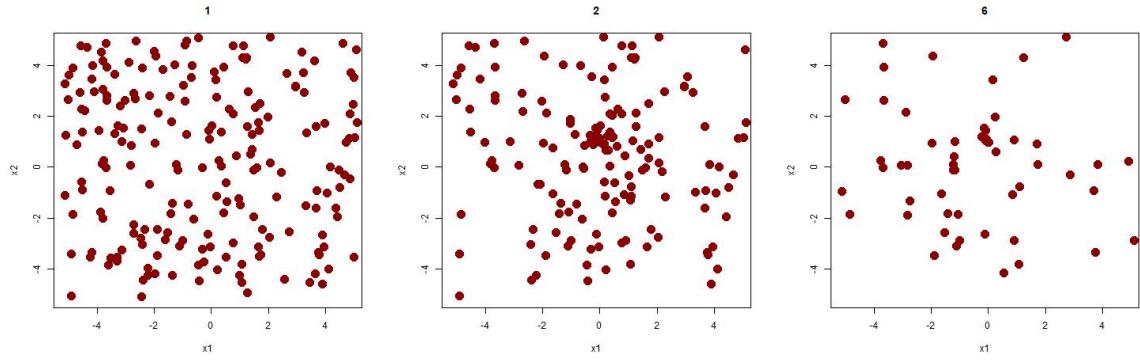


Rysunek 3.5.101: Iteracja 10, 15 i 48

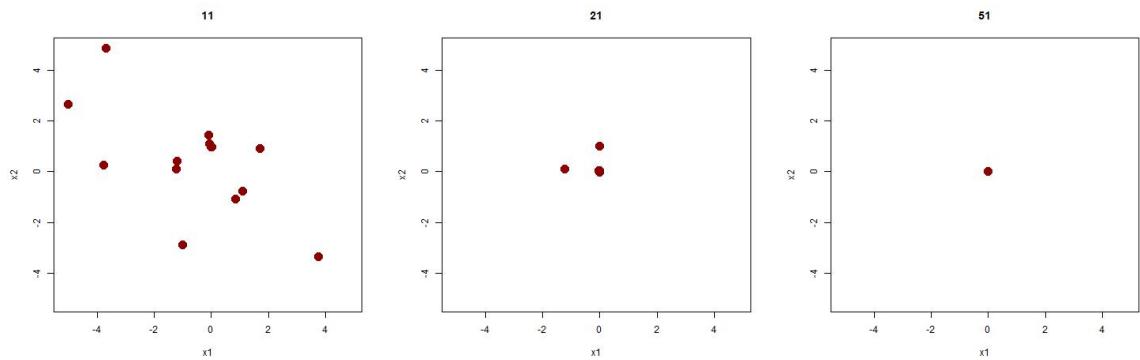


Rysunek 3.5.102: Wartości funkcji z biegiem iteracji

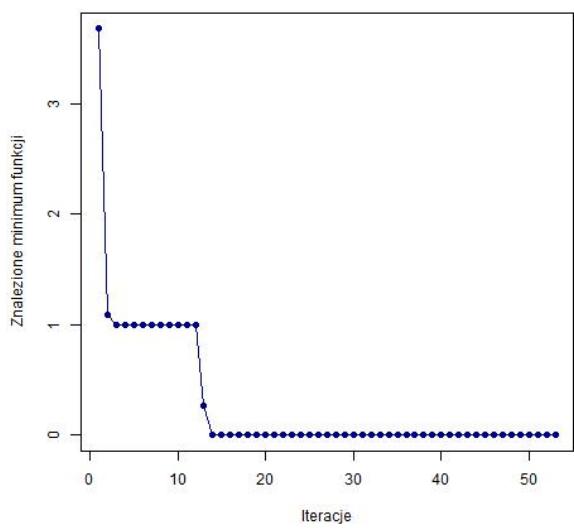
3.5.12. Rastrigin



Rysunek 3.5.103: Iteracja 0, 1 i 5

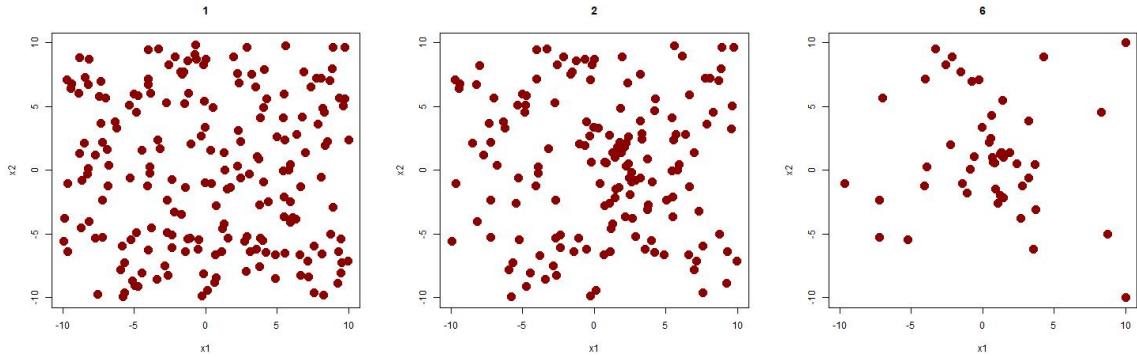


Rysunek 3.5.104: Iteracja 10, 20 i 50

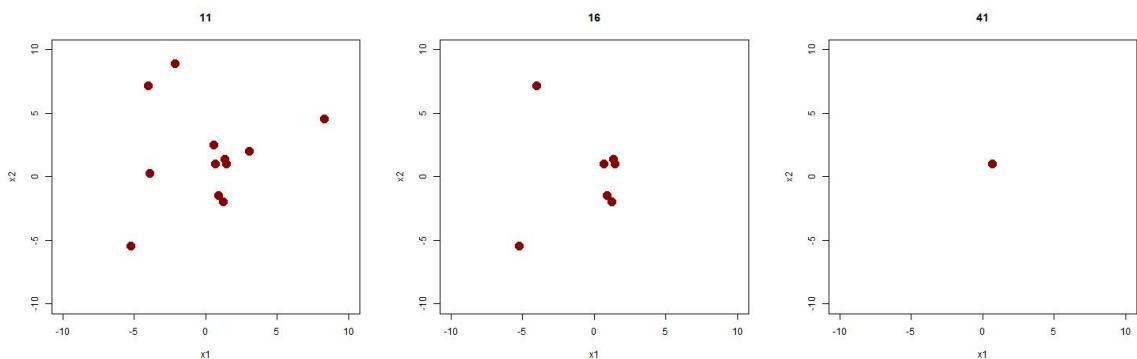


Rysunek 3.5.105: Wartości funkcji z biegiem iteracji

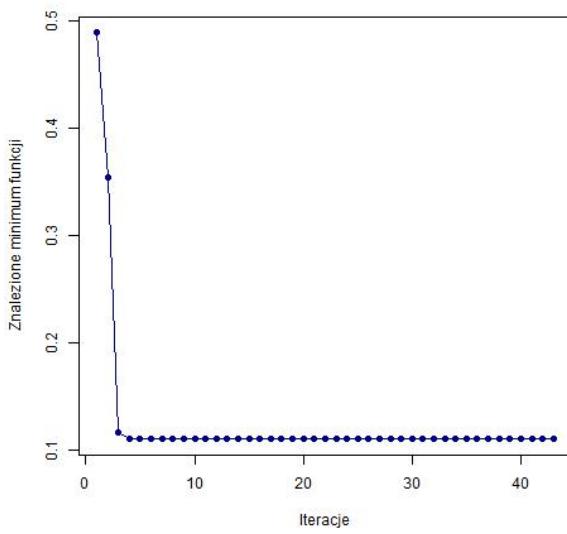
3.5.13. Levy N.13



Rysunek 3.5.106: Iteracja 0, 1 i 5



Rysunek 3.5.107: Iteracja 10, 15 i 40



Rysunek 3.5.108: Wartości funkcji z biegiem iteracji

3.6. Optymalizacja algorytmem genetycznym

Optymalizacja została przeprowadzona algorytmem nietoperza z pakietu *GA*. Argumenty funkcji, które zostały wprowadzone:

- nazwa funkcji
- wielkość populacji: [20,40,70,100,200]
- liczba powtórzeń o identycznym wyniku zatrzymująca pracę programu: 20
- maksymalna liczba iteracji: 100
- prawdopodobieństwo krzyżowania między parami chromosomów: 0,8
- prawdopodobieństwo mutacji między parami chromosomów: 0,1
- liczba określająca procent najlepszych osobników populacji przechodzących do następnego pokolenia: 5
- wektor określający dolne ograniczenia wartości zmiennych
- wektor określający górne ograniczenia wartości zmiennych

Średnie wyniki, tak jak w przypadku wcześniejszych algorytmów zamieszczone w tabelach, 3.6.12 - 3.6.16.

Tabela 3.6.12: Uśrednione wyniki optymalizacji algorytmem genetycznym

Ackley						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe
1	20	0,004	-0,002	0,170	0,16	0,36
2	40	0,001	-0,001	0,024	0,01	0,07
3	70	0,000	0,000	0,001	0,00	0,00
4	100	0,000	0,000	0,001	0,00	0,00
5	200	0,000	0,000	0,000	0,00	0,00
Wartości szukane						

Beale						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe
1	20	2,830	0,442	0,026	0,00	0,04
2	40	2,953	0,483	0,011	0,00	0,02
3	70	2,965	0,491	0,003	0,00	0,00
4	100	2,982	0,493	0,003	0,00	0,00
5	200	2,993	0,497	0,001	0,00	0,00
Wartości szukane						

Goldstein						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczylenie_standardowe
1	20	0,005	-0,982	3,652	3,03	1,63
2	40	0,004	-0,995	3,094	0,03	0,14
3	70	0,001	-0,998	3,028	0,00	0,06
4	100	0,001	-1,000	3,018	0,00	0,03
5	200	0,001	-1,000	3,005	0,00	0,01
Wartości szukane						

Tabela 3.6.13: cd. Uśrednione wyniki optrymalizacji algorytmem genetycznym

Bartels Conn								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,107	-0,285	10,154	509,59	20,84	55,46	0,0688
2	40	0,128	0,004	2,792	18,47	3,95	58,88	0,0988
3	70	-0,005	-0,030	1,349	1,65	1,25	71,68	0,1622
4	100	0,079	-0,046	1,428	4,51	2,10	69,42	0,2084
5	200	-0,001	0,043	1,040	0,01	0,11	79,12	0,4056
	Wartości szukane	0	0	1				

Leon								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,614	0,394	0,175	0,04	0,12	41,18	0,0456
2	40	0,769	0,602	0,076	0,01	0,06	38,06	0,0724
3	70	0,843	0,730	0,051	0,01	0,05	39,24	0,099
4	100	0,854	0,740	0,038	0,00	0,04	32,64	0,1028
5	200	0,921	0,854	0,017	0,00	0,02	29,7	0,1594
	Wartości szukane	1	1	0				

Eggholder								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	-14,709	211,910	-831,726	24317	91,02	53,3	0,0604
2	40	38,193	321,456	-874,517	11337	65,46	53,86	0,0904
3	70	83,472	415,712	-897,569	5282	39,23	62,62	0,1464
4	100	152,118	411,942	-903,743	4158	33,58	50,2	0,1538
5	200	275,515	428,405	-917,263	2660	30,60	51,96	0,2832
	Wartości szukane	512	404	-959				

Tabela 3.6.14: cd. Uśrednione wyniki optymalizacji algorytmem genetycznym

Ventter						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	0,072	0,105	-396,501	47,03	5,96
2	40	-0,005	0,128	-398,444	18,97	4,11
3	70	-0,001	0,000	-399,940	0,03	0,18
4	100	0,001	-0,001	-399,967	0,01	0,09
5	200	-0,002	0,001	-399,984	0,00	0,06
Wartości szukane						
		0	0	-400		

Matyas						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	-0,024	-0,030	0,003	0,00	0,01
2	40	0,009	0,007	0,000	0,00	0,00
3	70	0,000	0,000	0,000	0,00	0,00
4	100	0,000	0,001	0,000	0,00	0,00
5	200	0,000	0,000	0,000	0,00	0,00
Wartości szukane						
		0	0	0		

Zirilli						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	-1,008	0,003	-0,341	0,00	0,02
2	40	-1,030	0,006	-0,348	0,00	0,01
3	70	-1,048	0,001	-0,351	0,00	0,00
4	100	-1,046	-0,007	-0,351	0,00	0,00
5	200	-1,044	-0,002	-0,351	0,00	0,00
Wartości szukane						
		-1,04	0	-0,35		

Tabela 3.6.15: cd. Uśrednione wyniki optymalizacji algorytmem genetycznym

Drop Wave								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,030	0,033	-0,944	0,00	0,02	53,66	0,063
2	40	-0,062	-0,004	-0,971	0,00	0,03	60,7	0,1032
3	70	0,006	-0,035	-0,982	0,00	0,03	67,46	0,1676
4	100	0,011	0,006	-0,988	0,00	0,02	65,06	0,2182
5	200	0,000	0,000	-0,999	0,00	0,00	69,54	0,3882
		Wartości szukane	0	0	-1			

Levy N.13								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	1,005	1,004	0,034	0,00	0,06	52,24	0,0666
2	40	1,000	0,996	0,009	0,00	0,03	56,08	0,096
3	70	1,000	1,005	0,001	0,00	0,00	73,98	0,1746
4	100	1,000	1,002	0,001	0,00	0,00	73,38	0,2122
5	200	1,000	0,999	0,000	0,00	0,00	76,68	0,3872
		Wartości szukane	1	1	0			

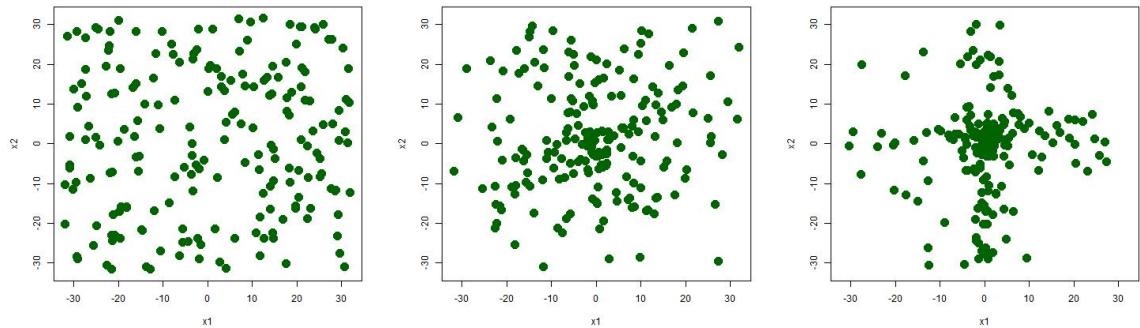
Rastrigin								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,000	-0,060	0,194	0,22	0,43	63,68	0,0812
2	40	0,000	0,000	0,001	0,00	0,00	76,84	0,1238
3	70	0,000	0,000	0,001	0,00	0,00	74,7	0,1766
4	100	0,000	0,000	0,000	0,00	0,00	76,54	0,2304
5	200	0,000	0,000	0,000	0,00	0,00	82,12	0,4158
		Wartości szukane	0	0	0			

Tabela 3.6.16: cd. Uśrednione wyniki optymalizacji algorytmem genetycznym

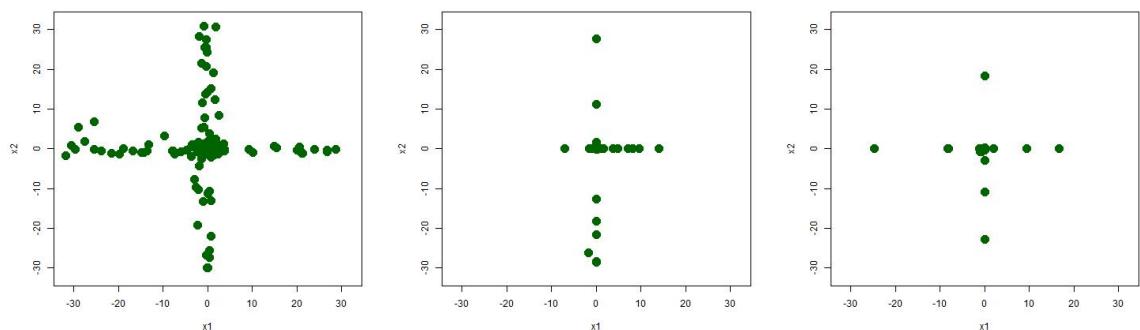
Easom								
Nr_sredniej	Liczebnosc.pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	3,309	2,811	-0,536	0,40	0,43	49,28	0,0568
2	40	2,961	2,945	-0,748	0,21	0,38	62,26	0,1022
3	70	3,138	3,093	-0,944	0,03	0,18	69,1	0,1556
4	100	3,122	3,127	-0,977	0,01	0,12	75,76	0,2184
5	200	3,145	3,136	-0,996	0,00	0,02	78,96	0,3994
	Wartości szukane	3,14	3,14	-1				

Proces optymalizacji GA dla wszystkich funkcji został przedstawiony graficznie na rysunkach 3.6.109 - 3.6.147.

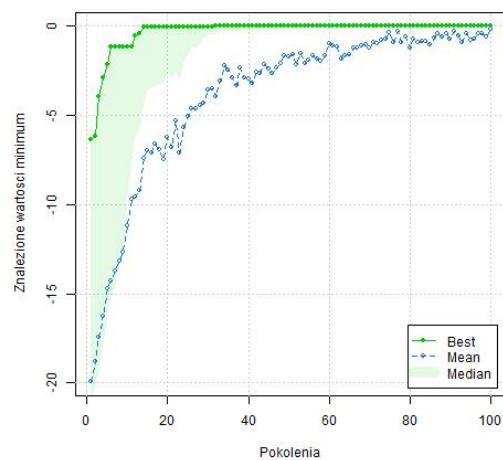
3.6.1. Ackley



Rysunek 3.6.109: Iteracja 0, 3 i 6

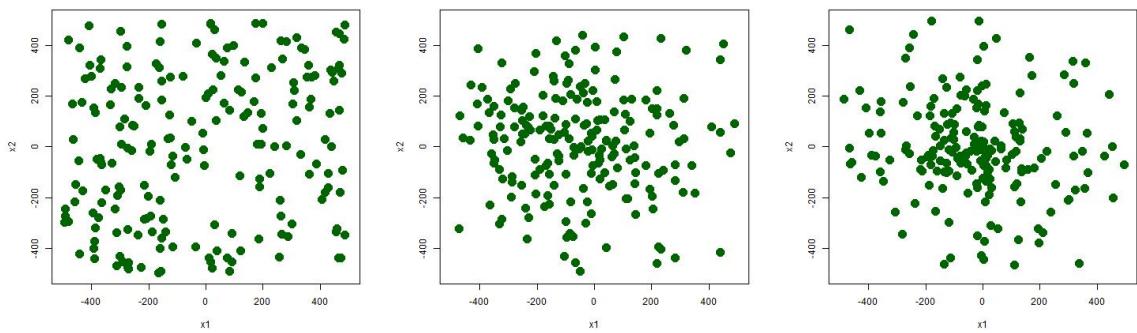


Rysunek 3.6.110: Iteracja 12, 50 i 68

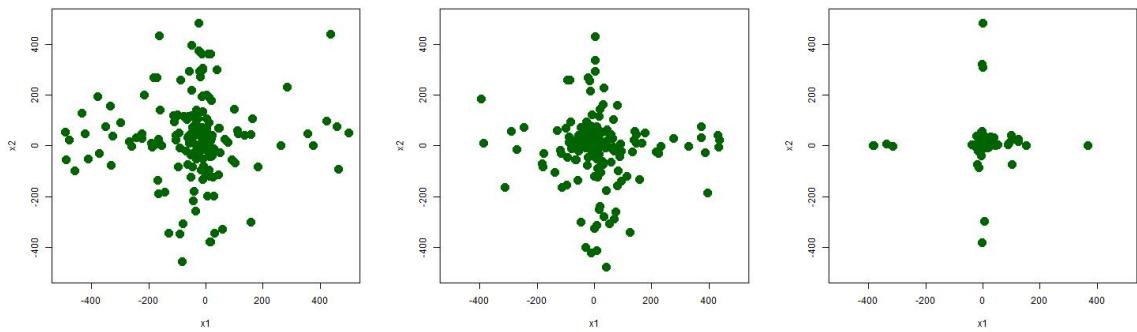


Rysunek 3.6.111: Wartości funkcji z biegiem iteracji

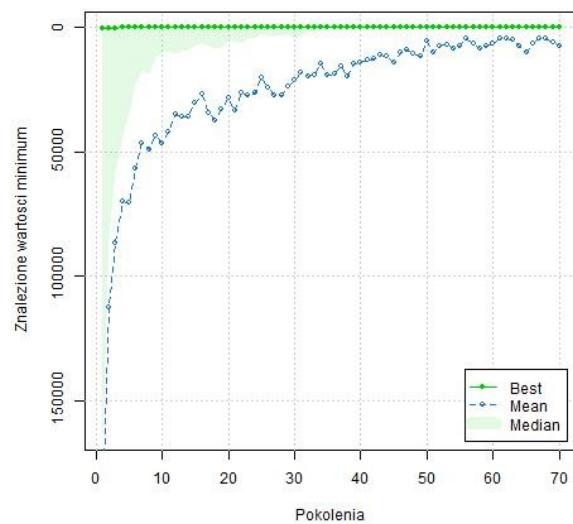
3.6.2. Bartels



Rysunek 3.6.112: Iteracja 0, 3 i 5

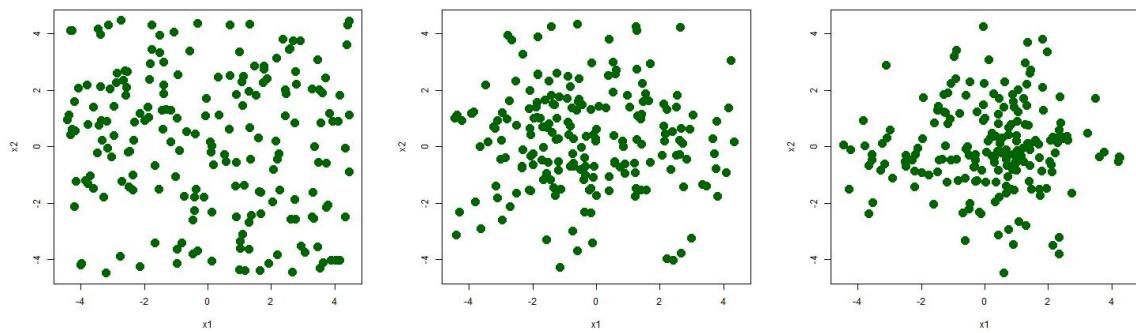


Rysunek 3.6.113: Iteracja 10, 20 i 70

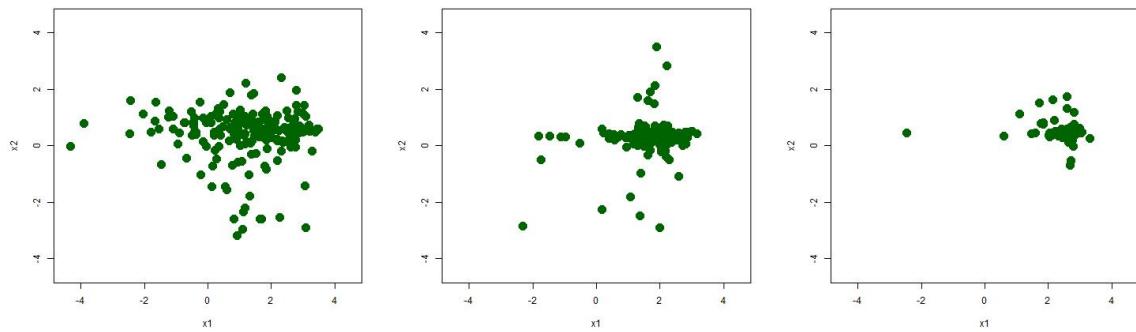


Rysunek 3.6.114: Wartości funkcji z biegiem iteracji

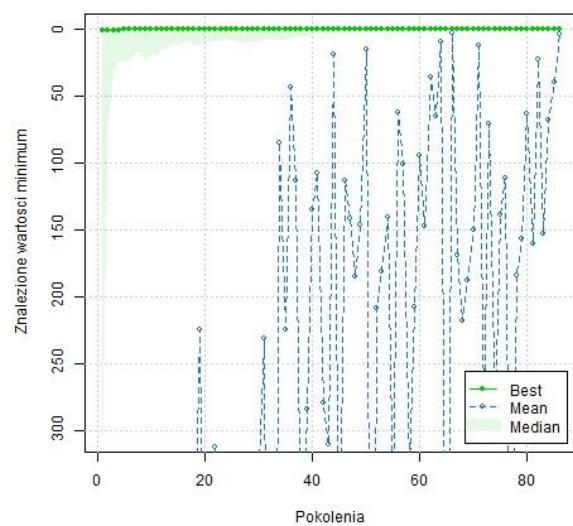
3.6.3. Beale



Rysunek 3.6.115: Iteracja 0, 3 i 10

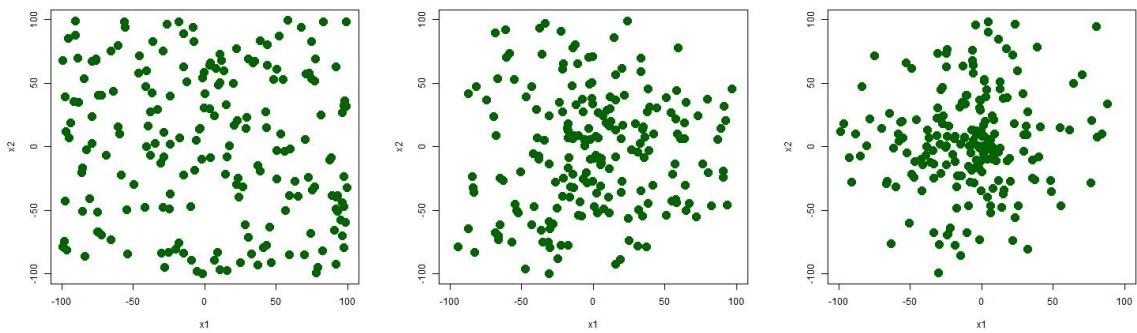


Rysunek 3.6.116: Iteracja 34, 60 i 86

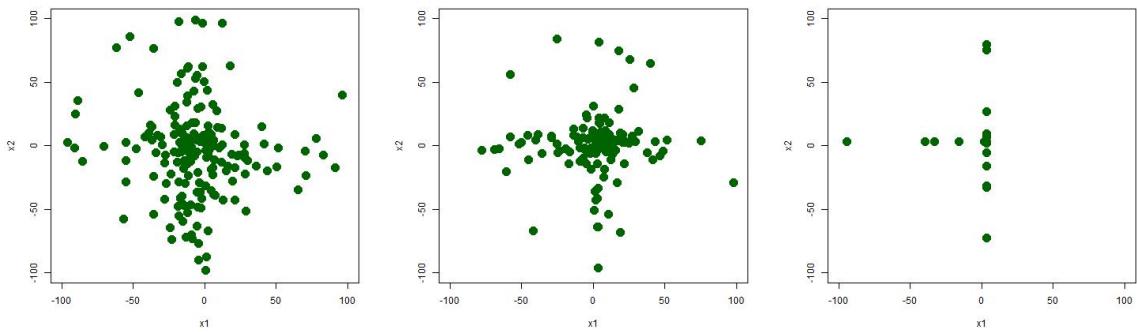


Rysunek 3.6.117: Wartości funkcji z biegiem iteracji

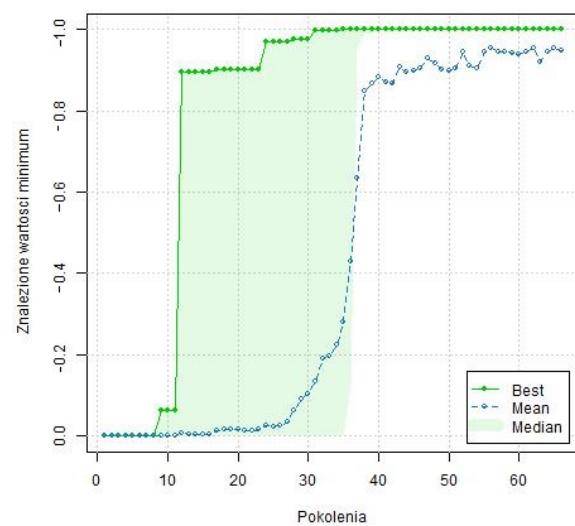
3.6.4. Easom



Rysunek 3.6.118: Iteracja 0, 3 i 10

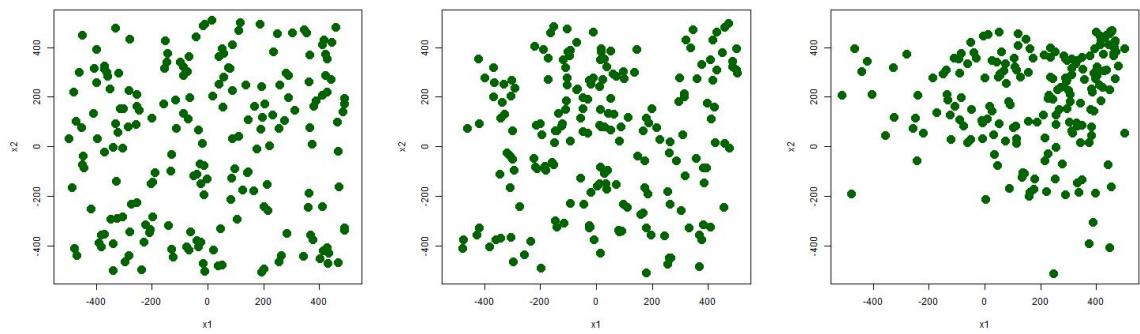


Rysunek 3.6.119: Iteracja 20, 32 i 63

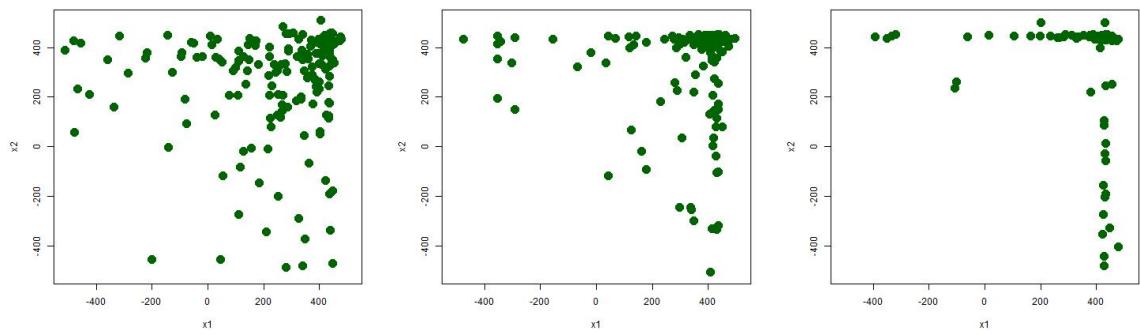


Rysunek 3.6.120: Wartości funkcji z biegiem iteracji

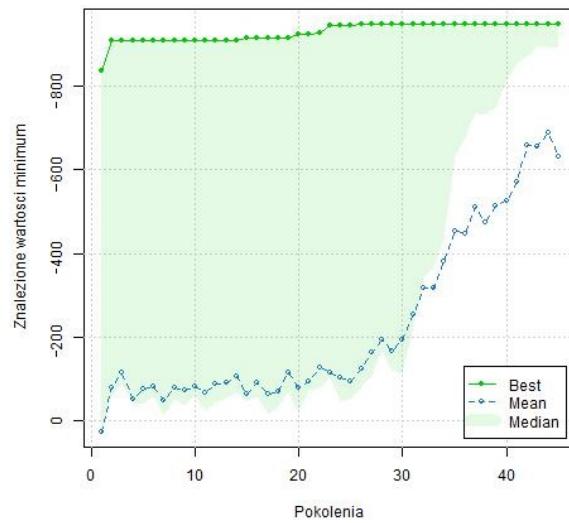
3.6.5. Eggholder



Rysunek 3.6.121: Iteracja 0, 3 i 20

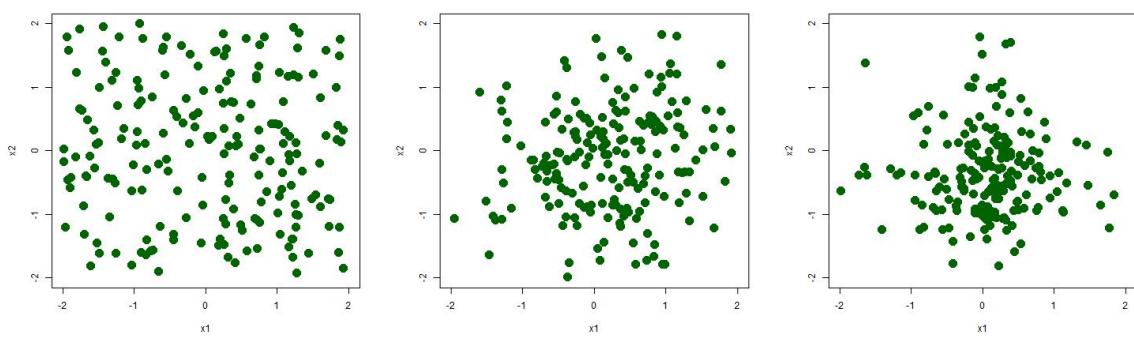


Rysunek 3.6.122: Iteracja 28, 34 i 45

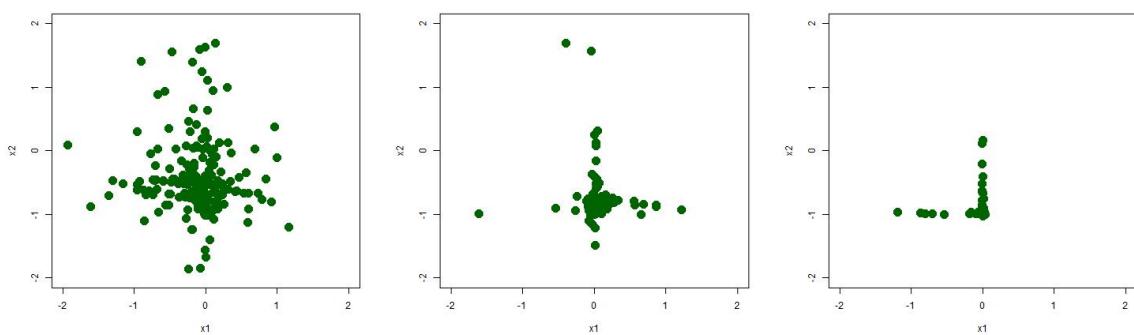


Rysunek 3.6.123: Wartości funkcji z biegiem iteracji

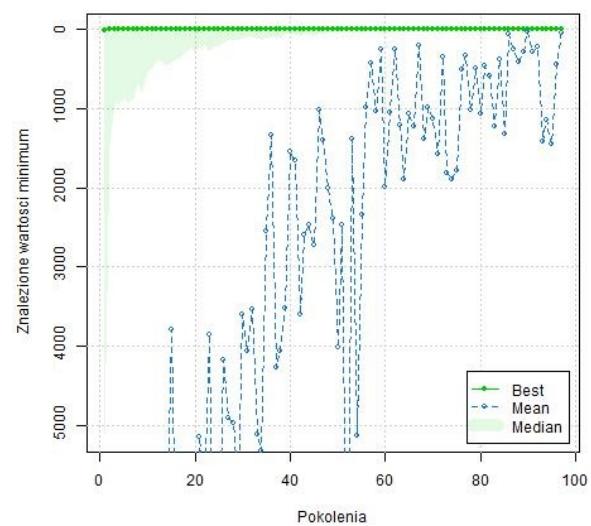
3.6.6. Goldstein



Rysunek 3.6.124: Iteracja 0, 3 i 10

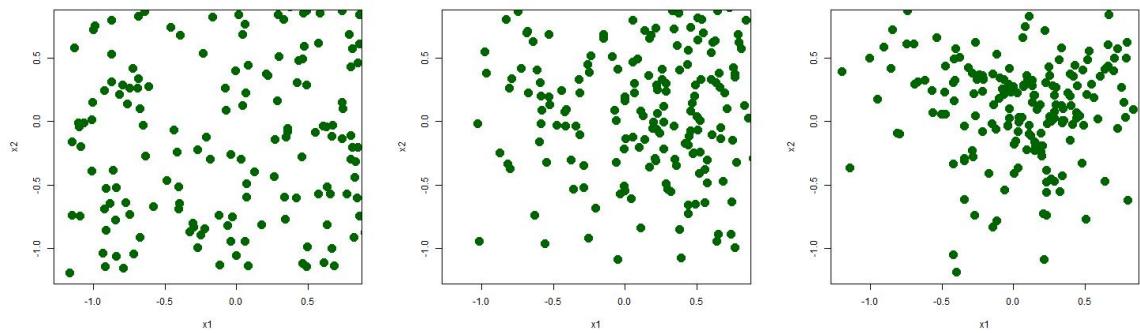


Rysunek 3.6.125: Iteracja 25, 60 i 97

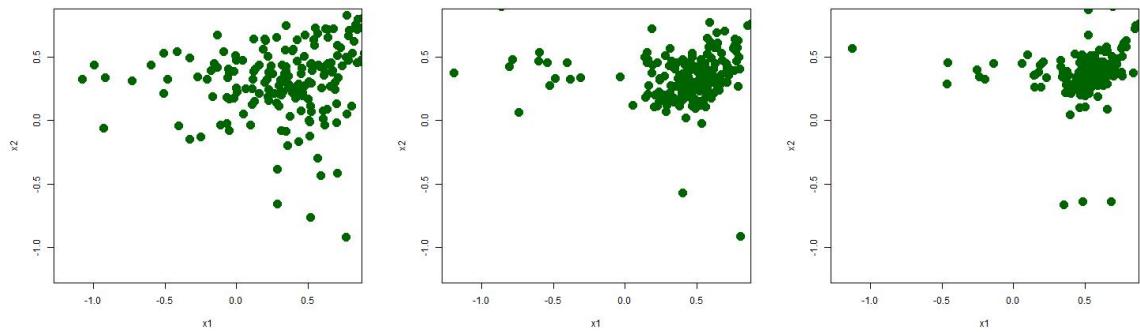


Rysunek 3.6.126: Wartości funkcji z biegiem iteracji

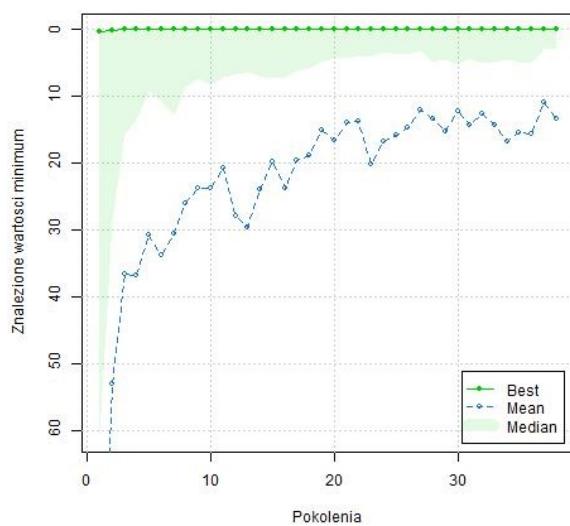
3.6.7. Leon



Rysunek 3.6.127: Iteracja 0, 3 i 10

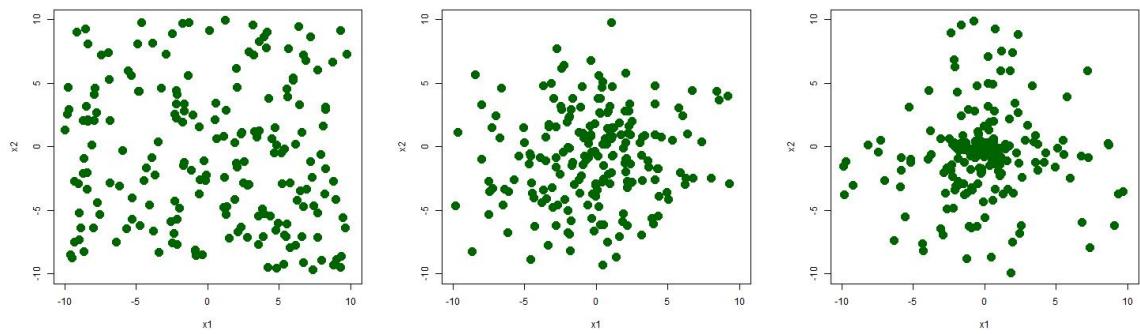


Rysunek 3.6.128: Iteracja 30, 45 i 49

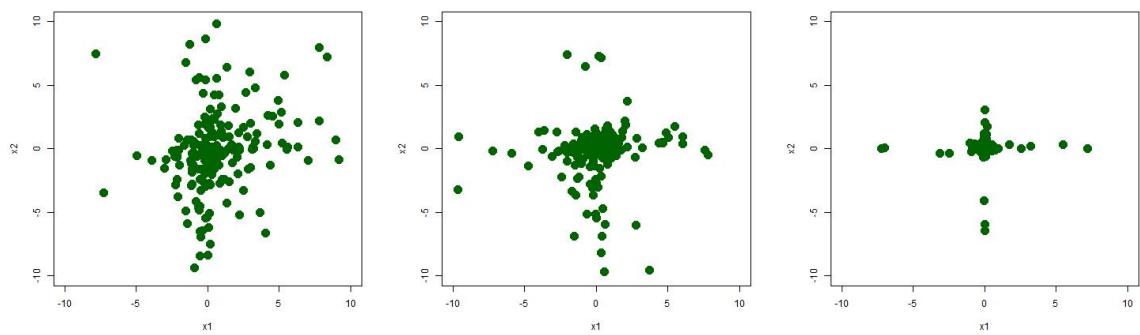


Rysunek 3.6.129: Wartości funkcji z biegiem iteracji

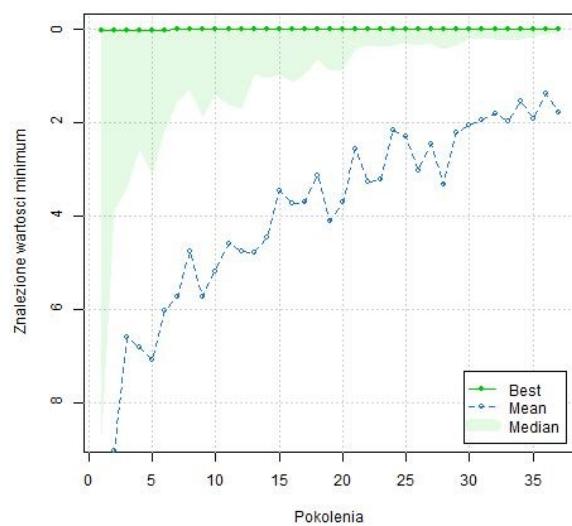
3.6.8. Matyas



Rysunek 3.6.130: Iteracja 0, 3 i 7

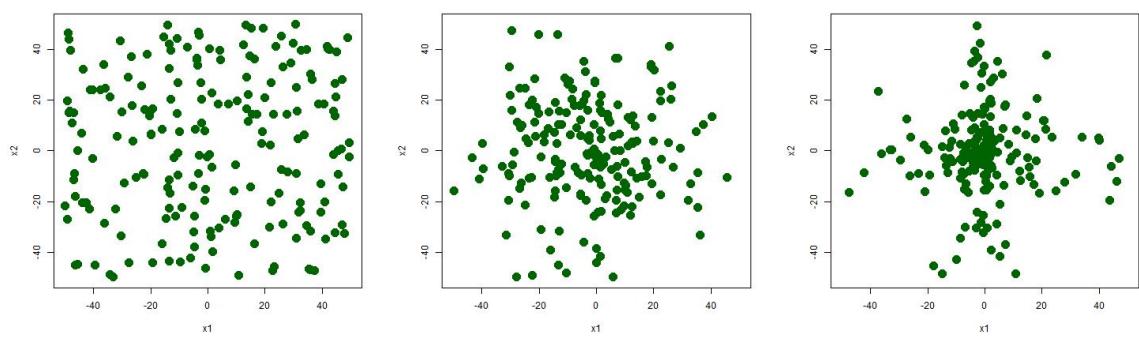


Rysunek 3.6.131: Iteracja 15, 25 i 66

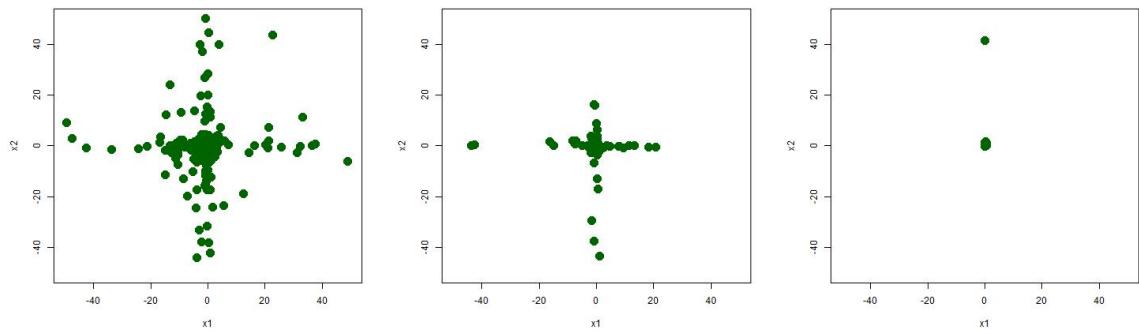


Rysunek 3.6.132: Wartości funkcji z biegiem iteracji

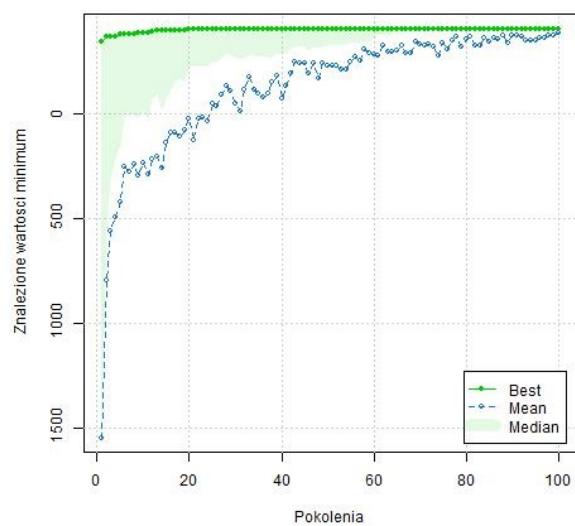
3.6.9. Venter



Rysunek 3.6.133: Iteracja 0, 3 i 7

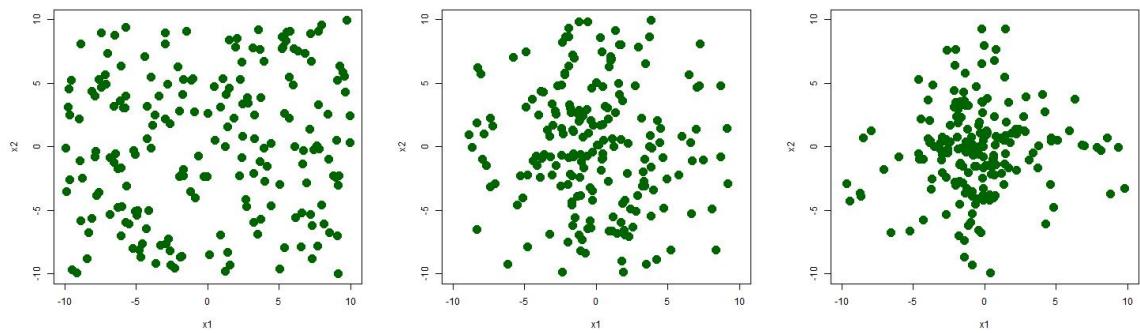


Rysunek 3.6.134: Iteracja 20, 60 i 100

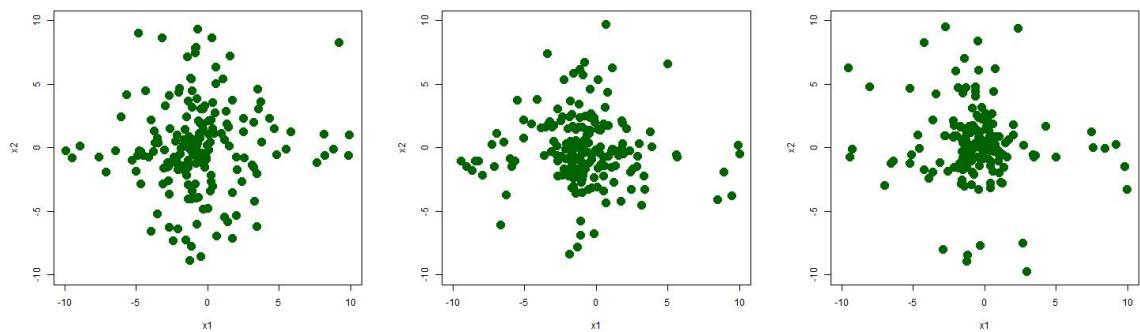


Rysunek 3.6.135: Wartości funkcji z biegiem iteracji

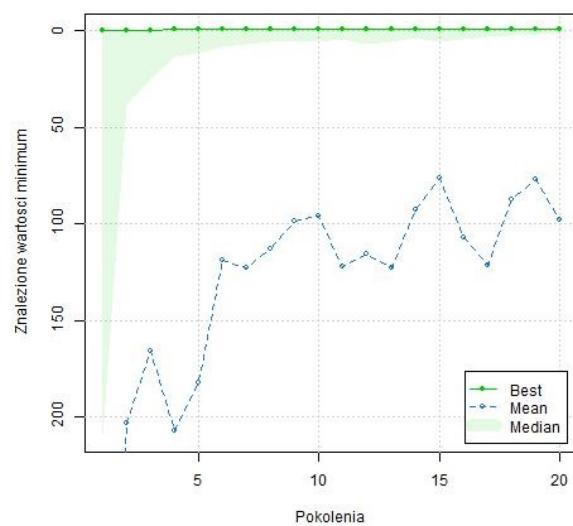
3.6.10. Zirilli



Rysunek 3.6.136: Iteracja 0, 3 i 7

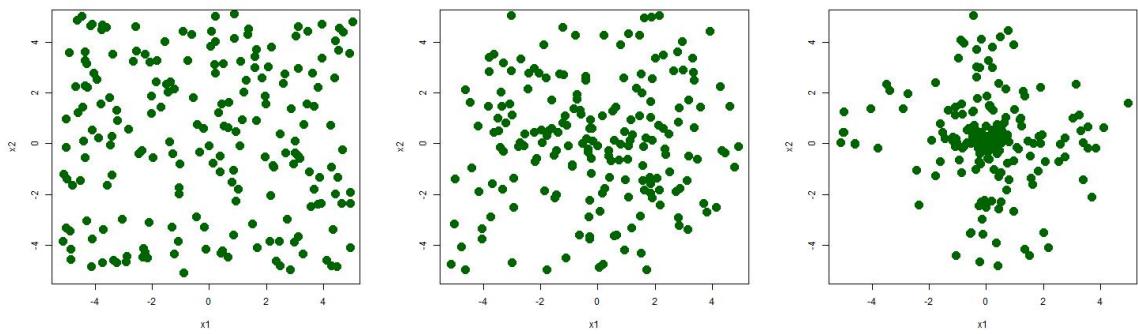


Rysunek 3.6.137: Iteracja 12, 17 i 20

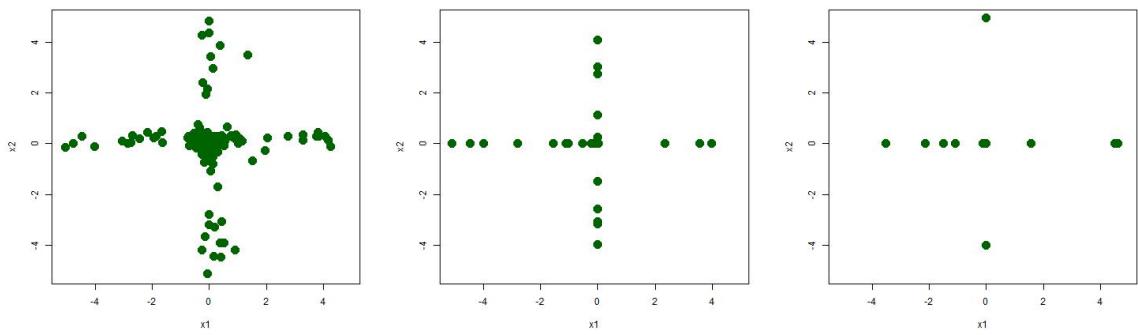


Rysunek 3.6.138: Wartości funkcji z biegiem iteracji

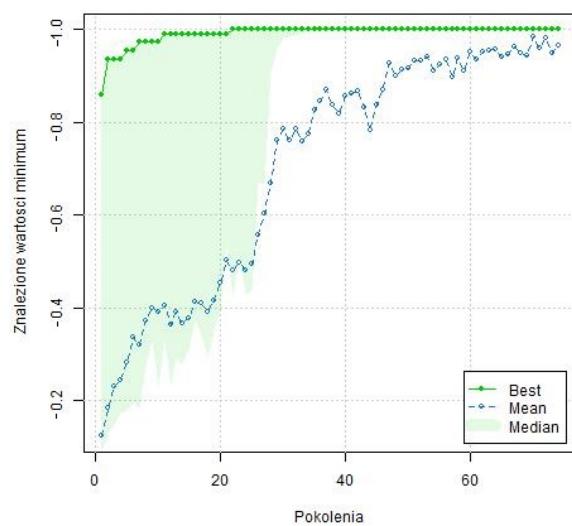
3.6.11. Drop Wave



Rysunek 3.6.139: Iteracja 0, 3 i 7

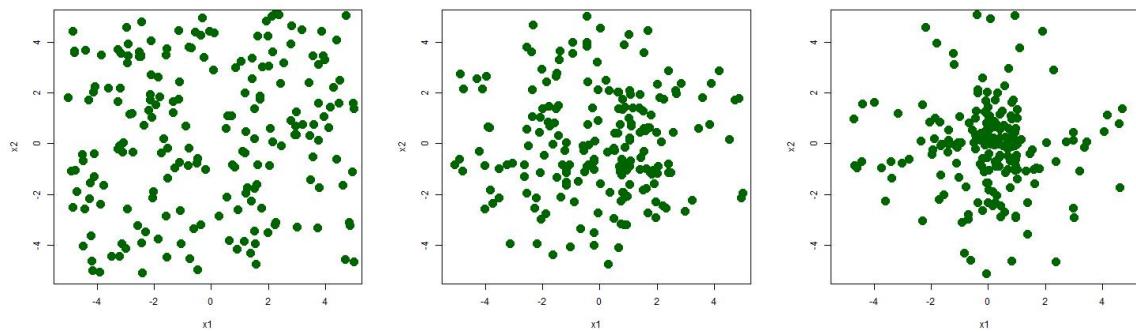


Rysunek 3.6.140: Iteracja 24, 50 i 74

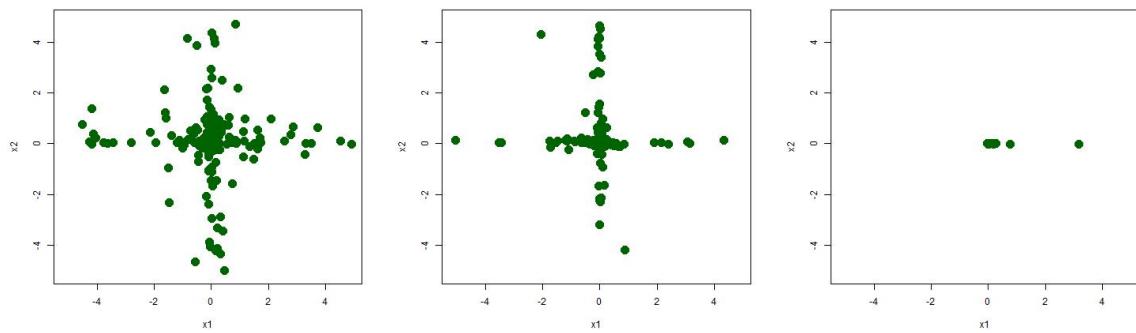


Rysunek 3.6.141: Wartości funkcji z biegiem iteracji

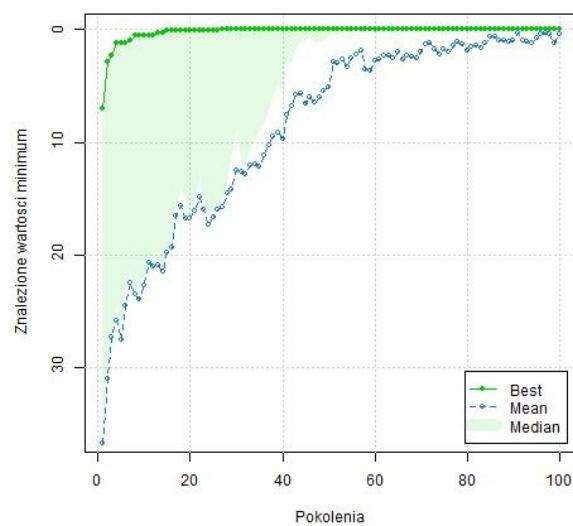
3.6.12. Rastrigin



Rysunek 3.6.142: Iteracja 0, 3 i 7

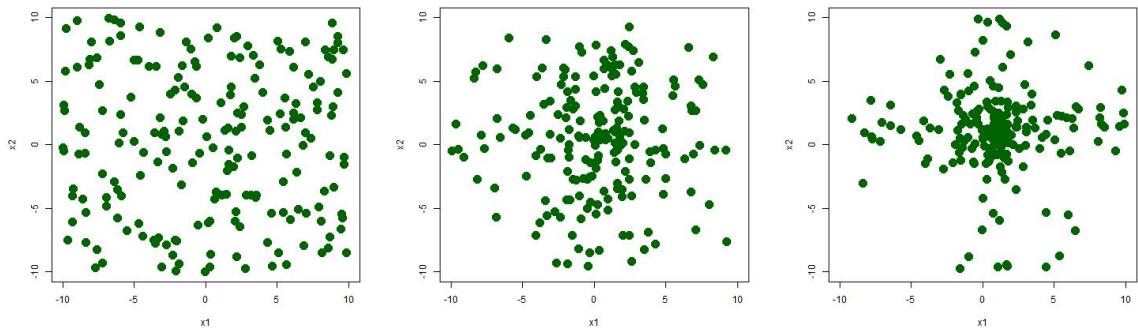


Rysunek 3.6.143: Iteracja 20, 40 i 100

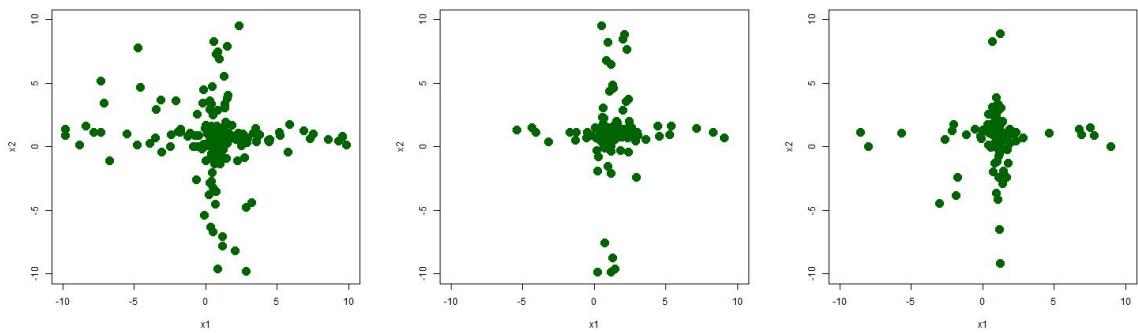


Rysunek 3.6.144: Wartości funkcji z biegiem iteracji

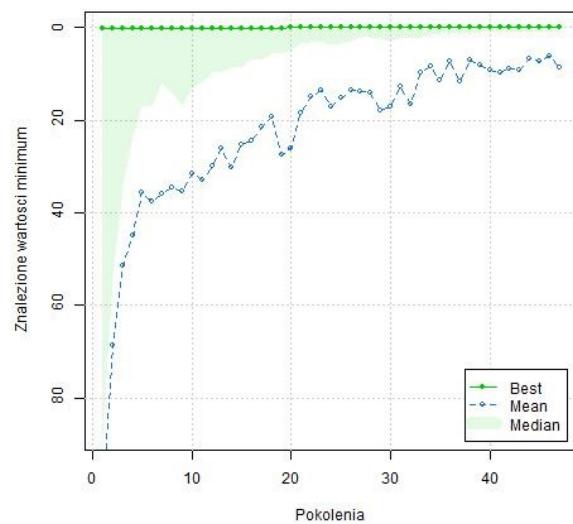
3.6.13. Levy N.13



Rysunek 3.6.145: Iteracja 0, 3 i 7



Rysunek 3.6.146: Iteracja 20, 37 i 47



Rysunek 3.6.147: Wartości funkcji z biegiem iteracji

3.7. Optymalizacja algorytmem ewolucji różnicowej

Argumenty funkcji *DEOptim*, które zostały wprowadzone podczas eksperymentu z wykorzystaniem ewolucji różnicowej:

- nazwa funkcji
- strategia algorytmu: DE / local-to-best / 1 / bin
- wielkość populacji: [20,40,70,100,200]
- maksymalna liczba iteracji: 100
- prawdopodobieństwo krzyżowania: 0,5
- współczynnik wagi różnicowej: 0,8
- prędkość adaptacji po krzyżowaniu: 0
- wektor określający dolne ograniczenia wartości zmiennych
- wektor określający górne ograniczenia wartości zmiennych
- liczba określająca tzw. ziarno dla generatora liczb pseudolosowych argument stosowany w celu uzyskania powtarzalności otrzymywanych wyników: rand(0:1000)

Uśrednione wyniki optymalizacji zawarto w tabelach 3.7.17 - 3.7.21.

Tabela 3.7.17: Uśrednione wyniki optymalizacji algorytmem ewolucji różnicowej

Ackley						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe
1	20	0,000	0,000	0,000	0,00	100
2	40	0,000	0,000	0,000	0,00	100
3	70	0,000	0,000	0,000	0,00	100
4	100	0,000	0,000	0,000	0,00	100
5	200	0,000	0,000	0,000	0,00	100
	Wartości szukane	0	0	0		

Beale						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe
1	20	2,850	0,514	0,015	0,01	100
2	40	3,000	0,500	0,000	0,00	100
3	70	3,000	0,500	0,000	0,00	100
4	100	3,000	0,500	0,000	0,00	100
5	200	3,000	0,500	0,000	0,00	100
	Wartości szukane	3	0,5	0		

Goldstein						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe
1	20	0,000	-1,000	3,000	0,00	0,00
2	40	0,000	-1,000	3,000	0,00	0,00
3	70	0,000	-1,000	3,000	0,00	0,00
4	100	0,000	-1,000	3,000	0,00	0,00
5	200	0,000	-1,000	3,000	0,00	0,00
	Wartości szukane	0	-1	3		

Tabela 3.7.18: cd. Uśrednione wyniki optymalizacji algorytmem ewolucji różnicowej

Bartels Conn						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	0,000	0,000	1,000	0,00	0,00
2	40	0,000	0,000	1,000	0,00	0,00
3	70	0,000	0,000	1,000	0,00	0,00
4	100	0,000	0,000	1,000	0,00	0,00
5	200	0,000	0,000	1,000	0,00	0,00
Wartości szukane						
		0	0	1		

Leon						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	1,000	1,000	0,000	0,00	0,00
2	40	1,000	1,000	0,000	0,00	0,00
3	70	1,000	1,000	0,000	0,00	0,00
4	100	1,000	1,000	0,000	0,00	0,00
5	200	1,000	1,000	0,000	0,00	0,00
Wartości szukane						
		1	1	0		

Eggholder						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe
1	20	197,306	398,599	-927,745	2015	32,54
2	40	444,374	427,229	-947,299	337	14,27
3	70	496,297	418,435	-955,498	27	3,85
4	100	497,511	417,136	-956,717	8	1,81
5	200	494,656	420,648	-957,273	5	1,27
Wartości szukane						
		512	404	-959		

Tabela 3.7.19: cd. Uśrednione wyniki optymalizacji algorytmem ewolucji różnicowej

Venter								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Zmaleznie_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,000	0,000	-400,000	0,00	0,00	99	0,01
2	40	0,000	0,000	-400,000	0,00	0,00	100	0,02
3	70	0,000	0,000	-400,000	0,00	0,00	99	0,02
4	100	0,000	0,000	-400,000	0,00	0,00	100	0,03
5	200	0,000	0,000	-400,000	0,00	0,00	100	0,07
	Wartości szukane	0	0	-400				

Matyas								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Zmaleznie_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,000	0,000	0,000	0,00	0,00	100	0,01
2	40	0,000	0,000	0,000	0,00	0,00	100	0,01
3	70	0,000	0,000	0,000	0,00	0,00	100	0,02
4	100	0,000	0,000	0,000	0,00	0,00	100	0,03
5	200	0,000	0,000	0,000	0,00	0,00	100	0,05
	Wartości szukane	0	0	0				

Zirilli								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Zmaleznie_minimum	MSE_minimum	Odczytanie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-1,047	0,000	-0,352	0,00	0,00	100	0,01
2	40	-1,047	0,000	-0,352	0,00	0,00	100	0,01
3	70	-1,047	0,000	-0,352	0,00	0,00	100	0,02
4	100	-1,047	0,000	-0,352	0,00	0,00	100	0,03
5	200	-1,047	0,000	-0,352	0,00	0,00	100	0,06
	Wartości szukane	-1,04	0	-0,35				

Tabela 3.7.20: cd. Uśrednione wyniki optymalizacji algorytmem ewolucji różnicowej

Drop Wave						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe
1	20	0,002	0,000	-0,997	0,00	0,00
2	40	0,000	0,000	-1,000	0,00	0,00
3	70	0,000	0,000	-1,000	0,00	0,02
4	100	0,000	0,000	-1,000	0,00	0,03
5	200	0,000	0,000	-1,000	0,00	0,06
	Wartości szukane	0	0	-1		

Levy N.13						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe
1	20	1,000	1,000	0,000	0,00	0,00
2	40	1,000	1,000	0,000	0,00	0,01
3	70	1,000	1,000	0,000	0,00	0,02
4	100	1,000	1,000	0,000	0,00	0,03
5	200	1,000	1,000	0,000	0,00	0,07
	Wartości szukane	1	1	0		

Rastrigin						
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odczytanie_standardowe
1	20	-0,020	0,001	0,034	0,02	0,14
2	40	0,000	0,000	0,000	0,00	0,02
3	70	0,000	0,000	0,000	0,00	0,02
4	100	0,000	0,000	0,000	0,00	0,03
5	200	0,000	0,000	0,000	0,00	0,06
	Wartości szukane	0	0	0		

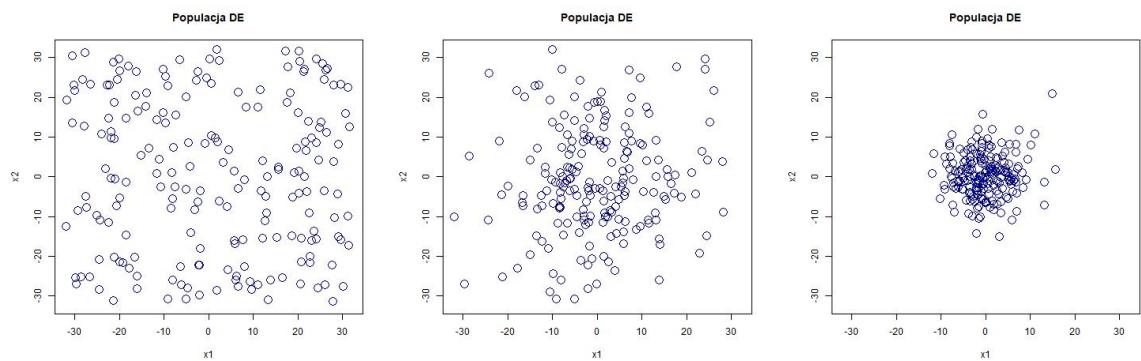
Tabela 3.7.21: cd. Uśrednione wyniki optymalizacji algorytmem ewolucji różnicowej

Easom						
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Zmaleznie_minimum	MSE_minimum	Odcylenie_standardowe
1	20	3,142	3,142	-1,000	0,00	0,00
2	40	3,142	3,142	-1,000	0,00	0,01
3	70	3,142	3,142	-1,000	0,00	0,02
4	100	3,142	3,142	-1,000	0,00	0,03
5	200	3,142	3,142	-1,000	0,00	0,06
	Wartosci szukane	3,14	3,14	-1		

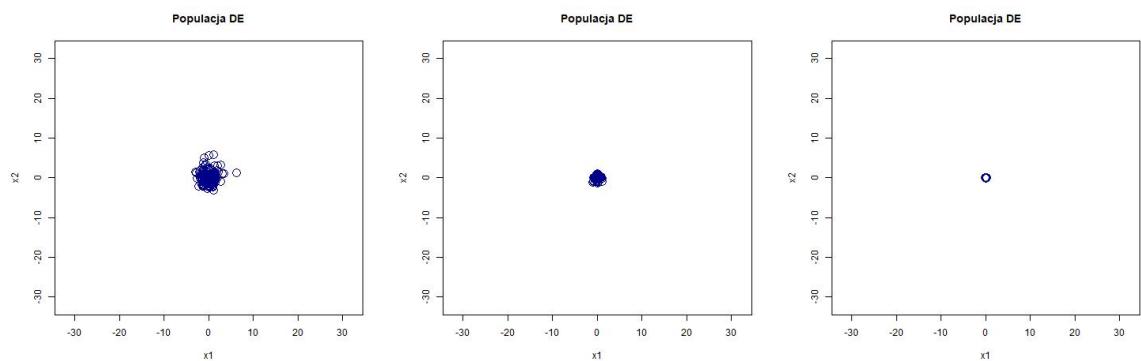
Optymalizacja DE (populacja 200) została przedstawiona także na rysunkach 3.7.148

- 3.7.186.

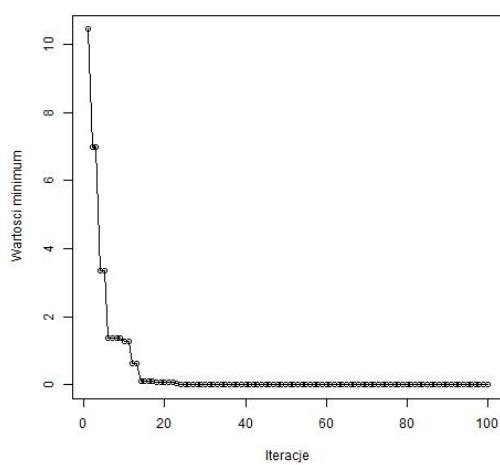
3.7.1. Ackley



Rysunek 3.7.148: Iteracja 0, 4 i 8

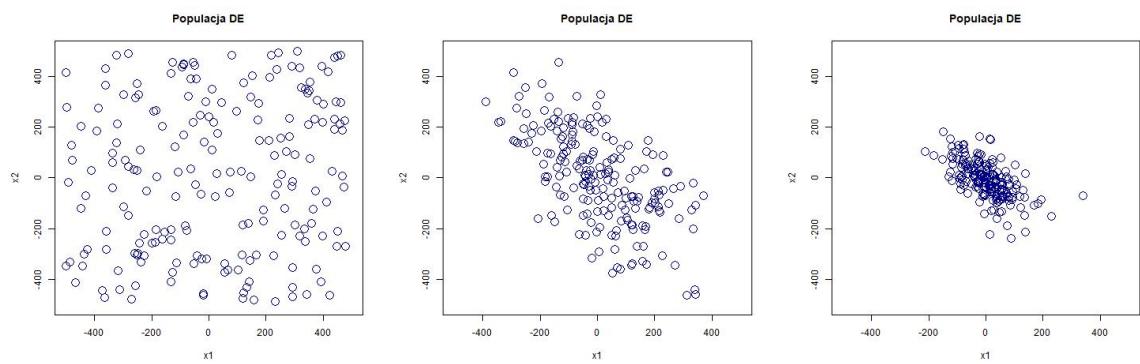


Rysunek 3.7.149: Iteracja 14, 20 i 30

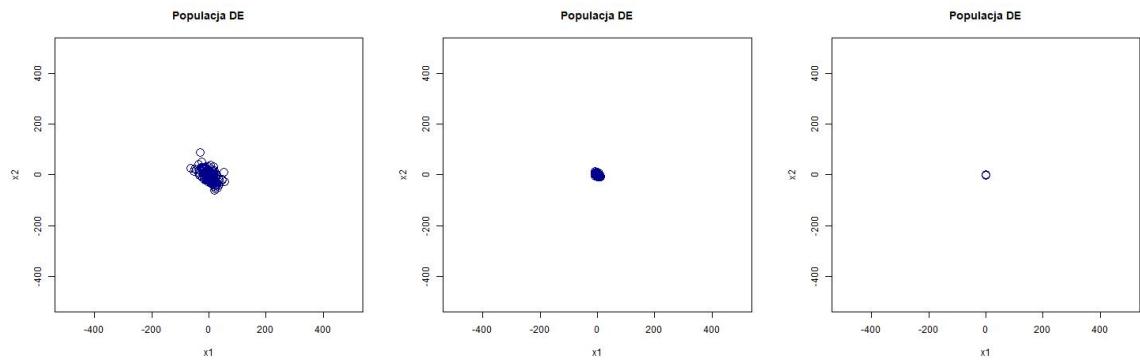


Rysunek 3.7.150: Wartości funkcji z biegiem iteracji

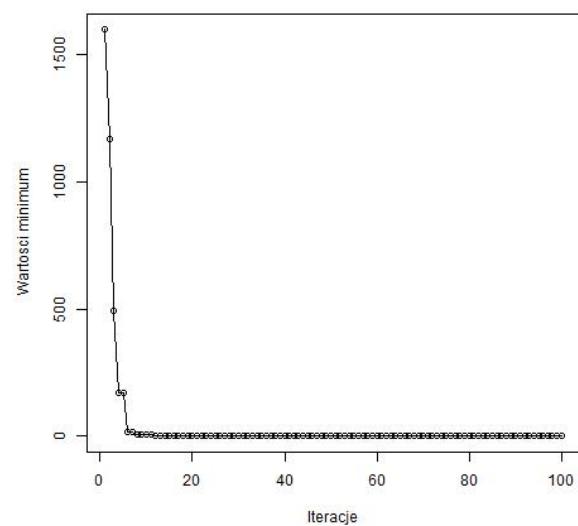
3.7.2. Bartels



Rysunek 3.7.151: Iteracja 0, 4 i 8

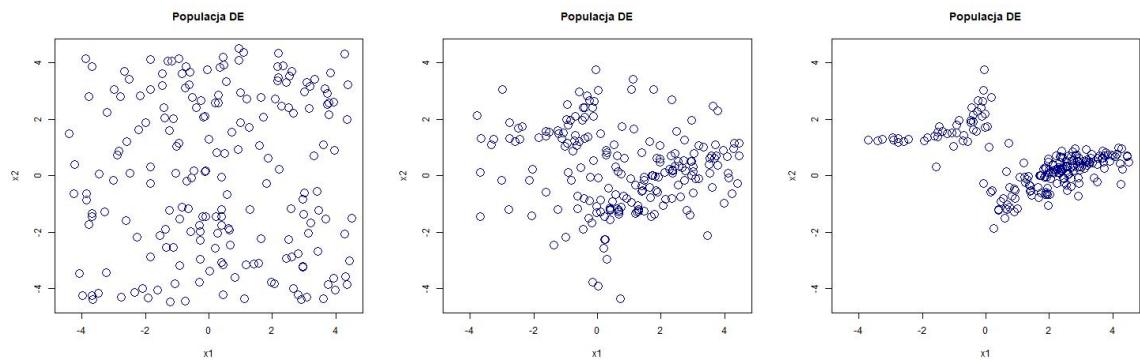


Rysunek 3.7.152: Iteracja 14, 20 i 30

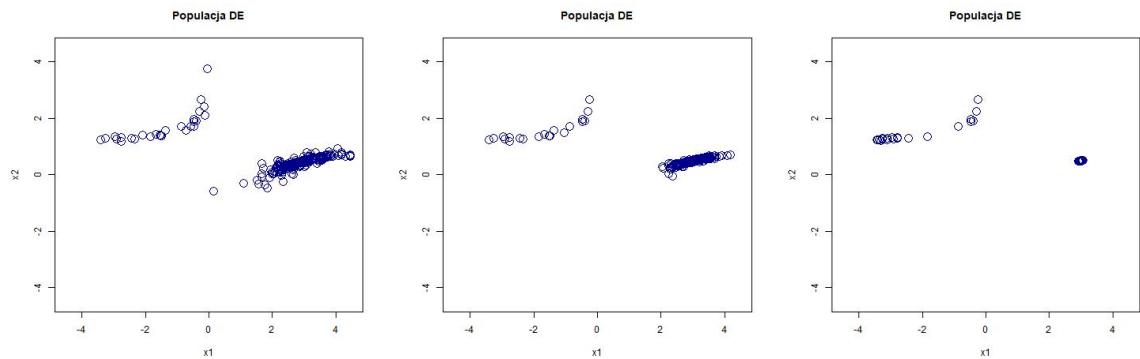


Rysunek 3.7.153: Wartości funkcji z biegiem iteracji

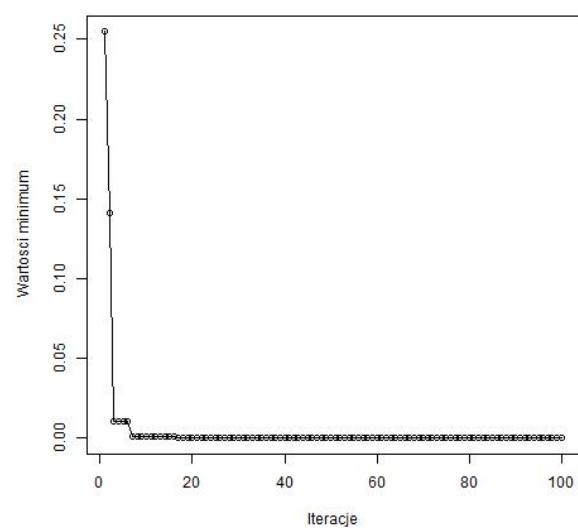
3.7.3. Beale



Rysunek 3.7.154: Iteracja 0, 4 i 8

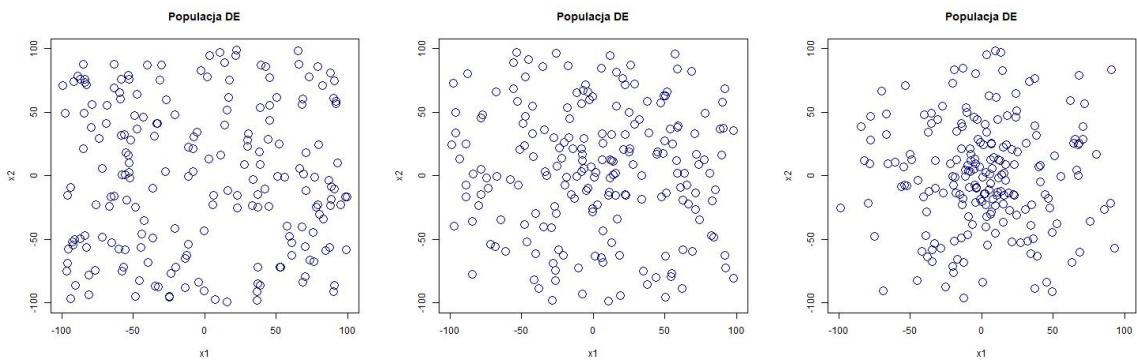


Rysunek 3.7.155: Iteracja 14, 20 i 40

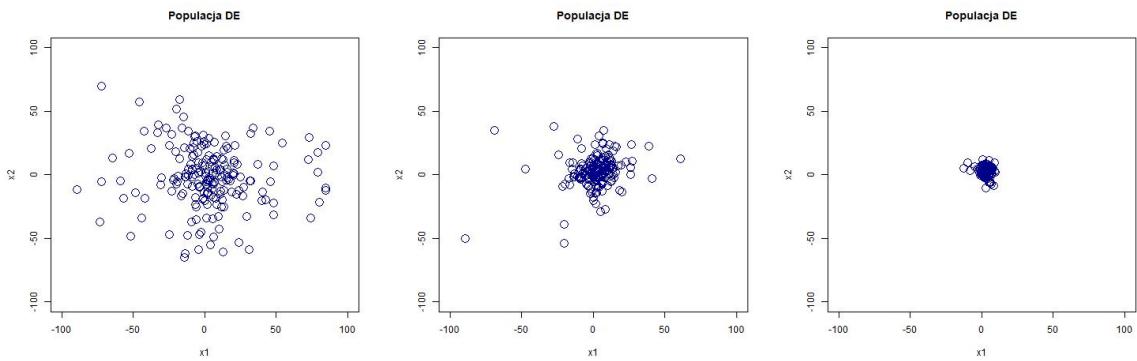


Rysunek 3.7.156: Wartości funkcji z biegiem iteracji

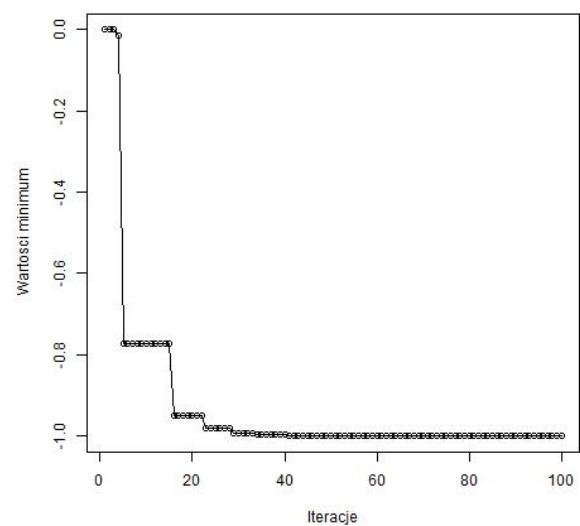
3.7.4. Easom



Rysunek 3.7.157: Iteracja 0, 4 i 8

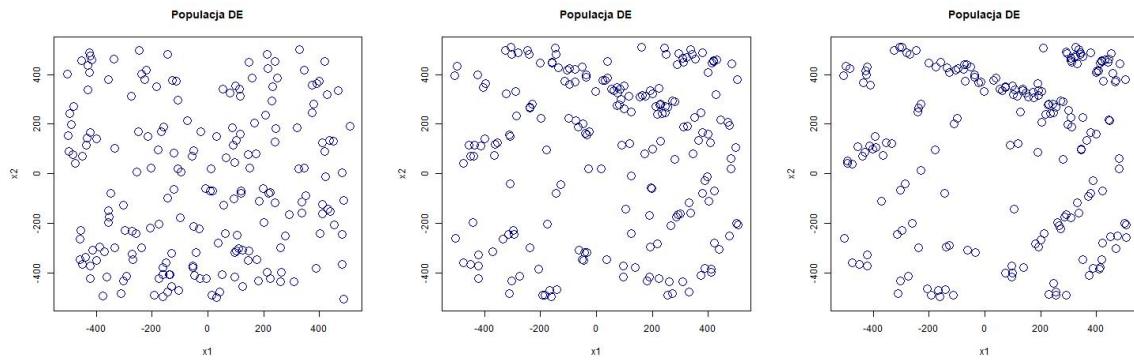


Rysunek 3.7.158: Iteracja 14, 20 i 30

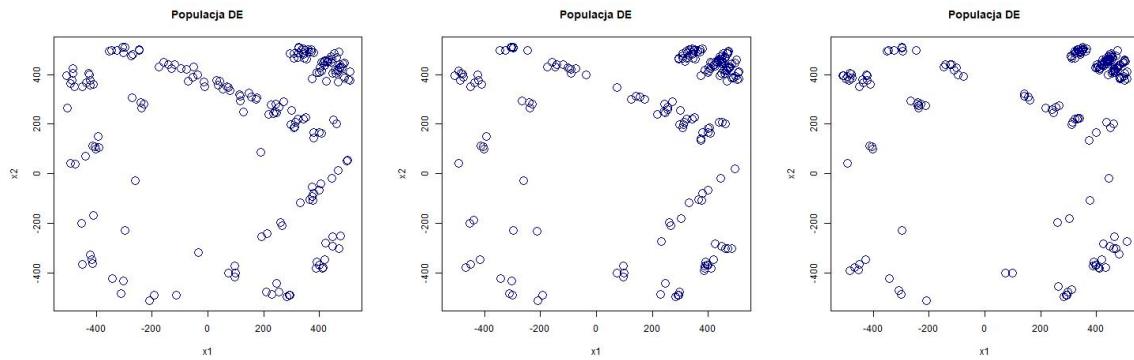


Rysunek 3.7.159: Wartości funkcji z biegiem iteracji

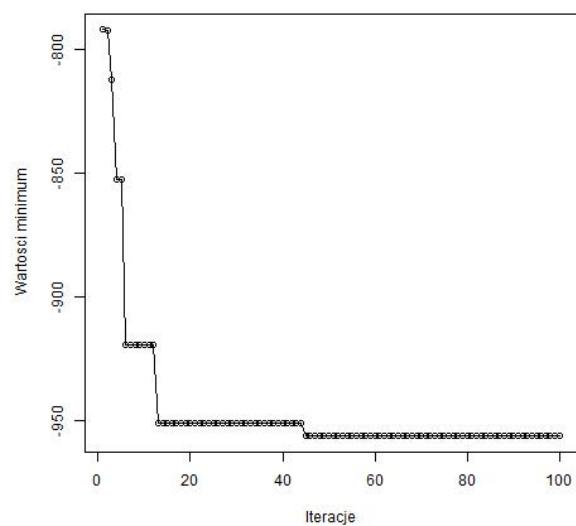
3.7.5. Eggholder



Rysunek 3.7.160: Iteracja 0, 4 i 8

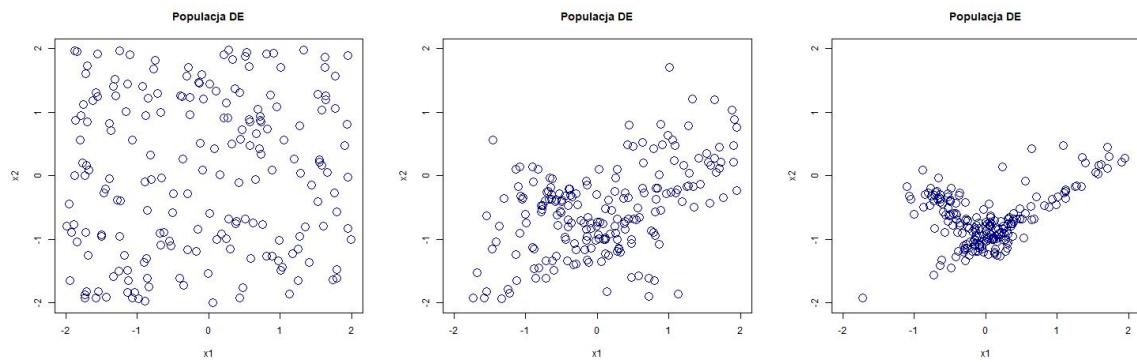


Rysunek 3.7.161: Iteracja 14, 20 i 30

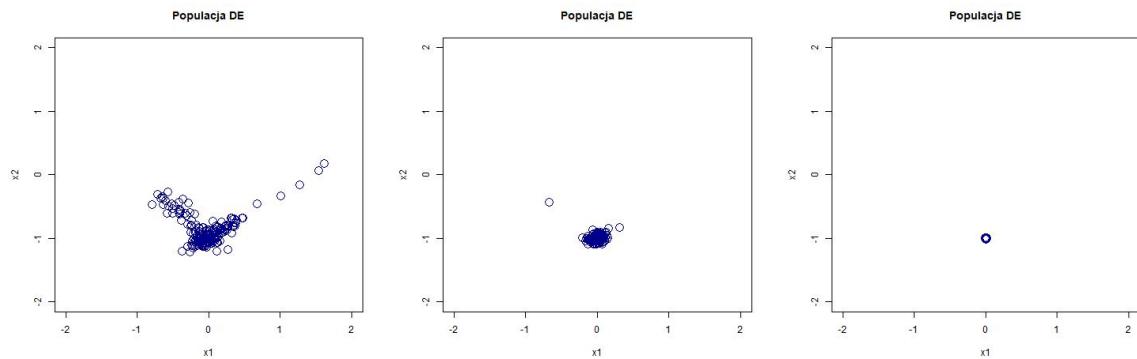


Rysunek 3.7.162: Wartości funkcji z biegiem iteracji

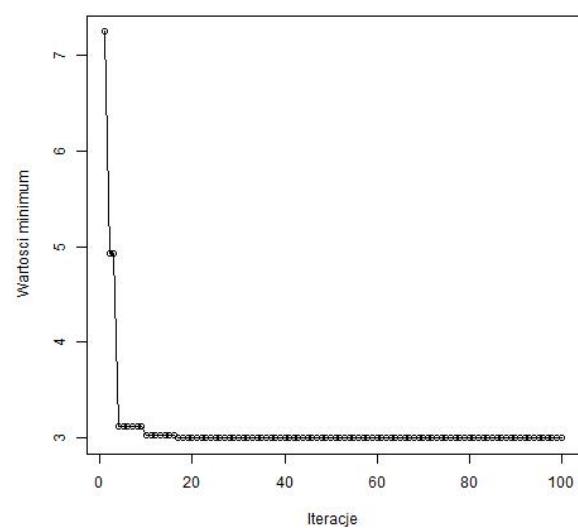
3.7.6. Goldstein



Rysunek 3.7.163: Iteracja 0, 4 i 8

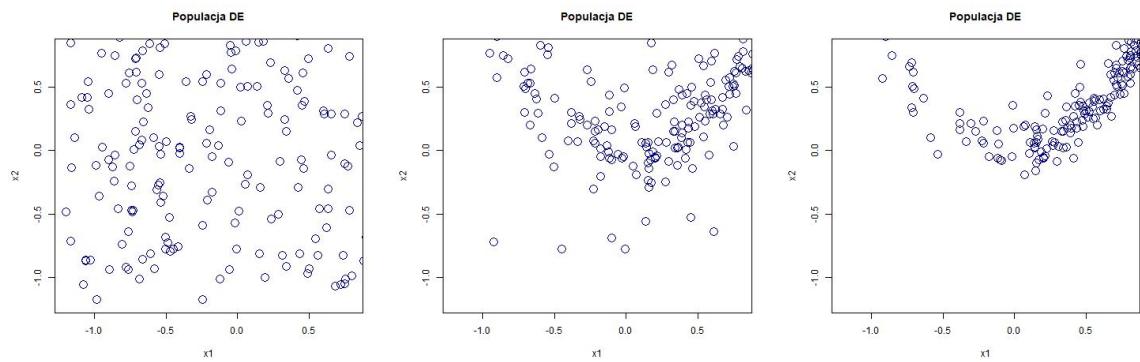


Rysunek 3.7.164: Iteracja 14, 20 i 30

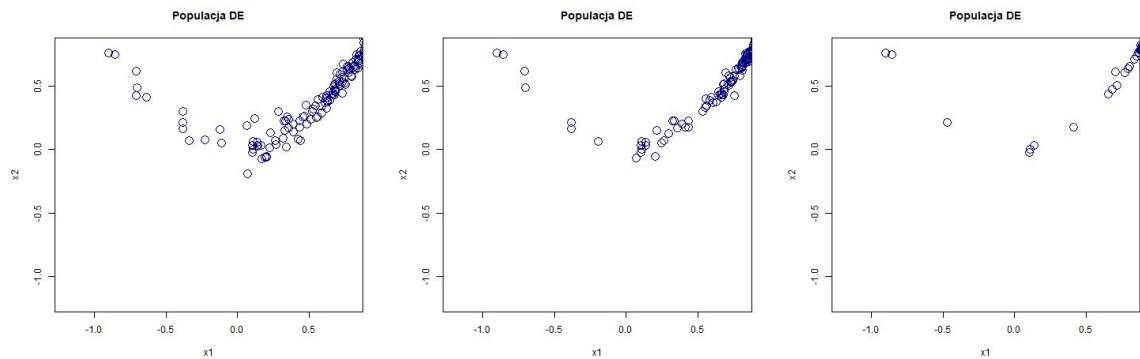


Rysunek 3.7.165: Wartości funkcji z biegiem iteracji

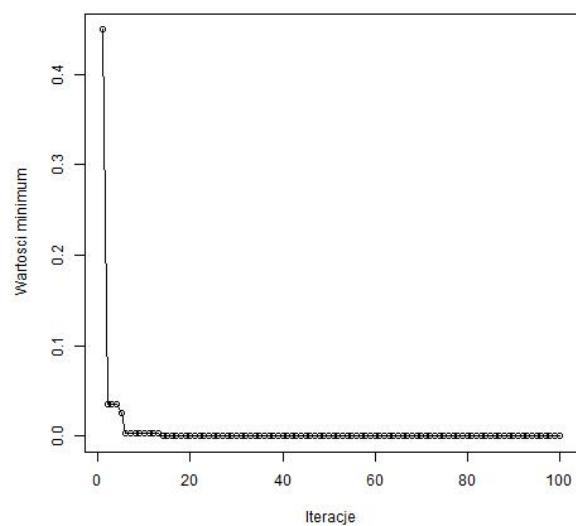
3.7.7. Leon



Rysunek 3.7.166: Iteracja 0, 4 i 8

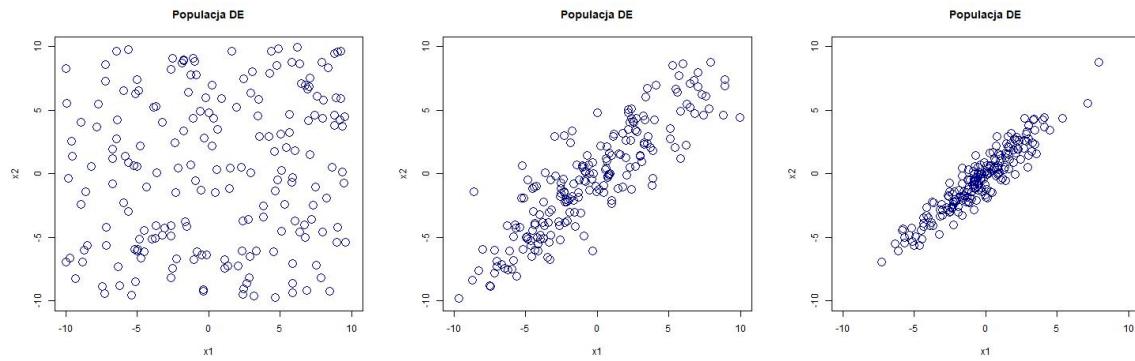


Rysunek 3.7.167: Iteracja 14, 20 i 30

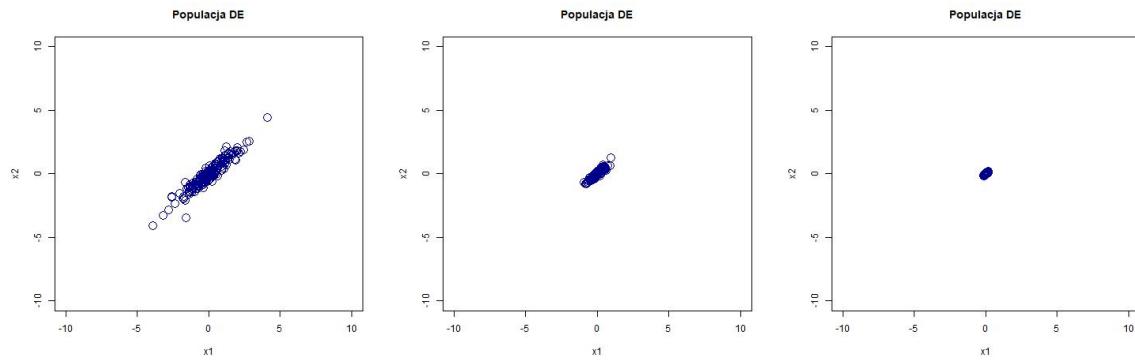


Rysunek 3.7.168: Wartości funkcji z biegiem iteracji

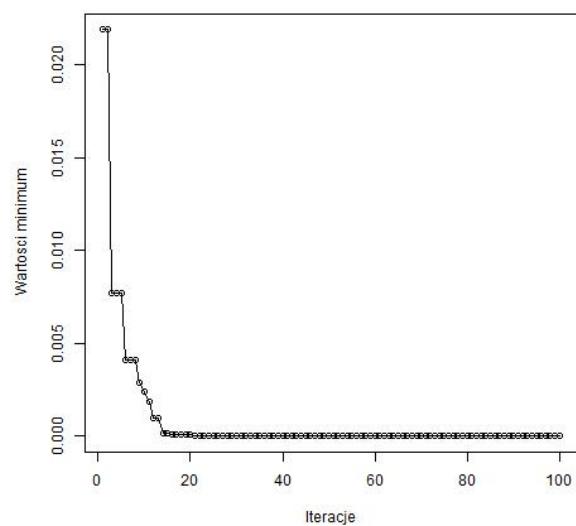
3.7.8. Matyas



Rysunek 3.7.169: Iteracja 0, 4 i 8

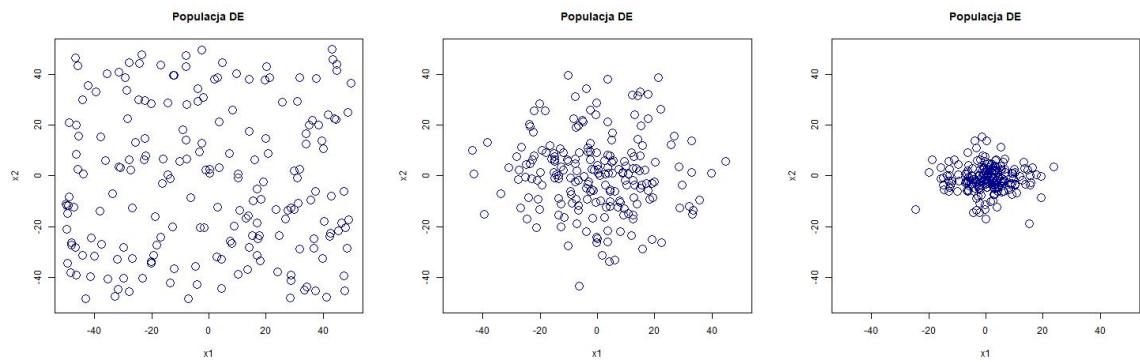


Rysunek 3.7.170: Iteracja 14, 20 i 30

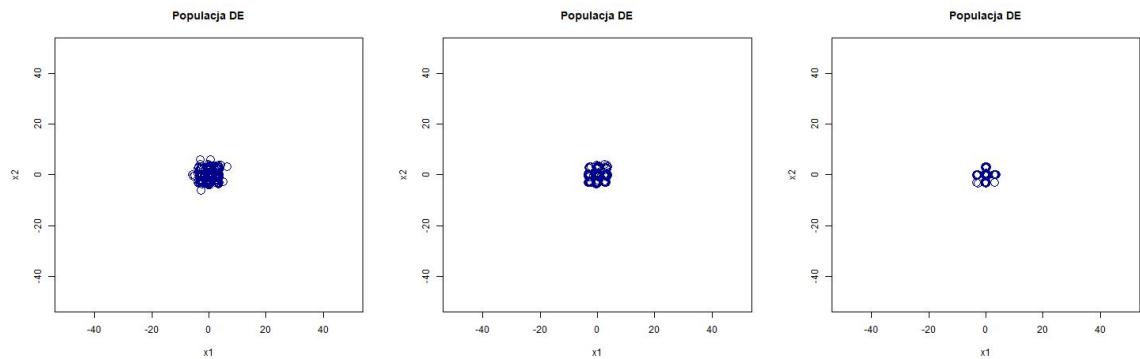


Rysunek 3.7.171: Wartości funkcji z biegiem iteracji

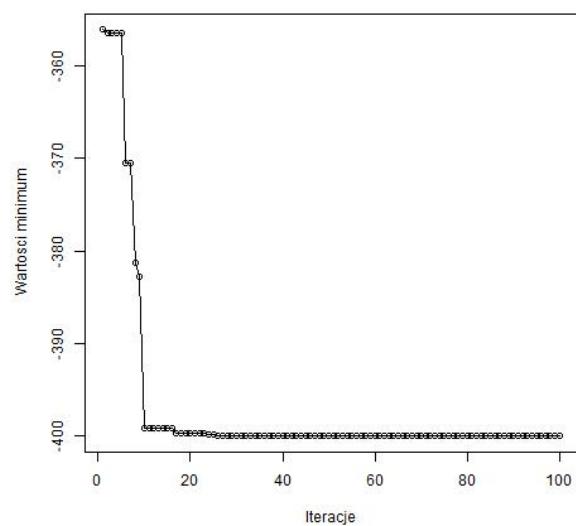
3.7.9. Venter



Rysunek 3.7.172: Iteracja 0, 4 i 8

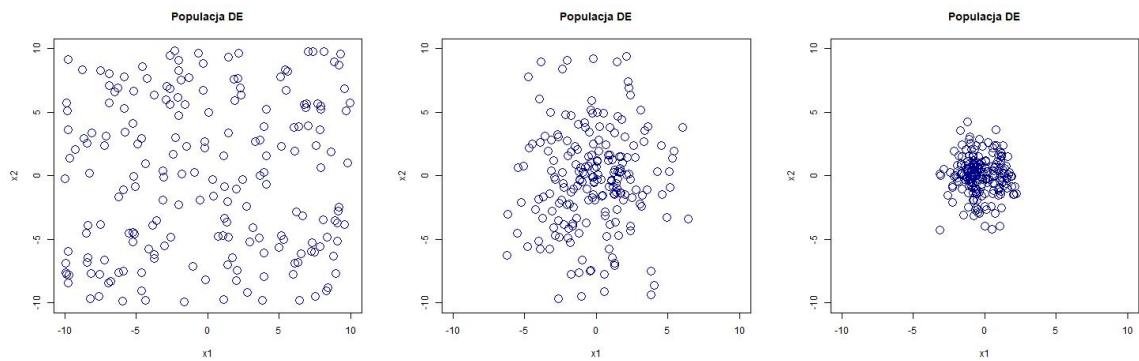


Rysunek 3.7.173: Iteracja 14, 20 i 30

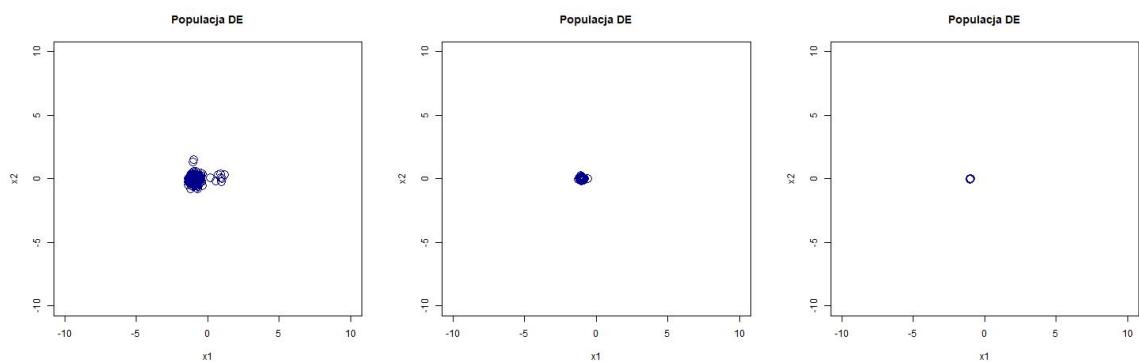


Rysunek 3.7.174: Wartości funkcji z biegiem iteracji

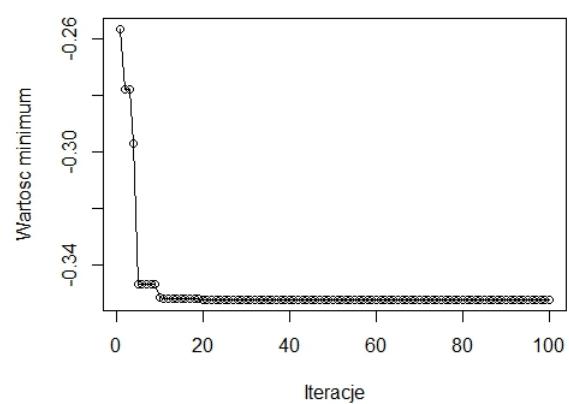
3.7.10. Zirilli



Rysunek 3.7.175: Iteracja 0, 4 i 8

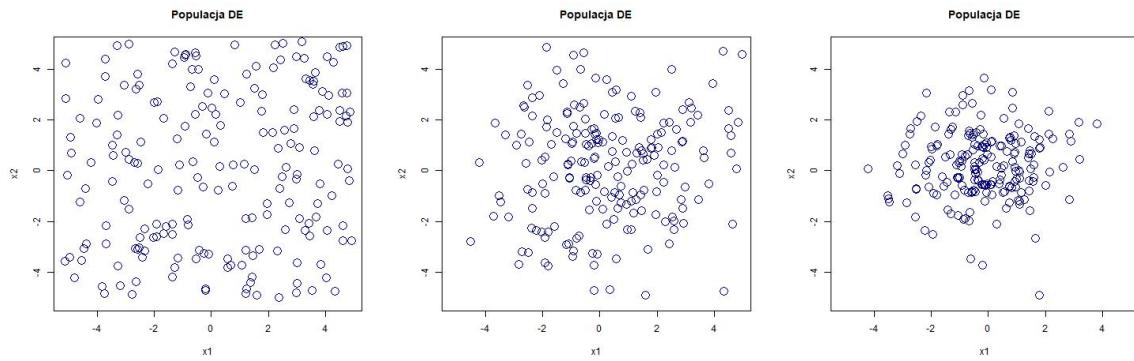


Rysunek 3.7.176: Iteracja 14, 20 i 30

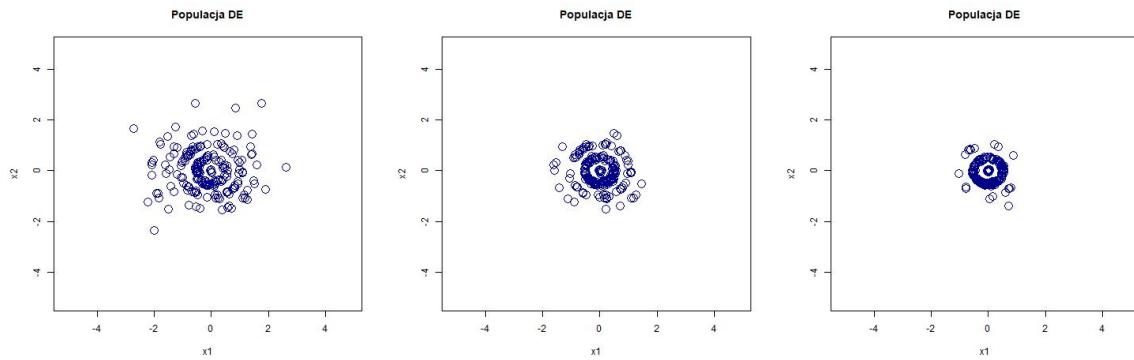


Rysunek 3.7.177: Wartości funkcji z biegiem iteracji

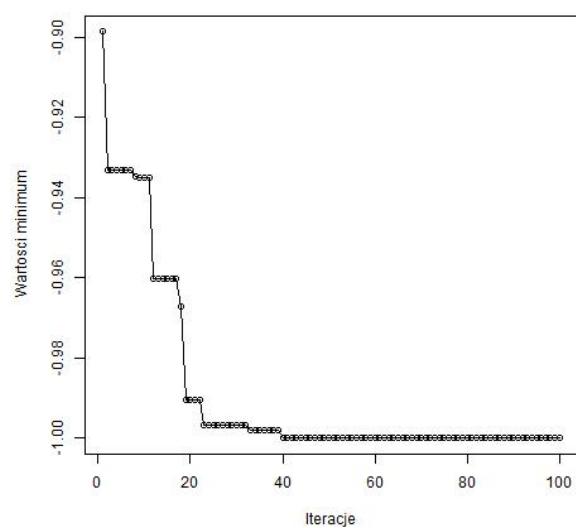
3.7.11. Drop Wave



Rysunek 3.7.178: Iteracja 0, 4 i 8

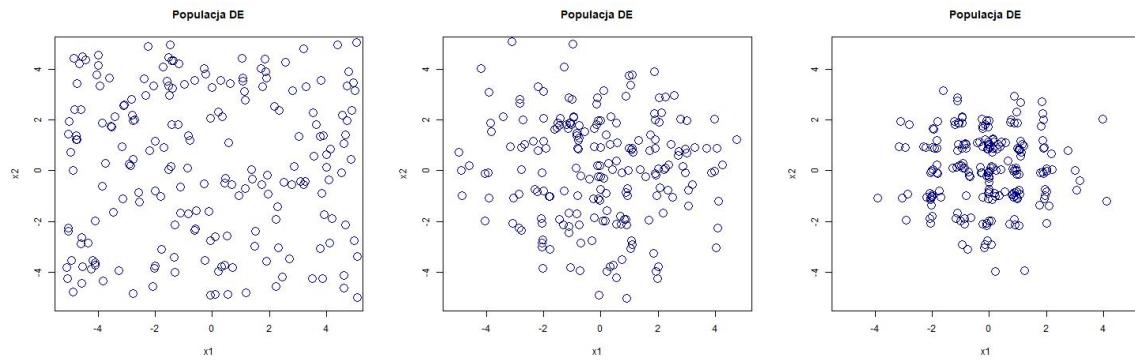


Rysunek 3.7.179: Iteracja 14, 20 i 30

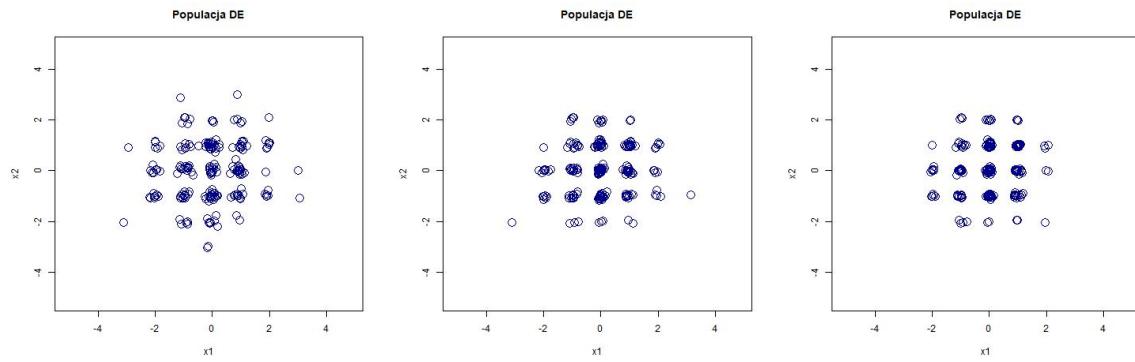


Rysunek 3.7.180: Wartości funkcji z biegiem iteracji

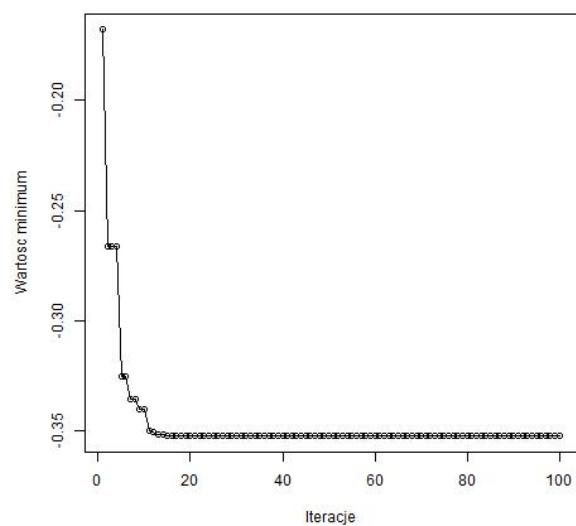
3.7.12. Rastrigin



Rysunek 3.7.181: Iteracja 0, 4 i 8

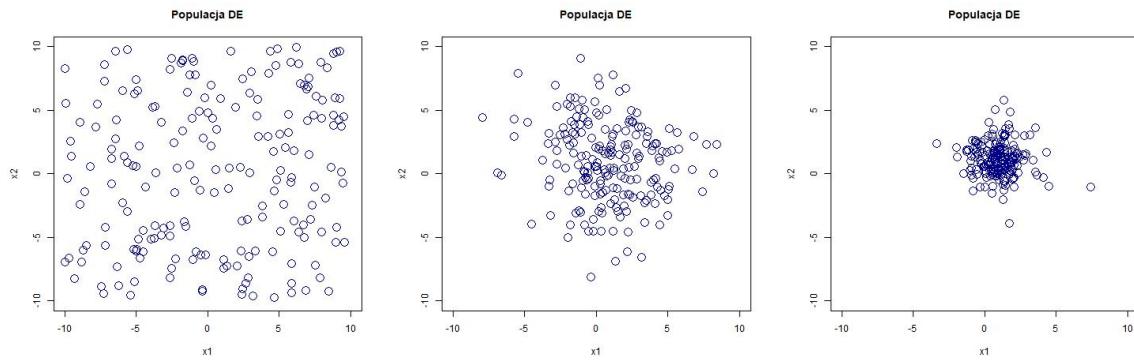


Rysunek 3.7.182: Iteracja 14, 20 i 30

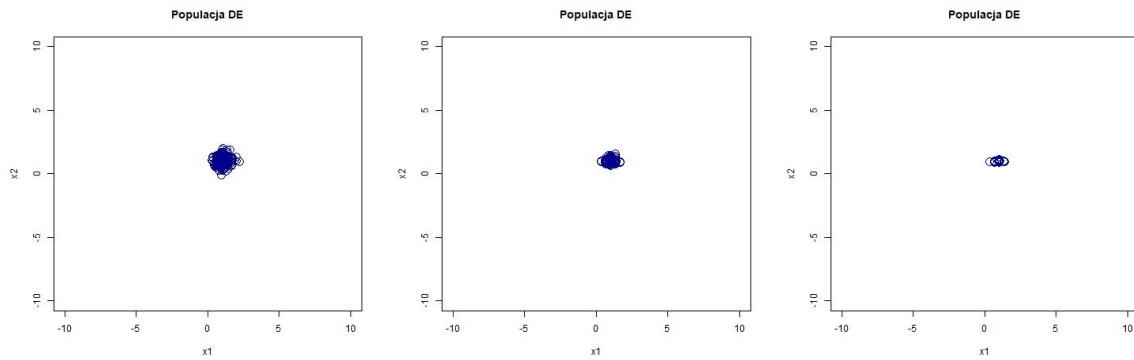


Rysunek 3.7.183: Wartości funkcji z biegiem iteracji

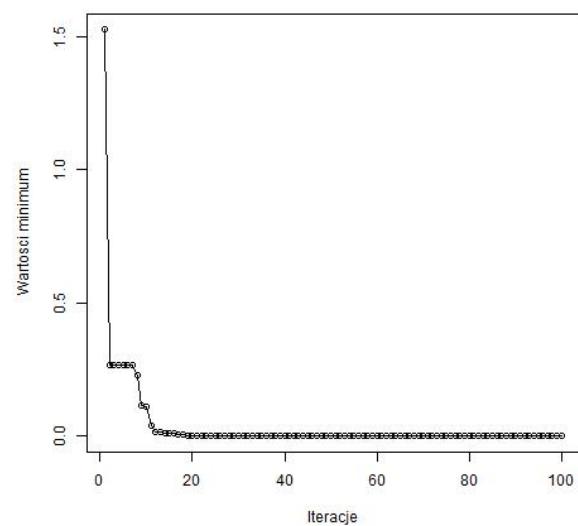
3.7.13. Levy N.13



Rysunek 3.7.184: Iteracja 0, 4 i 8



Rysunek 3.7.185: Iteracja 14, 20 i 30



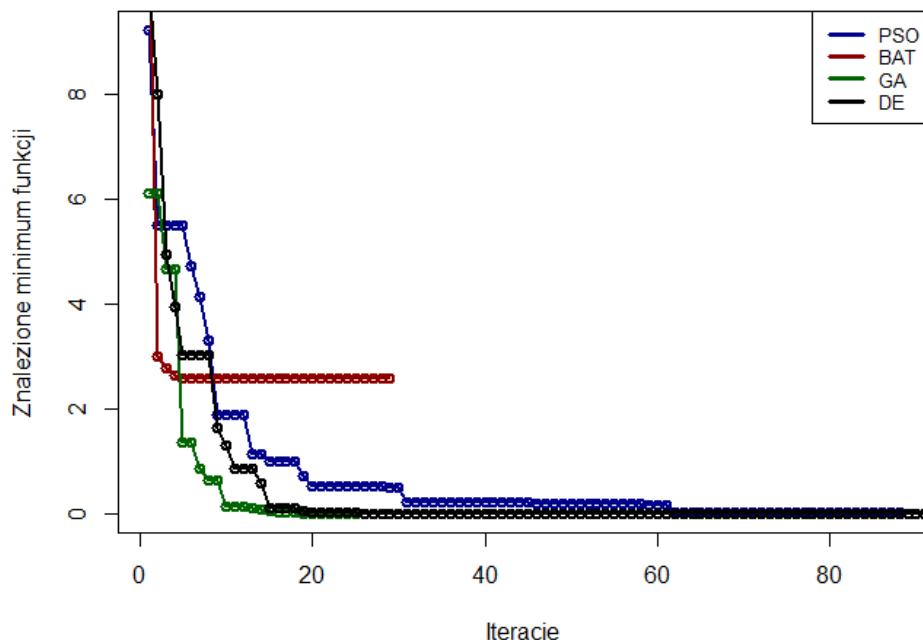
Rysunek 3.7.186: Wartości funkcji z biegiem iteracji

3.8. Omówienie wyników eksperymentu

Otrzymane wyniki z rozdziałów 3.4 - 3.7 dotyczące poszukiwania minimów poszczególnych funkcji zostaną teraz omówione w kontekście różnic rezultatów dla różnych algorytmów. Na rysunkach 3.8.187 - 3.8.199 przedstawiono zestawienie wykresów zmian wartości znalezionych ekstremów ze wzrostem iteracji, uzyskane przez wszystkie badane metody SI dla populacji 200.

3.8.1. Ackley

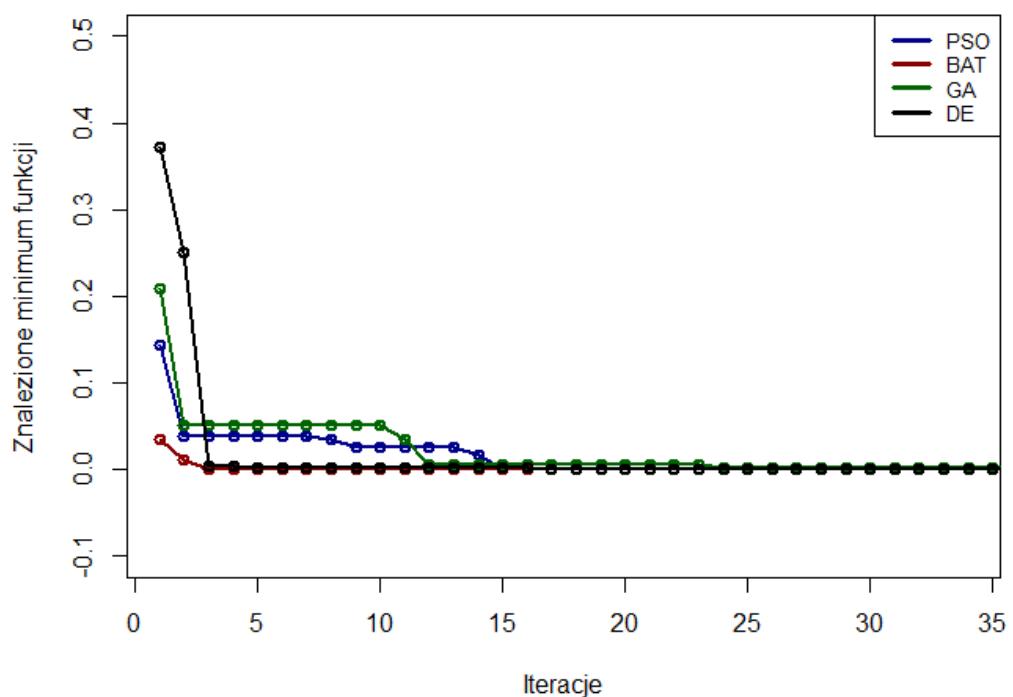
Poszukiwanie minimum pierwszej funkcji - Ackley, zostało zakończone z wynikami identycznymi z rzeczywistym minimum tej funkcji (0) w przypadku metod ewolucyjnych (GA i DE), zaś wyniki metod rojowych (PSO i BA) są tu mniej dokładne. Zwłaszcza spore błędy szukanych niewiadomych i minimum wykazał algorytm nietoperza (35 MSE dla 20 osobników). Błędy i niedokładności spadają znacznie ze wzrostem liczby iteracji (0,01-0,07s średnio zajmuje optymalizacja funkcji), a GA naj wolniejszy (0,07-0,43s w zależności od liczby populacji rozwiązań). Czasy algorytmów rojowych były do siebie zbliżone (0,02-0,17). Wszystkie czasy wykonania rosną ze wzrostem populacji.



Rysunek 3.8.187: Wartości funkcji Ackley znalezione w kolejnych iteracjach

3.8.2. Beale

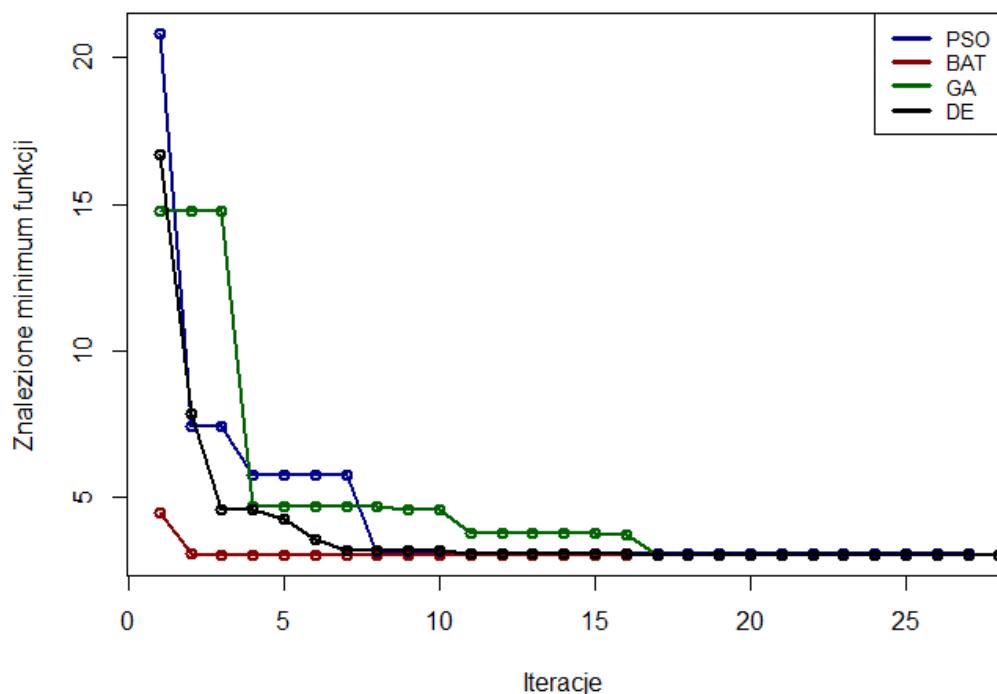
Dokładność wyników metod GA i DE jest na podobnie dobrym poziomie jak w przypadku Ackley, metody rojowe wykazały tutaj większą dokładność, obie uzyskały wyniki bliskie szukanego minimum równego 0. Ponownie GA okazał się najwolniejszy, choć szybszy niż w poprzedniej funkcji (0,05-0,28s).



Rysunek 3.8.188: Wartości funkcji Beale znalezione w kolejnych iteracjach

3.8.3. Goldstein

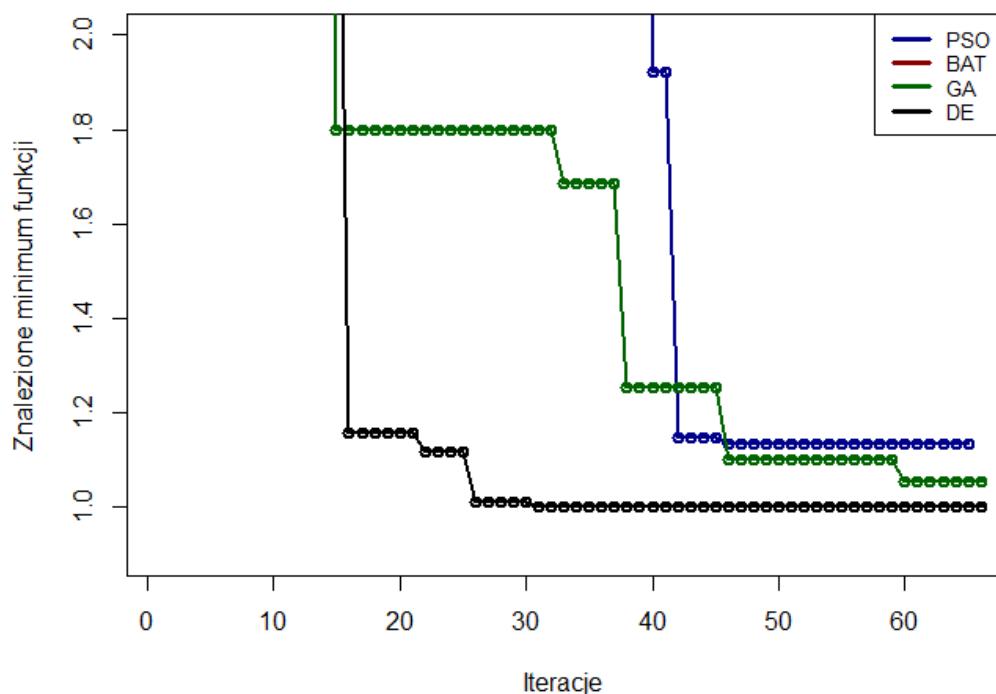
Uwagę zwracają tu rezultaty algorytmu nietoperza, które w przypadku małej liczebności roju (np. 20) są obarczone dużym błędem (250 MSE) i odchyleniem standaryzowanym (15), zaś przy dużej liczecnosti zbioru (od 70 wzwyż) jego wyniki są już doskonałe. Reszta metod - podobna jakość wyników jak poprzednio.



Rysunek 3.8.189: Wartości funkcji Goldstein znalezione w kolejnych iteracjach

3.8.4. Bartels Conn

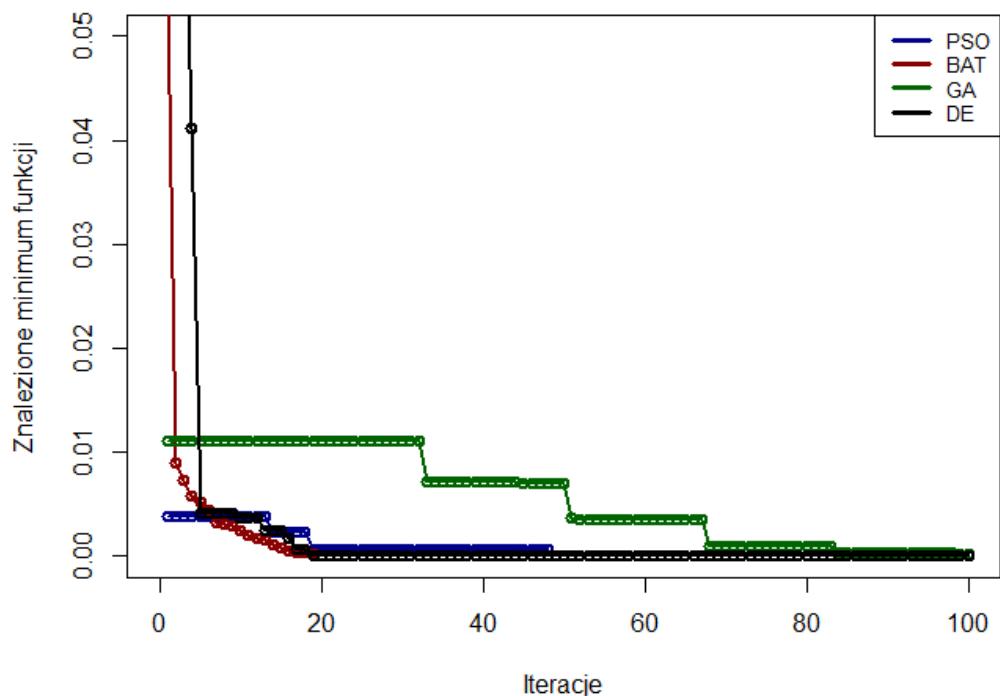
W tym przypadku jedynie algorytm DE znalazł poprawny wynik minimum funkcji wynoszący 1. Pozostałe metody dały błędne rezultaty szczególnie przy niewielkich populacjach (20-40), a zwłaszcza metody rojowe. Algorytm PSO uzyskał co prawda wynik zbliżony do rzeczywistego przy populacji 100 osobników, ale błędy algorytmu BA są ogromne dla każdej wielkości populacji (nawet przy populacji 200, $f(x) = 8$ dla tej metody zamiast 1). Dlatego na rysunku 3.8.190 wartości metody BA znalazły się poza widoczną częścią wykresu .Algorytm GA znalazł poprawny wynik przy populacji równej 200.



Rysunek 3.8.190: Wartości funkcji Bartels Conn znalezione w kolejnych iteracjach

3.8.5. Leon

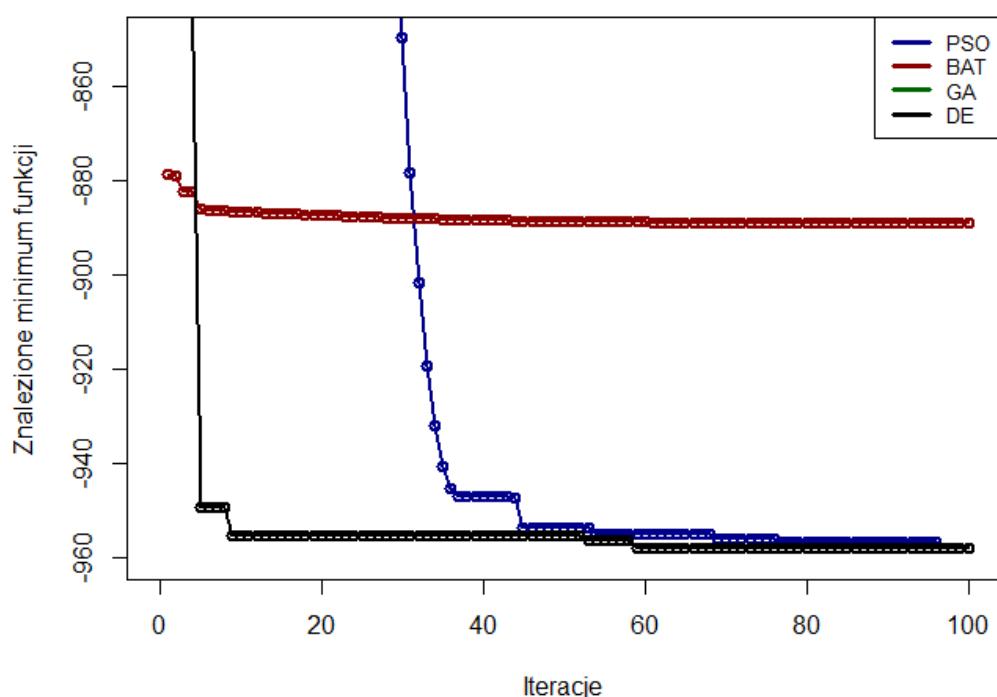
Funkcja Leon o minimum 0 była poprawnie optymalizowana przez wszystkie algorytmy, minimalne błędy dotyczą jedynie GA, którego czasy wykonania były za to wyraźnie krótsze niż w poprzednich funkcjach (0,04-0,16s) i są zbliżone do czasów alg. rojowych.



Rysunek 3.8.191: Wartości funkcji Leon znalezione w kolejnych iteracjach

3.8.6. Eggholder

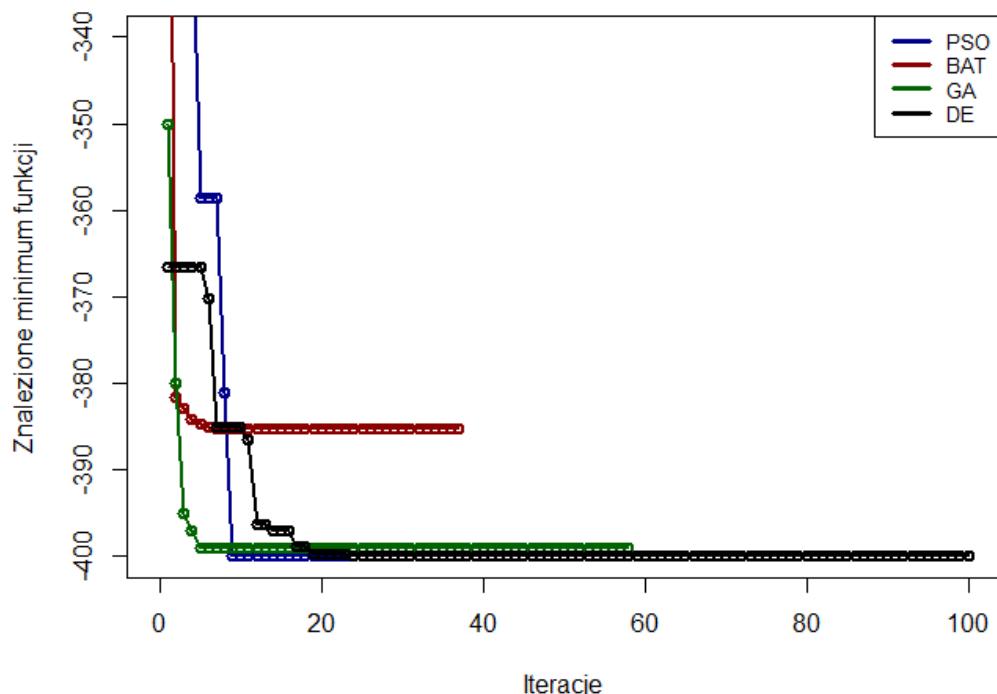
Dobre wyniki uzyskały PSO i DE ($f(x)$) od -897 do -959 dla PSO i od -928 do -957 dla DE, szukana wartość to -959). GA i BA odstają od nich z dużymi błędami MSE i odchyleniem standardowym równym kilkudziesięciu jednostek. DE wciąż jest najszybszy. Na rysunku 3.8.192 widać jak duże są różnice, linia reprezentująca GA z powodu dużego błędu znalazła się poza widocznym obszarem.



Rysunek 3.8.192: Wartości funkcji Eggholder znalezione w kolejnych iteracjach

3.8.7. Venter

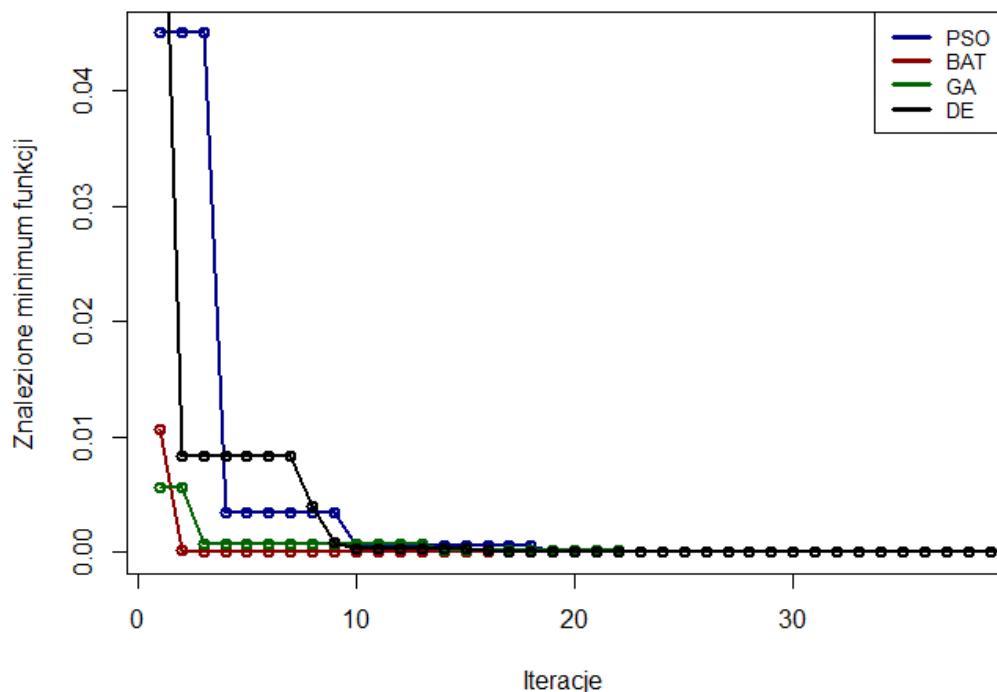
Idealne wyniki metody DE, nieco gorsze GA i PSO (błędy przy małej populacji), wyraźne niepoprawności wyniku BA (najlepszy to $f(x) = -387$, kiedy powinno być -400), aczkolwiek ze zwiększeniem populacji maleją. Algorytm GA najwolniejszy.



Rysunek 3.8.193: Wartości funkcji Venter znalezione w kolejnych iteracjach

3.8.8. Matyas

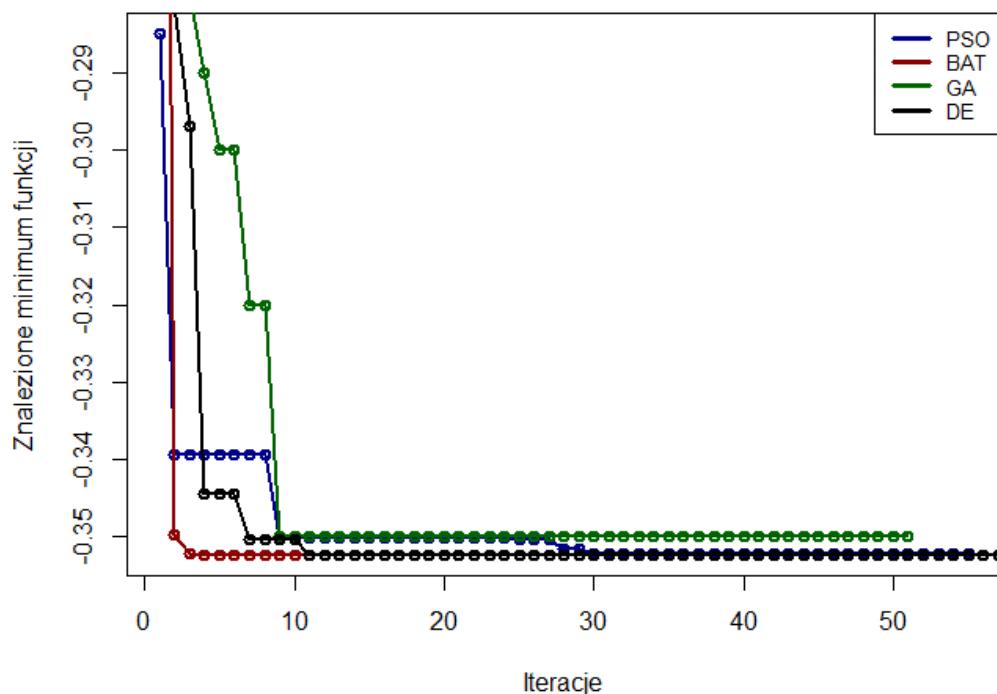
Minimum funkcji jest 0 i taki wynik został osiągnięty przez wszystkie metody nawet dla niskich liczebności zbioru poszukującego.



Rysunek 3.8.194: Wartości funkcji Matyas znalezione w kolejnych iteracjach

3.8.9. Zirilli

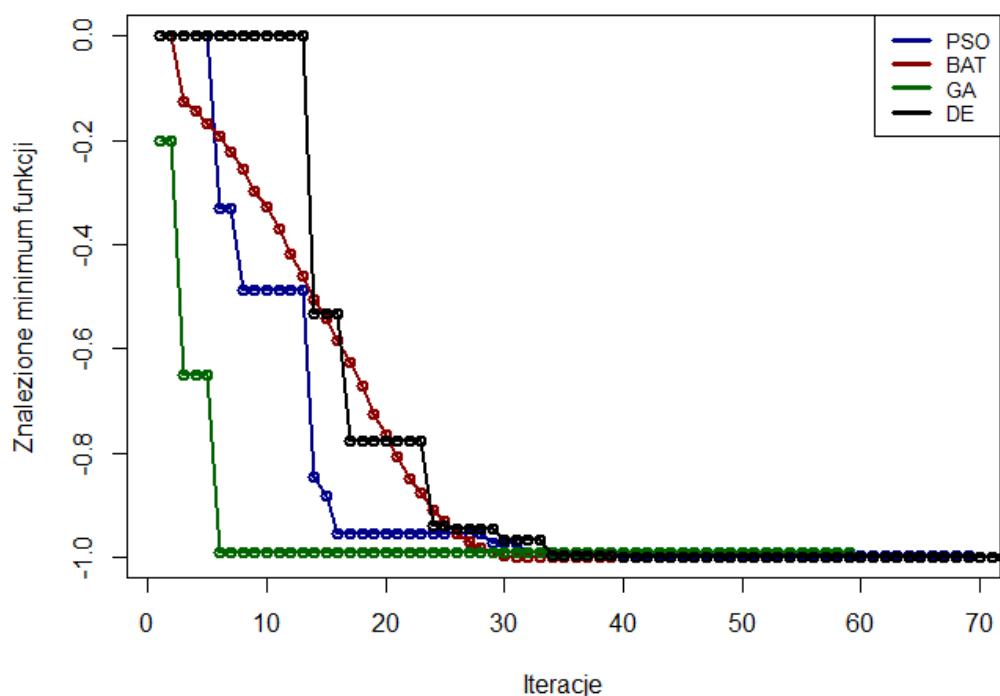
Analogicznie jak w przypadku Matyas, optymalizacja Zirilli dała wyniki bardzo dobrej jakości dla wszystkich metod. Różnica dotyczy czasów wykonania: najszybszy jest tym razem GA (0,04-0,08s).



Rysunek 3.8.195: Wartości funkcji Zirilli znalezione w kolejnych iteracjach

3.8.10. Easom

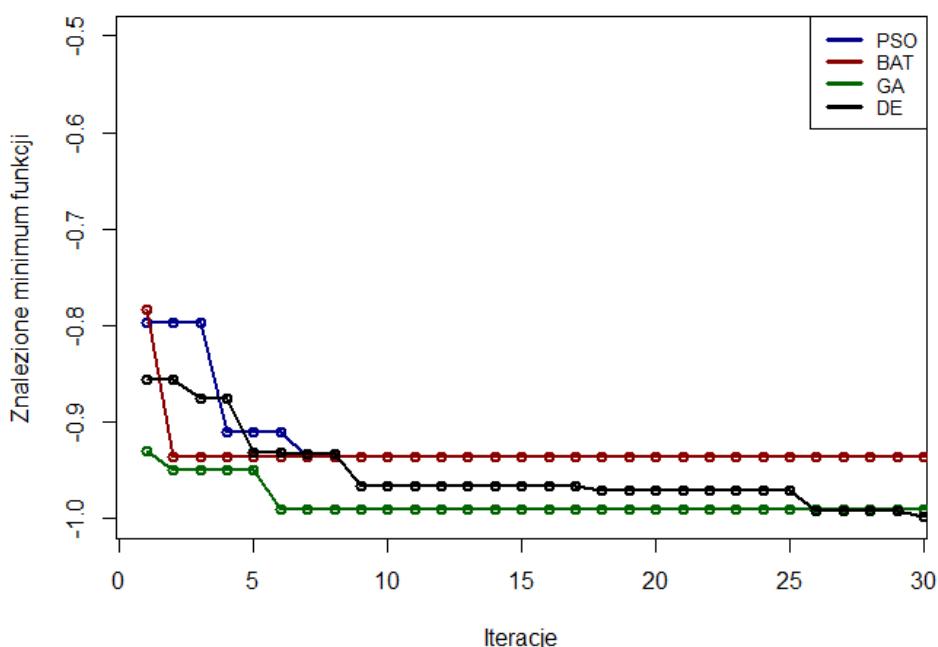
Istotne znaczenie miała tym razem liczebność populacji. Jeśli wynosiła 20, wyniki z aż trzech algorytmów (PSO, BA i GA) były niedokładne, jednak z jej wzrostem poprawność optymalizacji rosła zdecydowanie. GA znów najwolniejszym, a DE najszybszym z algorytmów.



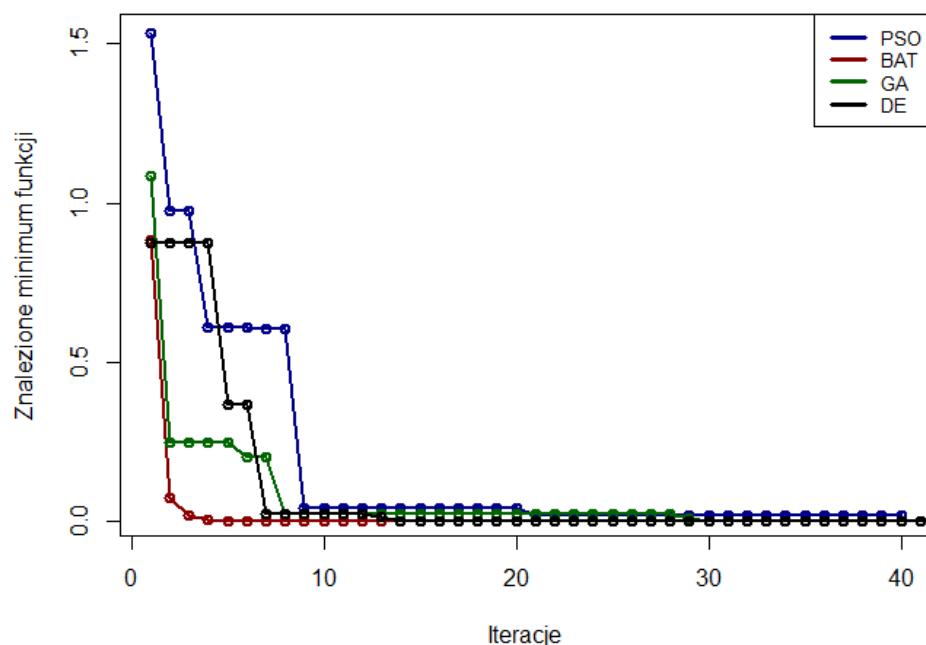
Rysunek 3.8.196: Wartości funkcji Easom znalezione w kolejnych iteracjach

3.8.11. Drop Wave, Levy N.13, Rastrigin

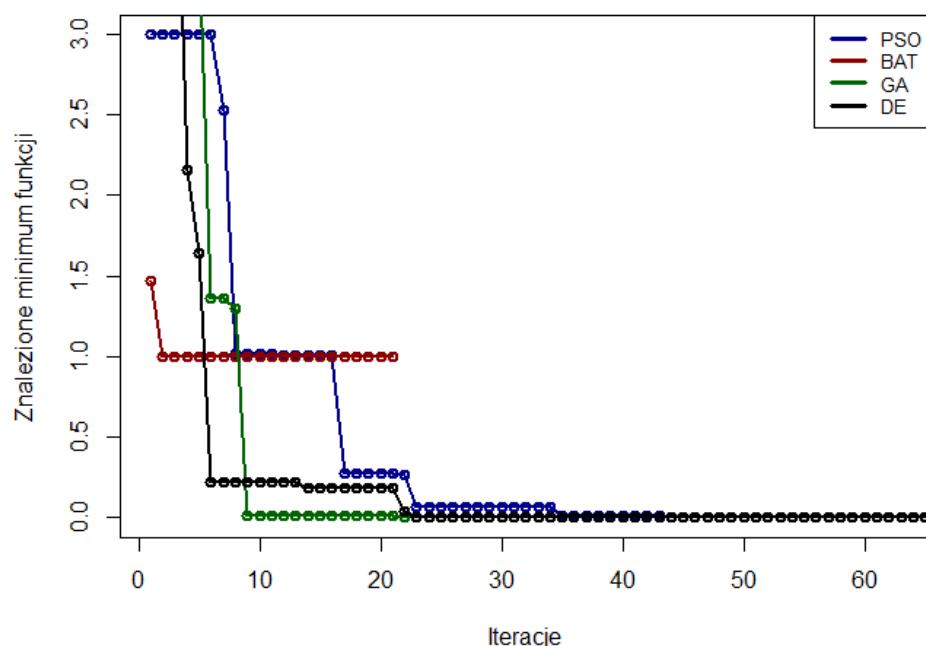
Poprawność wyników optymalizacji Drop Wave, Levy N.13 oraz Rastrigin jest do siebie zbliżona we wszystkich metodach, niewielkie błędy i odchylenie standardowe wystąpiły jedynie przy niskich liczebnościach populacji tj. 20 i 40. Zauważać można też gorszą jakość optymalizacji metody Bat na funkcji Rastrigin (minimum równe 1 przy 200 osobnikach, gdy powinno być 0). Czasy wykonania tych funkcji wynosiły przeciętnie 0,02-0,18s dla PSO, 0,02-0,16s dla Bat, 0,06-0,40s dla GA, 0,01-0,07,5



Rysunek 3.8.197: Wartości funkcji Drop Wave znalezione w kolejnych iteracjach



Rysunek 3.8.198: Wartości funkcji Levy N.13 znalezione w kolejnych iteracjach



Rysunek 3.8.199: Wartości funkcji Rastrigin znalezione w kolejnych iteracjach

4. Zakończenie

4.1. Podsumowanie

Przedstawione i omówione wyniki eksperymentów optymalizacji funkcji matematycznych, są potwierdzeniem, że wykorzystanie metod sztucznej inteligencji inspirowanych naturą może być wysoce korzystne jako wydajny sposób rozwiązywania problemów optymalizacyjnych. W trakcie prowadzonych badań korzystano z algorytmów rojowych (PSO, BA) i ewolucyjnych (DE, GA) do znajdywania minimum funkcji w ograniczonym zakresie wartości niewiadomych.

Jakość wyników optymalizacji zależała w głównej mierze od metody badania, funkcji i wielkości zbioru poszukującego rozwiązania. Bardzo ważną informacją jest fakt, że wraz ze zwiększeniem liczby populacji zdecydowanie rośnie poprawność uzyskiwanych rozwiązań. Szczególnie widoczne było to w przypadku metod rojowych podczas działania na funkcjach Eggholder, Bartels Conn lub Venter. Należy jednak zwrócić uwagę, że razem z dokładnością zdobytą dzięki np. zwiększeniu rojów rośnie gwałtownie czas wykonania programów, jest to wyraźne we wszystkich metodach. Kolejny niewątpliwym czynnikiem, który wpływał na wyniki to skomplikowanie samej funkcji badanej oraz zakres, w którym szukane są niewiadome. Funkcje o szerokich zakresach (Bartel Conn [-500,500], Eggholder [-512,512]) były najtrudniejsze do optymalizacji na podstawie analizy rezultatów. Natomiast funkcje o małym zakresie, których minimum ma nieduże wartości lub 0 (Matyas, Leon, Goldstein) były często rozwiązywane z właściwym wynikiem minimum nawet dla małych populacji (20,40). Trzeba zaznaczyć, iż trudność rozwiązywanej funkcji zdawała się wpływać na czas działania algorytmu jedynie w przypadku algorytmu nietoperza.

Eksperymenty wykazały, że algorytmy ewolucyjne cechują się większą dokładnością rozwiązywania problemów poszukiwania minimów funkcji niż metody rojowe. Na tym tle wyróżnił się algorytm ewolucji różnicowej (DE), który dawał bardzo poprawne wyniki, przy najkrótszym czasie działania. Również algorytm genetyczny (GA) charakteryzuje wysoka poprawność wyników (przy populacji liczby 70 i większych). Niestety jest to metoda najwolniejsza ze wszystkich, a jej wyniki dla trudnej optymalizacyjnie funkcji Eggholder były obarczone sporymi błędami. W przypadku tego zadania najlepiej wypadł algorytm roju częstek (PSO), który jest metodą uniwersalną - szybką i poprawną zarówno

dla funkcji trudniejszych jak i łatwych. Odpowiednia dokładność jest uzyskiwana w PSO przy rojach 100 oraz większych. Algorytm nietoperza (BA) okazał się najmniej dokładny. Uzyskane przez niego rezultaty w prostych funkcjach są zadowalające, lecz w trudniejszych błędy są za duże, żeby ta metoda SI mogła konkurować z pozostałymi. Być może jest to spowodowane sposobem implementacji tego algorytmu.

Badania potwierdziły skuteczność działania zaprezentowanych metod SI w problemach matematycznych. Używanie metod sztucznej inteligencji do tego typu zadań jest innowacyjnym i wydajnym sposobem, zapewniającym dobrą jakość wyników.

4.2. Wnioski

1. Przeprowadzono optymalizację poszukiwania minimum globalnego trzynastu funkcji matematycznych o dwóch niewiadomych, korzystając z czterech metod sztucznej inteligencji inspirowanych naturą.
2. Uzyskano wyniki o wysokiej zgodności z wartościami z literatury.
3. Zwiększenie populacji poszukującej rozwiązania zwiększa dokładność algorytmu, ale wydłuża czas jego działania.
4. Duże skomplikowanie funkcji oraz szeroki zakres ograniczenia wartości zmiennych powodują pogorszenie dokładności znalezionych rozwiązań, nie wpływają znacząco na czas poszukiwania.
5. Metoda ewolucji różnicowej jest najszybsza i najdokładniejsza.
6. Algorytm nietoperza nie powinien być używany do optymalizacji trudniejszych funkcji z powodu zbyt dużych błędów.
7. Algorytm genetyczny jest najbardziej czasochłonną metodą.
8. Metody ewolucyjne uzyskały lepsze wyniki od metod rojowych.
9. Wyniki badań potwierdziły przydatność metod rojowych i ewolucyjnych w rozwiązywaniu zadań optymalizacji funkcji.

Bibliografia

- [1] Russell S., Norvig P., *Artificial Intelligence A modern Approach*, wyd. Pearson, 2010, New Jersey.
- [2] <http://www.loebner.net/Prizef/loebner-prize.html>
- [3] Kisielewicz A., *Sztuczna inteligencja i logika podsumowanie przedsięwzięcia naukowego*, Wydawnictwo Naukowo-Techniczne, Warszawa 2011.
- [4] Głowiński C., *Sztuczna inteligencja, PC Kurier*, 14-27, 4 luty 1999.
- [5] Harel D., *Rzecz o istocie informatyki. Algorytmika*, Wydawnictwo Naukowo-Techniczne, Warszawa 2001.
- [6] Trojanowski K., *Metaheurystyki praktyczne Wydanie 2, poprawione*, Wyższa Szkoła Informatyki Stosowanej i Zarządzania, Warszawa 2008.
- [7] Hansen P., Mladenović N., *Handbook of Metaheuristics*, str. 145-184. Kluwer Academic Publisher, 2003.
- [8] Fonseca C. M., *Multiobjective Genetic Algorithms with Application to Control Engineering Problems*, PhD thesis, Department of Automatic Control and Systems Engineering, University of Sheffield, 1995.
- [9] Schaefer J. D., *Multiple objective optimization with vector evaluated genetic algorithms, InGenetic Algorithms and their Applications*, Proceedings of the First International Conference on Genetic Algorithms, pp. 93-100, Lawerence Erlnaum, 1985.
- [10] Zitzler E., Laumanns M., Bleuler S.: *A Tutorial on Evolutionary Multiobjective Optimization*, Swiss Federal Institute of Technology (ETH) Zurich.

- [11] Horn J., Nafpliotis N., Goldberg D.: *A Niche Pareto Genetic Algorithm for Multiobjective Optimization*, IEEE 1994, Volume 1, pp. 82-87.
- [12] Srinivas N., Kalyanmoy Deb: Multiobjective Optimization Using Nondominated Sorting In Genetic Algorithm, Department of Mechanical Engineering, Indiana Institute of Technology.
- [13] Rutkowski L., *Metody i techniki sztucznej inteligencji*, Wydawnictwo Naukowe PWN, Warszawa 2011.
- [14] Bonabeau E., Theraulaz G., Mądrości roju, *Świat nauki*, Nr 6, 49-56, Czerwiec 2000.
- [15] Brownlee J., *Clever Algorithms: Nature-Inspired Programming Recipes*, 2011.
- [16] Karaboga D.: *An idea based on honey bee swarm for numerical optimization*, Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
- [17] Kennedy J., Eberhart R.: *Particle swarm optimization*, Proceedings of the International Conference on Neural Network, pp. 1942-1948, 1995.
- [18] Engelbrecht A.: *Particle Swarm Optimization: Velocity Initialization*, in Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, pp. 70-77, 2012.
- [19] Helwig S., Branke J., Mostaghim S.: *Experimental analysis of bound handling techniques in particle swarm optimization*, IEEE Transactions on Evolutionary Computation, Vol. 17, No. 2, pp. 259-271, 2013.
- [20] Chen S., Montgomery J., Bolufé-Röhler A., Gonzalez-Fernandez Y.: *Standard particle swarm optimization on the CEC2013 real parameter optimization benchmark functions (revised)*, Technical Report, School of Information Technology, York University, Toronto, Ontario, December 2013.
- [21] Yang X.S.: *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, 2008.
- [22] Yang X.S.: *Firefly algorithms for multimodal optimization*, Stochastic Algorithms: Foundations and Applications, SAGA, Lecture Notes in Computer Sciences, 5792, 169–178, 2009.

- [23] Lukasik S., Źak S., *Firefly algorithm for continuous constrained optimization task*, Computational Collective Intelligenc, Semantic Web, Social Networks and Multiagent Systems LNCS, 5796, 97–106, 2009.
- [24] Yang X.S., *A new metaheuristic bat-inspired algorithm*, in Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), vol. 284, pp. 65–74, Springer, 2010.
- [25] Xie J., Zhou Y., Chen H., *A novel bat algorithm based on differential operator and Lévy flights trajectory*, Computational Intelligence and Neuroscience, vol. 2013, Article ID 453812, 13 pages, 2013.
- [26] Xie J., Zhou Y., Zheng H., *A hybrid bat algorithm with path relinking for capacitated vehicle routing problem*, Mathematical Problems in Engineering, vol. 2013, Article ID 392789, 10 pages, 2013.
- [27] Xie J., Zhou Y., Zheng H., *A hybrid metaheuristic for multiple runways aircraft landing problem based on bat algorithm*, Journal of Applied Mathematics, vol. 2013, Article ID 742653, 8 pages, 2013.
- [28] Gandomi A.H., Yang X.S., Alavi A.H., Talatahari S., *Bat algorithm for constrained optimization tasks*, Neural Computing and Applications, vol. 22, no. 6, pp. 1239–1255, 2013.
- [29] Yang X.S., Hossein Gandomi A., *Bat algorithm: a novel approach for global engineering optimization*, Engineering Computations, vol. 29, no. 5, pp. 464–483, 2012.
- [30] Yang X.S., *Bat algorithm for multi-objective optimisation*, International Journal of Bio-Inspired Computation, vol. 3, no. 5, pp. 267–274, 2011.
- [31] A. Rezaee Jordehi, *Chaotic bat swarm optimisation (CBSO)*, Applied Soft Computing, vol. 26, pp. 523–530, 2015.
- [32] Zhu B., Zhu W., Liu Z., Duan Q., Cao L., *A Novel Quantum-Behaved Bat Algorithm with Mean Best Position Directed for Numerical Optimization*, Computational Intelligence and Neuroscience Volume 2016, Article ID 6097484, 17 pages, 2016.
- [33] Michalewicz Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag Berlin Heidelberg, 1992.

- [34] Vose M. D., Leipins G.E.: *Schema disruption*, In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 237-243, Morgan Kaufmann (San Mateo), 1991.
- [35] Kozieł S.: *Algorytmy ewolucyjne i ich zastosowania do optymalizacji i modelowania analogowych układów elektronicznych*, Rozprawa doktorska, Politechnika Gdańsk, Wydział Elektroniki, Telekomunikacji i Informatyki, Gdańsk 1999.
- [36] Radcliffe N. J., Surry P. D.: *Fundamental Limitations on Search Algorithms: Evolutionary Computing in Perspective*, Lecture Notes in Computer Science, Vol. 1000, J. Van Leeuwen (ed.), Springer-Verlag, 1995.
- [37] Słowik A.: *Właściwości i zastosowania algorytmów ewolucyjnych w optymalizacji*, Metody Informatyki Stosowanej, nr 2/2007 Kwartalnik Komisji Informatyki Polskiej Akademii Nauk Oddział w Gdańsku.
- [38] Arabas J.: *Wykłady z algorytmów ewolucyjnych*, WNT, 2004.
- [39] Storn R., Price K., *Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces*, Journal of Global Optimization, 11(4), s. 341-359, 1997.
- [40] R Core Team (2017). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL <http://www.R-project.org/>.
- [41] RStudio Team (2016). RStudio: Integrated Development for R. RStudio, Inc., Boston, MA URL <http://www.rstudio.com/>.
- [42] Ciupke K.: psoptim: Particle Swarm Optimization, R package version 1.0, 2016, URL: <https://CRAN.R-project.org/package=psoptim>.
- [43] Hwang S. H., Moon R. M., microbats: An Implementation of Bat Algorithm in R, R package version 0.1-1, 2016, URL: <https://CRAN.R-project.org/package=microbats>, <https://github.com/stathwang/microbats>.
- [44] Scrucca, L. (2013) GA: A Package for Genetic Algorithms in R. Journal of Statistical Software, 53(4), 1-37. <https://www.jstatsoft.org/article/view/v053i04>.

- [45] Ardia, D., Boudt, K., Carl, P., Mullen, K.M., Peterson, B.G. (2011) Differential Evolution with DEoptim. An Application to Non-Convex Portfolio Optimization. *The R Journal*, 3(1), 27-34. URL: https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Ardia~et~al.pdf. Differential Evolution homepage: URL <http://www.icsi.berkeley.edu/~storn/code.html>.
- [46] Surjanovic, S. & Bingham, D. (2013). Virtual Library of Simulation Experiments: Test Functions and Datasets. Retrieved August 9, 2017, from <http://www.sfu.ca/~ssurjano>.
- [47] Jamil M., Yang X. S., A literature survey of benchmark functions for global optimization problems, *Int. Journal of Mathematical Modelling and Numerical Optimisation*, Vol. 4, No. 2, pp. 150–194 (2013).

5. Streszczenie

Streszczenie pracy dyplomowej

Zastosowanie metod sztucznej inteligencji inspirowanych naturą do rozwiązywania problemów wielokryterialnych

Imię i nazwisko autora pracy: Jonaś Kulpinski

Nr albumu: 79579

Imię i nazwisko promotora pracy: Jacek Czerniak

Słowa kluczowe: sztuczna inteligencja, algorytmy, R, optymalizacja, metody rojowe, metody ewolucyjne

Treść streszczenia:

W przedstawionym opracowaniu zaprezentowano kilka rodzajów metod sztucznej inteligencji inspirowanych naturą, które mogą być użyteczne do rozwiązywania problemów wielokryterialnych.

W części eksperimentalnej przeprowadzono optymalizację na trzynastu funkcjach matematycznych, szukając ich minimum globalnego. Wykorzystano cztery metody SI: optymalizację rojem częstek, algorytm nietoperza, algorytm genetyczny, ewolucję różnicową. Badania powtarzano wielokrotnie, sprawdzając wydajność i poprawność wyników poszukiwania rozwiązań przez metody z różną liczebnością populacji. Badania potwierdziły przydatność algorytmów sztucznej inteligencji do optymalizacji funkcji. Stwierdzono, że badane metody ewolucyjne SI charakteryzują się większą dokładnością przy rozwiązywaniu problemów optymalizacji niż metody rojowe.

A summary of the thesis

Application of artificial intelligence methods inspired by nature to solve multi-criterion problems

Name and surname of the author: Józef Kulpinski

Album number: 79579

Name and surname of the thesis promoter: Jacek Czerniak

Keywords: artificial intelligence, algorithms, R, optimization, swarming methods, evolutionary methods

Summary:

The presented paper discusses several types of artificial intelligence methods inspired by nature that may be useful for solving multi-criterion problems.

In the experimental part, optimization was performed on thirteen mathematical functions, seeking their global minimum. Four SI methods were used: particle swarm optimization, bat algorithm, genetic algorithm, differential evolution. The research was repeated many times, checking the efficiency and the correctness of the results of the search for solutions by means of different populations. Research has confirmed the usefulness of artificial intelligence algorithms for function optimization. It was found that the investigated evolutionary methods of SI are more accurate in solving optimization problems than swarms.

Jonasz Kulpinski

79579

Edukacja techniczno-informatyczna

Inżynieria biokompozytów

Studia drugiego stopnia, stacjonarne

OŚWIADCZENIE autora pracy dyplomowej

Świadomy odpowiedzialności prawnej oświadczam, że praca dyplomowa

Zastosowanie metod sztucznej inteligencji inspirowanych naturą do rozwiązywania problemów wielokryterialnych

została wykonana samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że:

- 1) przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w uczelni;
- 2) drukowana wersja pracy dyplomowej jest identyczna z wprowadzoną do programu APD elektroniczną wersją.

.....

(podpis studenta)

Bydgoszcz, dn.