Wpływ wykorzystania archiwum do mutacji różnicowej w algorytmie ewolucji różnicowej na jakość optymalizacji w przestrzeni ciągłej.

Andrzej Fiedukowicz

Spis treści

Streszczenie

Algorytmy ewolucji różnicowej (DE) występują w wielu wariantach i odmianach, których właściwości są cały czas badane. Jedną z proponowanych modyfikacji tego algorytmu stanowi wykorzystanie przy operatorze mutacji różnicowej osobników pochodzących z szerszego niż jednopokoleniowego okna historii.

Niniejsza praca opisuje proces opracowywania metody zastosowania archiwum osobników w celu poprawy jakości wyników uzyskiwanych w ewolucji różnicowej. W dalszej części praca skupia się na sprawdzeniu hipotezy o możliwości poprawy klasycznego schematu ewolucji różnicowej DE/rand/1/bin *przez wykorzystanie archiwum, proponując algorytm* NADE. *Ponadto, praca odnosi także zaproponowaną modyfikację do zaawansowanego algorytmu ewolucji różnicowej* SHADE *[10], proponując algorytm* ANADE *wprowadzający ową modyfikację do algorytmu* SHADE.

W ramach pracy porównano proponowane algorytmy i wyciągnięto wnioski na temat możliwości poprawy uzyskiwanych przez algorytmy wyników przez zastosowanie archiwum. Badania skuteczności poszczególnych metod przeprowadzone zostały w oparciu o zestaw funkcji testowych opracowanych w ramach konferencji IEEE Congress on Evolutionary Computation 2013 (CEC2013).

Dodatkowo w ramach pracy poruszono problematykę metodyki badań nad stochastycznymi algorytmami optymalizacyjnymi, wskazując na potencjalne drogi jej rozwoju.

Słowa kluczowe

Ewolucja Różnicowa, DE, Algorytmy Ewolucyjne, AE, Optymalizacja, Optymalizacja black–box

Abstract

There are many forms of differential evolution algorithms (DE), which properties are yet to be discovered. One of possible modifications of those algorithms is using historical points to perform differential mutation..

This thesis describes development process of method of using such archive to increase DE results quality. Hereafter this thesis is checking hipotesis about possibility of improving classic DE/rand/1/bin *by using archive, proposing the* NADE *algorithm. This thesis also propose* ANADE *algorithm that applies proposed archive technique to advanced differencial evolution algorithm –* SHADE.

This thesis compares proposed algorithms and puts a proposal about possibility of improving algorithm performance by applying archive to their basic design. All experiments performed during this reasearch are based on benchmark functions set developed during the IEEE Congress of Evolutionary Computation 2013 conference (CEC2013).

Additionally this thesis describes issues of methodology used in this research, pointing possible ways of its improvements.

Keywords

Differential Evolution, DE, Evolutionary Algorithms, EA, Optimization, Black–box optimization

Rozdział 1

Wstęp

Problem optymalizacji jest jednym z najczęściej pojawiających się w praktycznych zastosowaniach problemów numerycznych. Sama powszechność podstawowego problemu, ale także możliwość sprowadzenia do problemu optymalizacji problemów klasyfikacji, regresji, grupowania czy też określając szerzej – uczenia maszynowego [1, 2, 3], sprawia, że przed algorytmami optymalizacji stawiane są coraz bardziej złożone i wymagające zadania.

Stale zwiększający się poziom trudności problemów optymalizacyjnych jak i ciągła niedoskonałość dotychczas wytworzonych metod ich rozwiązywania powoduje duże zainteresowanie badaniami dotyczącymi tworzenia coraz doskonalszych, szybszych i dających lepsze rezultaty algorytmów owe problemy rozwiązujących [4].

1.1 Definicja problemu

Praca ta, umiejscowiona została w kontekście rozwiązywania problemów optymalizacji gdzie analityczne wyznaczenie ekstremum funkcji jest niemożliwe bądź w ogóle nie jest znana jej analityczna forma. Zakłada się bowiem, że zadanie jest zdefiniowane następująco [5]:

Definition 1. *Problem globalnej optymalizacji funkcji $f : X \to \mathbb{R}$ polega na wyznaczeniu takiego $x_0 \in X$, że:*

$$\forall_{x \in X} f(x_0) \leqslant f(x)$$

Dla celów tej pracy przyjmuje się możliwość zbadania wartości funkcji f w dowolnym punkcie ze zbioru X, jednak brak możliwości bezpośredniego zbadania jakichkolwiek innych własności tejże funkcji. W tej formie problem ten można nazwać problemem optymalizacji funkcji zadanej w formie reaktywnej (*Black–box optimization*).

W ramach rozpatrywanych problemów istnieje także możliwość, że dziedzina funkcji f jest szersza niż zbiór rozwiązań dopuszczalnych. W takich wypadkach mówimy o problemie optymalizacji z ograniczeniami.

Definition 2. *Problem globalnej optymalizacji funkcji $f : X \to \mathbb{R}$ z ograniczeni $X_D \subset X$ polega na wyznaczeniu takiego $x_0 \in X_D$, że:*

$$\forall_{x \in X_D} f(x_0) \leqslant f(x)$$

Dla tego rodzaju problemów, możliwe jest wyznaczanie wartości funkcji w całej jej dziedzinie X, jednak poszukiwane rozwiązanie musi pochodzić z ograniczonego zbioru X_D. Problem globalnej optymalizacji funkcji z ograniczeniami stanowi uogólnienie problemu optymalizacji globalnej bez ograniczeń.

Szczególnym przypadkiem problemu optymalizacji globalnej z ograniczeniami jest przeszukiwanie przestrzeni ciągłej.

Definition 3. *Problem globalnej optymalizacji funkcji* $f : \mathbb{R}^n \to \mathbb{R}$ *z ograniczeniem* $X_D = \{x \in \mathbb{R}^n : g_1(x) \leqslant 0, g_2(x) \leqslant 0, \ldots, g_m(x) \leqslant 0\}$*, gdzie* $g_1, g_2 \ldots g_m : \mathbb{R}^n \to \mathbb{R}$*, nazywamy problemem optymalizacji globalnej przestrzeni ciągłej z ograniczeniami nierównościowymi.*

W tym wariancie przyjmuje się, że $X = \mathbb{R}^n$ gdzie $n \in \mathbb{N}$, a X_D zdefiniowane jest przez funkcje ograniczeń.

Przy tych założeniach, należy zauważyć, że dla każdego z powyższych problemów jeśli zbiór X jest nieskończenie liczny (nawet przeliczalny), nie jest możliwe zweryfikowanie, czy wyznaczony punkt jest rozwiązaniem zadanego problemu. Co więcej, w wielu przypadkach (brak istnienia hiperpłaszczyzny X, takiej, że w każdym jej punkcie wartość funkcji jest równa poszukiwanemu ekstremum) prawdopodobieństwo odnalezienia rozwiązania problemu (niezależnie od metody przeszukiwania przestrzeni) wynosi zero.

Mając to na względzie by problem można było uznać za rozwiązany, należy go przeformułować tak, że rozwiązaniem jest dowolny element $x \in X_D$, a jego jakość wyznaczana jest przez wartość funkcji f w tym punkcie. Dla tak zadanego problemu możliwe jest porównywanie różnych metod przeglądania przestrzeni X_D pod względem jakości otrzymanego rozwiązania.

Niniejsza praca będzie skupiała się na rozwiązywaniu tak przeformułowanego problemu optymalizacji globalnej funkcji w przestrzeni ciągłej z ograniczeniami nierównościowymi. Ze względu na zwięzłość zapisu w dalszej części pracy problem ten nazywany będzie **problemem optymalizacji** lub po prostu **problemem**.

Warto zwrócić uwagę, że pomimo iż między różnymi rozwiązaniami problemu istnieje relacja częściowego porządku generowana przez wartość funkcji f, to jednak przy przyjętych założeniach wartość (rozumiana jako liczba rzeczywista) tej funkcji nie niesie żadnej dodatkowej informacji. Z tego też powodu, wszelkie badania porównawcze przeprowadzane według poszczególnych metodyk rozwiązywania problemu muszą być oparte na tej właśnie relacji porządku.

1.2 Motywacje

Jednym z podejść do zdefiniowanego powyżej problemu optymalizacji jest wykorzystanie algorytmów ewolucji różnicowej (DE) [6]. Algorytm ten zaproponowany w 1995r. od wielu lat rozwijany w ramach licznych prac naukowych [7, 1] cieszy się zwykle bardzo dobrymi rezultatami w testach jakościowych [8].

Typową procedurą przy heurystycznych algorytmach optymalizacyjnych jest wykorzystywanie do generowania kolejnych punktów w przestrzeni przeszukiwań punktów z okna historii o zmiennej szerokości. W przypadku klasycznego algorytmu ewolucji różnicowej, okno historii ma rozmiar równy liczności populacji – starsze punkty nie są wykorzystywane w kolejnych pokoleniach. Choć w przeszłości podejmowano próby wykorzystywania

szerszego okna historii w tym algorytmie [9, 10], temat ten zwykle występował przy okazji innych badań i nie był osobno zbadany. Praca ta będzie starała się dokładnie zbadać możliwości wykorzystania tego rodzaju strategii i określić w jaki sposób wykorzystanie tej strategii wpłynie na jakość uzyskiwanych przy pomocy ewolucji różnicowej wyników.

1.3 Cel

Celem pracy jest zbadanie zmodyfikowanego klasycznego algorytmu ewolucji różnicowej tak by wektory mutacji różnicowej były budowane w oparciu o osobniki z archiwum o parametryzowalnym rozmiarze.

Przebadane zostaną możliwe podejścia do wykorzystania tych osobników i wynikające z nich zmiany jakości działania algorytmu. W wyniku tych działań zdefiniowany zostanie możliwie najskuteczniejszy algorytm parametryzowany wielkością archiwum (H) (zwany dalej *NADE*).

Cel pracy stanowi także zbadanie gotowego algorytmu *NADE* przy różnych wartościach nowego parametru i ustalenie jak zmiana parametru wpływa na jakość wyników. Otrzymane rezultaty zostaną następnie zestawione z klasycznym algorytmem ewolucji różnicowej, a także przedstawione w kontekście zmodyfikowanych wariantów ewolucji różnicowej przebadanych w ramach konferencji CEC2013 (*IEEE Congress on Evolutionary Computation*).

W szczególności utworzony zostanie algorytm *ANADE* oparty o algorytm *SHADE* [10] w celu sprawdzenia, jak proponowane podejście do archiwum będzie sprawdzało się w przypadku zaawansowanych wariantów ewolucji różnicowej.

Wszelkie badania jakościowe w ramach pracy będą opracowane przy użyciu testowych funkcji wyspecyfikowanych w ramach konferencji CEC2013 i porównania rezultatów otrzymanych w określonym w ramach konferencji budżecie zapytań o wartości funkcji [11].

Rozdział 2

Problematyka ewolucji różnicowej

2.1 Podejscie heurystyczne do problemu optymalizacji

Do rozwiązywania problemów optymalizacji ze względu na ich złożoność i (w wielu przypadkach) brak możliwości odnalezienia precyzyjnego rozwiązania stosuje się algorytmy o charakterze heurystycznym. Algorytmy te mają za zadanie rozwiązywać problemy w sposób przybliżony tj. z założenia nie muszą podać rozwiązania dokładnego. Są one stosowane zamiast algorytmów dokładnych tam, gdzie:

- nie jest możliwe stworzenie precyzyjnego algorytmu
- nie jest znany precyzyjny algorytm
- precyzyjny algorytm jest zbyt kosztowny w wykonaniu

Algorytmy tej klasy bywają stosowane jako rozwiązanie kompromisowe pomiędzy jakością rozwiązania a czasem jego poszukiwania.

Podejście tego rodzaju niosą za sobą ciekawe konsekwencje. Warto zauważyć, że ponieważ z definicji prowadzą one do rozwiązania nie koniecznie najlepszego, to jeśli istnieje metoda porównywania jakości wyników, można także mówić o porównywaniu jakości działania różnych algorytmów w kontekście danego zadania. Mimo, iż wiele z algorytmów heurystycznych ma charakter niedeterministyczny (tj. dla tego samego wejścia mogą wygenerować dwa różne wyjścia) posługując się narzędziami statystyki i probabilistyki można wysnuwać wnioski dt. jakości algorytmów w kontekście problemu.

Wartym wspomnienia w kontekście algorytmów heurystycznych jest sposób ich konstruowania. Często dyktowany jest on intuicją, a podstawy teoretyczne działania tego rodzaju algorytmów bywają badane dużo później [7]. Tego rodzaju podejście pozwoliło na stworzenie wielu dobrze sprawujących się algorytmów, lecz zarazem ich analityczne badanie pozwala na dokładniejsze zrozumienie problematyki, uogólnienie lub uproszczenie reguł ich działania [7].

2.1.1 Model heurystycznego algorytmu optymalizacyjnego

Algorymty heurystyczne, służące rozwiązywaniu problemu optymalizacji, opierają się o ideę domniemywania pewnych właściwości przeszukiwanej przestrzeni na podstawie punktów już sprawdzonych i przyjętych *a priori* złożeń. Mając na względzie ten fakt, jak i wcześniej przedstawiony problem, definiując [2] zadanie optymalizacyjne jako czwórkę $\Pi = \langle X,\ f,\ S,\ T \rangle$ gdzie:

- X – przestrzeń przeszukiwań
- f – funkcja optymalizowana
- S – zbiór punktów początkowych
- T – kryterium zatrzymania

możemy określić także metodę przeszukiwania jako trójkę $M = \langle X,\ I,\ O \rangle$ [2], gdzie:

- X – przestrzeń przeszukiwań
- I – operator inicjacji – $I : S \times U \to X$
- O – operator zagregowany – $X : \Pi \times H \times U \to X$

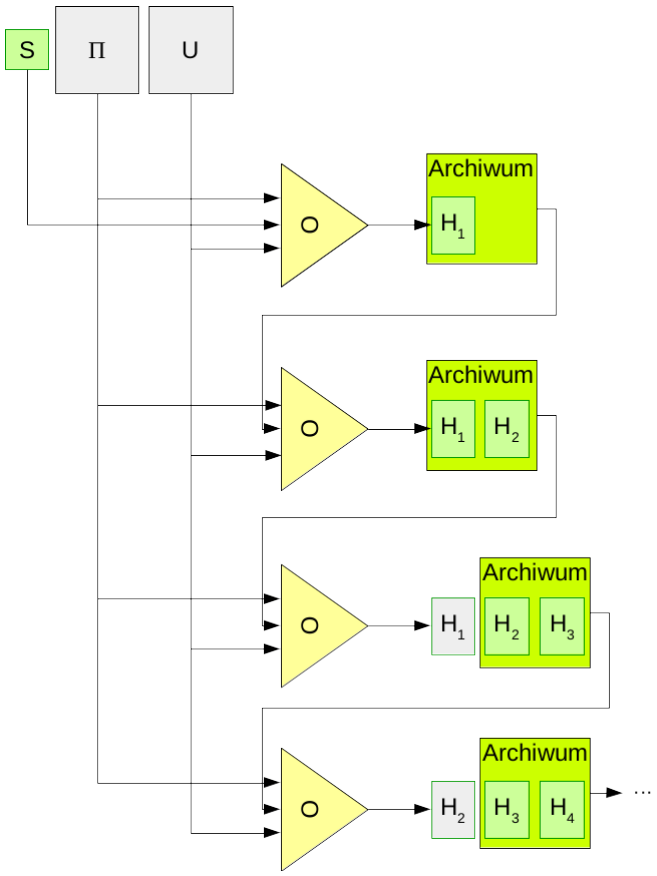
Jako U oznaczono przestrzeń sekwencji losowych, natomiast jako H – dotychczas otrzymane przez algorytm wyniki badania zadania Π.

2.1.2 Archiwum

W kontekście przedstawionego powyżej modelu, można zauważyć, że istotą konkretnego algorytmu optymalizacyjnego jest operator zagregowany O, na którego działanie wpływ ma wyłącznie sam problem (Π), ewentualne wyniki pseudolosowych operacji (U) oraz wyniki otrzymane dotychczas przez algorytm przy badaniu optymalizowanej funkcji (H). Ten ostatni element nazywany jest **archiwum** i stanowi centralny punkt proponowanej modyfikacji algorytmu ewolucji różnicowej – stąd zostanie on bardziej dokładnie opisany.

Szeroka definicja archiwum, nie wymaga by miało ono w jakikolwiek sposób ograniczoną pojemność. W skrajnym wypadku można więc pokazać algorytm, wykorzystujący wszystkie zbadane dotąd wartości funkcji optymalizowanej. Typowe algorytmy operują jednak tylko na oknie archiwum będącym podzbiorem H (Rysunek 2.1). Taka charakterystyka algorytmów jest uzasadniona w szczególności ze względu na:

- Kosztowność przechowywania i przetwarzania całego archiwum.
- Brak poprawy jakości algorytmu przy wykorzystaniu większej ilości punktów w archiwum lub poprawę niewystarczającą względem dodatkowych kosztów przetwarzania.
- Brak cennych dla działania algorytmu informacji i przesłanek w starszych punktach archiwum.



Rysunek 2.1: Działanie heurystycznego algorytmu optymalizacyjnego z oknem archiwum wielkości równej dwukrotności osobników generowanych przez pojedynczą iterację. Zastosowano okno archiwum będące kolejką FIFO.

I tak na przykład w opisanym w rozdziale 2.2.2 klasycznym algorytmie ewolucji różnicowej, mamy do czynienia z archiwum o rozmiarze równym wielkości populacji. Oznacza to, że do przeprowadzenia iteracji $N+1$. algorytmu, oprócz Π oraz U potrzebujemy także wyniku N. iteracji, ale już nie wyniku iteracji $N-1$. a tym bardziej całego archiwum H.

Tego rodzaju ograniczenia w wykorzystywaniu archiwum H skłaniają do rozważań nad możliwością i stosownością wykorzystania informacji lub przesłanek ukrytych w nieużywanej części archiwum do poprawienia funkcjonowania algorytmu. W dziedzinie algorytmów heurystycznych, nie jest koncepcją nową [IGdzieŹródło] by parametryzować wykorzystywaną wielkość okna archiwum – umożliwiając w ten sposób strojenie algorytmu oraz poprawianie jakości uzyskiwanych rezultatów. Pojawia się więc przypuszczenie, że także w przypadku ewolucji różnicowej powiększenie wykorzystywanego okna archiwum może umożliwiać uzyskiwanie lepszych jakościowo wyników optymalizacji.

2.2 Ewolucja różnicowa

2.2.1 Algorytmy z grupy ewolucyjnych

Przykładem algorytmów heurystycznych stosowanych do rozwiązywania problemu optymalizacyjnego są algorytmy należące do grupy algorytmów ewolucyjnych (ang. *Evolutionary Algorithms* – AE) [12, 1]. Algorytmy te, wzorowane na ewolucji naturalnej operują pojęciami takimi jak osobnik, środowisko czy przystosowanie w celu adaptacyjnego przeszukiwania przestrzeni optymalizacyjnej.

Ogólna idea stojąca za algorytmami ewolucyjnymi, polega na opisaniu genotypu osobników, z których następnie można odczytać fenotyp (stanowiący punkt przestrzeni przeszukiwań) oceniany przez środowisko (funkcję optymalizowaną). Osobniki – podobnie jak osobniki w naturze – podlegają krzyżowaniu oraz niewielkim mutacjom, walcząc między sobą o zasoby naturalne. Zgodnie z teorią doboru naturalnego, osobniki gorzej dostosowane do środowiska są – w szerokiej perspektywie czasowej i przy odpowiednio dużej populacji – wypierane przez te dostosowane lepiej.

W związku z powyższą analogią w ramach algorytmów ewolucyjnych wyróżnia się następujące operatory:

Krzyżowanie – to operator opisujący w jaki sposób na podstawie genotypu dwóch lub więcej osobników wytworzyć genotyp jednego lub więcej osobnika potomnego.

Mutacja – to operator wprowadzający w genotypie wybranego osobnika losowe (nieznaczne) zabużanie.

Selekcja – to operator odpowiadający selekcji naturalnej, który oceniając osobniki na podstawie przystosowania ich fenotypu do środowiska odrzuca część z nich preferując pozostawienie w populacji osobników lepiej dostosowanych.

Dobór do krzyżowania – to operator wybierający zestawy osobników podlegające krzyżowaniu. Operator ten opisuje znany z natury mechanizm preferencji (fenotypowej) organizmów przy rozmnażaniu.

Dobór do mutacji – to operator wybierający, które spośród osobników w populacji należy poddać mutacji. Odpowiada on zmiennemu w zależności od fenotypu organizmu prawdopodobieństwu wystąpienia u niego mutacji.

Inicjalizacja – to operator nieco oderwany od teorii ewolucji (gdyż ta nie opisuje powstawania pierwszych organizmów) opisujący sposób wygenerowania populacji początkowej spośród wszystkich możliwych genotypów. Operując na nomenklaturze z modelu opisanego w rozdziale 2.1.1, operator ten jest funkcją przeprowadzającą zbiór punktów początkowych S w początkowy zbiór genotypów.

Ogólny schemat działania algorytmu ewolucyjnego opiera się o iteracyjną adaptację populacji aż do ustalonego warunku stopu, według następujących kroków:

Algorithm 1 Podstawowy schemat algorytmów ewolucyjnych

```
1: function EWOLUCJA(S, U, f)
2:     P ← S
3:     while !stop do
4:         C ← DOBORDOKRZYZOWANIA(P, U, f)
5:         P ← P ∪ KRZYZOWANIE(C, U)
6:         M ← DOBORDOMUTACJI(P, U, f)
7:         P ← P ∪ MUTACJA(M)
8:         P ← SELEKCJA(P, U, f)
9:     end while
10:    return argmin_{x∈X} f(x)
11: end function
```

W tym miejscu warto nadmienić, że choć w powyższym opisie algorytmów ewolucyjnych jest wprost mowa o podziale na genotyp i fenotyp, to na potrzeby zdefiniowanego problemu pojęcie fenotypu będzie wystarczające. Algorytmy opierające się o operacje na zakodowanym genotypie oderwanym od jego semantyki stanowią odrębną gałąź algorytmów ewolucyjnych i są nazywane algorytmami genetycznymi. Algorytmy te, ze względu na specyfikę swojego działania wymagają odpowiedniej reprezentacji danych i odpowiednio dobranych do reprezentacji operatorów. W przypadku rozważanego w pracy problemu optymalizacji, operowanie bezpośrednio na fenotypie ocenianym przez środowisko (a więc wektorach liczb rzeczywistych dla których wyznaczane są wartości funkcji optymalizowanej) ze względu na właściwości i możliwości samych liczb rzeczywistych wydaje się być w zupełności wystarczające. Zagadnienie podziału na genotyp i fenotyp będzie w dalszej części pracy zaniedbane.

Eksploracja i Eksploatacja

Ważnymi pojęciami opisującymi sposób w jaki osobniki w ramach algorytmów ewolucyjnych przeszukują przestrzeń są pojęcia **eksploracji** i **eksploatacji**. Definiują one dwutorowe działanie grup osobników:

Eksploracja – to proces poszukiwania nowych, przyciągających osobniki obszarów przestrzeni przeszukiwań charakteryzujących się niską wartością funkcji przystosowania. Pojęcie to opisuje proces poszukiwania w całej przestrzeni obszarów przyciągania wielu minimów lokalnych.

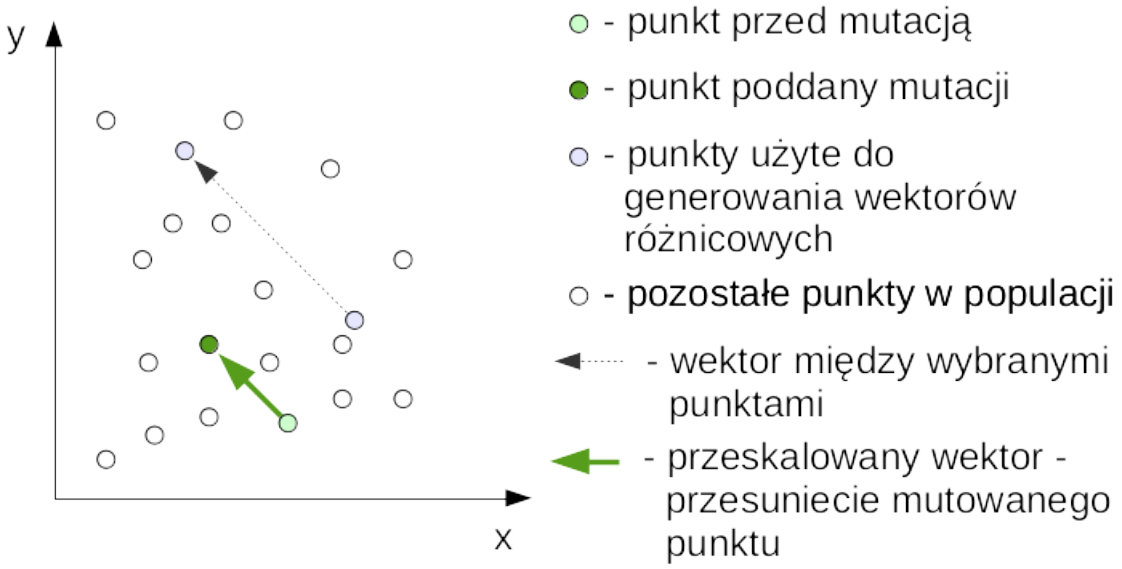
Eksploatacja – to proces badania obszaru przestrzeni przeszukiwań charakteryzującego się niską średnią wartością funkcji przystosowania. Pojęcie to opisuje proces poszukiwania minimum lokalnego przez punkty zawierające się w jego obszarze przyciągania.

Uzyskanie właściwego balansu między tymi dwoma aspektami działania algorytmów ewolucyjnych jest kluczowe dla powodzenia algorytmu ewolucyjnego. Efekt sterowania siłą dwóch charakterystyk przeszukiwań otrzymuje się, przez odpowiednie definiowanie operatorów oraz ich parametrów. W szczególności, zadaniem operatora krzyżowania jest przede wszystkim prowadzenie populacji w kierunku eksploatacji, natomiast zadaniem

mutacji jest generowanie punktów poza w danej chwili badanym obszarem przyciągania - a więc eksploracja przestrzeni przeszukiwań. Istotnym elementem regulującym tę równowagę jest także operator selekcji, której charakter znacząco wpływa na przeżywalność i siłę przyciągania osobników pionierów (wygenerowanych losowo, nielicznych punktów, w nowym, atrakcyjnym obszarze przyciągania).

2.2.2 Klasyczny algorytm ewolucji różnicowej

Szczególnym przypadkiem algorytmu należącego do grupy algorytmów ewolucyjnych są algorytmy ewolucji różnicowej (ang. *differential evolution*, DE). Algorytmy te, zbudowane są w sposób zakładający ich wykorzystanie do przeszukiwania przestrzeni ciągłej co jest zgodne z przyjętą we wstępie definicją problemu. Ich specyfika zakłada wykorzystanie do operatora mutacji odpowiednio przeskalowanych wektorów przesunięć wygenerowanych z bieżącej populacji punktów (Rysunek 2.2) [1].



Rysunek 2.2: Schemat działania operatora mutacji w ewolucji różnicowej na przykładzie przestrzeni dwuwymiarowej.

Zgodnie z tą zasadą schemat ewolucyjny w populacji o indeksie t – $\mathbf{X^t}$, składa się z następujących kroków:

Generowanie mutantów :

Ten krok polega na wygenerowaniu z wektora osobników mutantów $\mathbf{V^t}$ o długości równej wielkości populacji - tak by każdemu osobnikowi $\mathbf{x_i^t}$ odpowiadał osobnik mutant $\mathbf{v_i^t}$. Do tego celu wykorzystywane są wybrane z populacji punkty - punkt mutowany oraz punkty wymagane do wygenerowania wektorów różnicowych. W najbardziej klasycznym wariancie algorytmu wszystkie te punkty są dla każdego i generowane losowo [1] (oznaczamy r_1, r_2, r_3). W opisanym wariancie $\mathbf{v_i^t}$ wyznaczane jest jako:

$$\mathbf{v_i^t} = \mathbf{r_1} + F \cdot (\mathbf{r_2} - \mathbf{r_3}) \tag{2.1}$$

Gdzie F to rzeczywistoliczbowy współczynnik skalujący najczęściej należący do przedziału $[0, 1]$. Literatura [1] podaje również że w podstawowym wariancie ewolucji różnicowej zachodzi:

$$\mathbf{x_1^t} \neq \mathbf{r_1} \neq \mathbf{r_3} \neq \mathbf{r_3} \tag{2.2}$$

nie jest to jednak warunek wymagany dla każdego algorytmu tej klasy.

Krzyżowanie :

W tym kroku każdy z osobników w populacji $\mathbf{X^t}$ jest krzyżowany z odpowiadającym mu osobnikiem z populacji mutantów $\mathbf{V^t}$ w celu wyłonienia osobnika dziecka $\mathbf{z_i^t}$ dla każdego osobnika $\mathbf{x_i^t}$. Choć do krzyżowania można stosować wiele różnych operatorów, najczęściej wybierane jest krzyżowanie wymieniające z różnym rozkładem prawdopodobieństwa. W najprostszym wypadku dwumianowy wymieniający operator krzyżujący (ang. *binominal crossover*) może być zdefiniowany następująco:

$$z_{i,j}^t = \begin{cases} v_{i,j}^t & \text{gdy } r \leqslant Cr \\ x_{i,j}^t & \text{w przeciwnym wypadku} \end{cases} \tag{2.3}$$

Gdzie:

- r – zmienna losowa o rozkładzie jednostajnym na przedziale [0, 1]
- Cr – współczynnik krzyżowania, parametr algorytmu z dziedziny [0, 1]

Wartym odnotowania jest konieczność zapewnienia by do każdego osobnika dziecka $\mathbf{z_i^t}$ trafiła co najmniej jedna wartość z osobnika mutanta, a więc by zachodziło $\mathbf{z_i^t} \neq \mathbf{x_i^t}$ [1]. Ze względu na prostotę zapisu element ten zaniedbano jednak w powyższej definicji.

Selekcja :

Ostatnim elementem podstawowej strategii ewolucyjnej dla algorytmów ewolucji różnicowej jest wybór osobników spośród par $(\mathbf{x_i^t}, \mathbf{z_i^t})$. Wybór ten w typowym przypadku dokonywany jest wg. następującej formuły:

$$\mathbf{x_i^{t+1}} = \begin{cases} \mathbf{v_i^t} & \text{gdy } f(v_i^t) \leqslant f(x_i^t) \\ \mathbf{x_i^t} & \text{w przeciwnym wypadku} \end{cases} \tag{2.4}$$

Tego rodzaju podejście stanowi o elitarności wyboru osobników do kolejnych pokoleń, co powoduje zwiększenie możliwości eksploatacyjnych algorytmu kosztem jego możliwości eksploracyjnych.

Działanie algorytmu ewolucji różnicowej zdeterminowane jest przez zestaw operatorów (opisane niżej) jak również podstawowe parametry:

- μ - liczebność populacji - określa ile osobników zostanie stworzonych we wstępnej fazie algorytmu i ile z nich przetrwa na końcu każdego pokolenia.

- Cr - współczynnik krzyżowania określający (w przypadku wariantu bin) prawdopodobieństwo wybrania każdego z elementów osobnika mutanta do osobnika dziecka, jak w równaniu 2.3.

- F - współczynnik skalujący wektor przesunięcia punktu w populacji.

Podsumowując działanie klasycznego algorytmu ewolucji różnicowej można można posłużyć się poniższym pseudokodem, operującym na wprowadzonych w rozdziałach 2.1.1 i 2.2.2 symbolach:

Algorithm 2 Klasyczny algorytm ewolucji różnicowej – *DE/rand/1/bin*

```
1: function EwolucjaRoznicowa(S, U, f, μ, Cr, F, μ)
2:     X = S
3:     while !stop do
4:         V ← Mutacja(X, μ, U, F)
5:         Z ← Krzyzowanie(X, V, μ, U, Cr)
6:         X ← Selekcja(Z, X, μ, f)
7:     end while
8:     return argmin_{x∈X} f(x)
9: end function
10:
11: function Mutacja(X, μ, U, F)
12:     for all i ∈ 1:μ do
13:         V[i] ← X[U_μ] + F·(X[U_μ] - X[U_μ])
14:     end for
15:     return V
16: end function
17:
18: function Krzyzowanie(X, V, μ, U, Cr)
19:     for all i ∈ 1:μ do
20:         for all j ∈ 1:D do
21:             if Cr < U_u then
22:                 Z[i][j] ← V[i][j]
23:             else
24:                 Z[i][j] ← X[i][j]
25:             end if
26:         end for
27:     end for
28:     return return Z
29: end function
30:
31: function Selekcja(Z, X, μ, f)
32:     for all i ∈ 1:μ do
33:         if f(Z) ⩽ f(X) then
34:             X_new[i] ← Z[i]
35:         else
36:             X_new[i] ← X[i]
37:         end if
38:     end for
39:     return X_new
40: end function
```

Dla uproszczenia zapisu jako U_μ oznaczono liczbę losową wygenerowaną z przestrzeni U, z równym prawdopodobieństwem ze zbioru $[1, \mu] \cap \mathbb{N}$. Symbolem U_u oznaczono natomiast liczbę losową wygenerowaną z przestrzeni U z rozkładem $U(0, 1)$.

2.2.3 DE a klasyczne strategie ewolucyjne

Podstawową różnicą w stosunku do klasycznych strategii ewolucyjnych jest nietypowy schemat mutacji i ścisły dobór par poddawanych krzyżowaniu tak, by zawsze stanowiły one kombinację mutanta i osobnika z populacji podstawowej.

Ponadto, ważnym elementem tego algorytmu jest także fakt, że wykorzystuje on zawartą w populacji informację o uzyskanych dotąd przesłankach na temat kształtu badanej funkcji. W klasycznych algorytmach ewolucyjnych, sposób rozłożenia populacji w przestrzeni determinuje typowo jedynie punkty bazowe dla nowej populacji. W przypadku ewolucji różnicowej, nowe punkty generowane z populacji podlegają wyciągnięciu w kierunkach, w których przeszukiwanie przyniosło lepsze rezultaty. Pozwala to na szybsze poruszanie się populacji w adaptującym się do bieżącego krajobrazu kierunku i prędkości.

Warto także zwrócić uwagę, że algorytmy ewolucji różnicowej mają zwykle bardziej elitarny charakter selekcji niż klasyczne algorytmy ewolucyjne, powoduje to częstszą zbieżność populacji do pojedynczych punktów w przestrzeni a także przyspiesza przejście algorytmu w fazę ekspoatacji. Fakt ten dodatkowo potęguje typowy wybór wartości $F \in [0, 1]$, który powoduje geometryczne zmniejszanie się średniej długości wektora różnicowego wraz z upływem pokoleń, w przypadku przejścia algorytmu w fazę eksploatacji pojedynczego punktu.

Co do zasady algorytmy ewolucji różnicowej stanowią cenny punkt wyjścia dla dalszych rozważań, ponieważ już przy bardzo prostych operatorach pozwalają na uzyskiwanie bardzo dobrych właściwości eksporacyjnych i eksploatacyjnych.

2.2.4 Podstawowe warianty ewolucji różnicowej

Ilość par różnicowanych

Opisany wyżej standardowy schemat ewolucji różnicowej, jest wersją podstawową, której zachowanie można na wiele sposobów zmodyfikować, pozostając nadal w tej klasie algorytmów. Podstawowa różnica jaką mogą charakteryzować się poszczególne instancje DE, to ilość par wykorzystywanych do mutacji różnicowej. W ewolucji różnicowej uogólnionej ze względu na ilość różnicowanych par, krok mutacji różnicowej (2.1) zostaje zamieniony na [13]:

$$\mathbf{v_i^t} = \mathbf{r_1} + \sum_{d=1}^{D} F_d \cdot (\mathbf{r_{j,1}} - \mathbf{r_{j,2}}) \tag{2.5}$$

gdzie jako D oznaczono ilość elementów poddawanych mutacji różnicowej.

W praktyce, najczęściej stosuje się $D = 1$ lub $D = 2$ i $F_1 = F_2$, ponieważ nie znaleziono przesłanek by dalsze zwiększenie wartości D powodowało poprawę jakości otrzymywanych przy użyciu algorytmu wyników [13].

Schemat mutacji

Najistotniejszą z punktu widzenia charakterystyki DE możliwością zmodyfikowania zachowania algorytmu jest możliwość, zamiany schematu mutacji. Podstawowy schemat mutacji (*rand*) zakłada losowanie osobników $\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}$ i postępowanie wg schematu 2.1.

W ramach licznych badań powstała jednak znaczna ilość wariantów bardziej wysublimowanych. Najpopularniejsze w literaturze i najistotniejsze ze względu na zawartość pracy schematy mutacji to:

rand [6, 1] – wariant podstawowy

best [13] – ten schemat zakłada zmianę schematu 2.1 na:

$$\mathbf{v_i^t} = \mathbf{x_{best}^t} + F \cdot (\mathbf{r_1} - \mathbf{r_2}) \tag{2.6}$$

gdzie $\mathbf{x_{best}^t}$ to osobnik z populacji t, który uzyskał najlepszy wynik oceny funkcją f. Schemat ten, promuje poszukiwanie rozwiązania wokół najlepszego dotychczas znalezionego osobnika, wedle rozrzutu bieżącej populacji. Pozwala to na adaptacyjne skupienie się populacji wokół lokalnego optimum i eksploatowanie jego możliwości, kosztem możliwości eksploracyjnych algorytmu.

mid [14] – ten schemat zakłada zmianę schematu 2.1 na:

$$\mathbf{v_i^t} = \mathbf{x_{mid}^t} + F \cdot (\mathbf{r_1} - \mathbf{r_2}) \tag{2.7}$$

gdzie $\mathbf{x_{mid}^t}$ to punkt środkowy populacji t zdefiniowany jako:

$$x_{mid,j}^t = \frac{1}{n} \sum_{i=1}^{n} x_{i,j} \tag{2.8}$$

gdzie n to wielkość populacji. Schemat ten został opracowany niedawno i czerpie on swoją inspirację z podejrzenia o informacji niesionej przez środek populacji. Koncepcja ta zostanie rozwinięta w rozdziale 2.2.6.

current-to-rand [13] – ten schemat zakłada zmianę schematu 2.1 na:

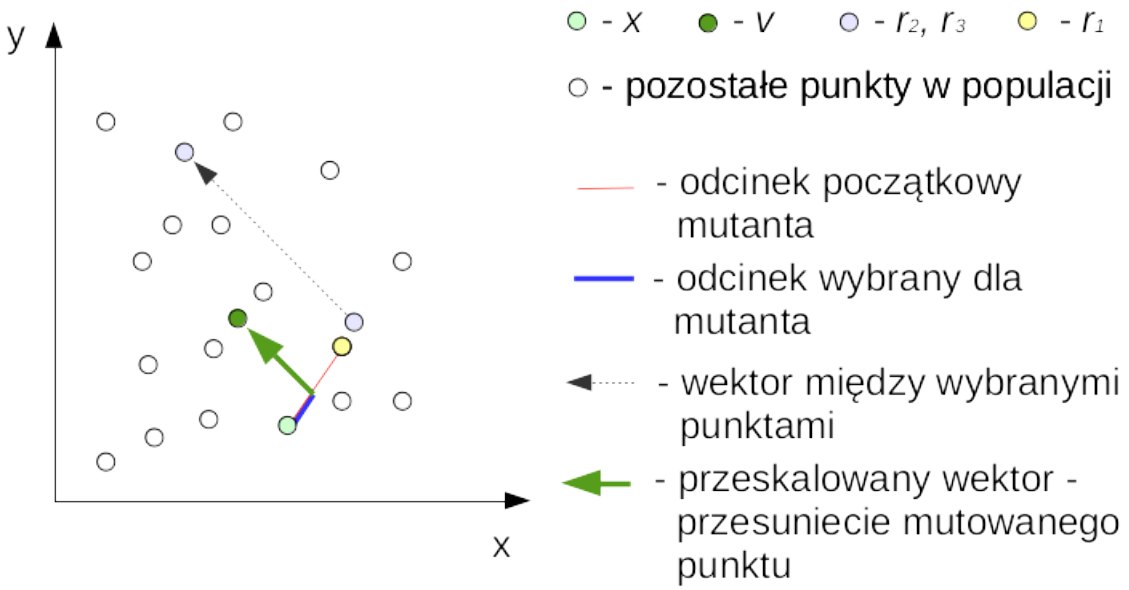
$$\mathbf{v_i^t} = K \cdot \mathbf{x_i^t} + (1 - K) \cdot \mathbf{r_1} + F \cdot (\mathbf{r_2} - \mathbf{r_3}) \tag{2.9}$$

W tym schemacie do standardowego schematu *rand* dołącza się element wiążący mutanta ze sparowanym z nim osobnikiem z populacji podstawowej $\mathbf{X}$. Wykorzystuje się w tym celu zmienną $K \in [0, 1]$, która stanowi wagę dla osobnika z populacji podstawowej $\mathbf{x_i^t}$. W praktyce oznacza to, że mutant $\mathbf{v_i^t}$ powstaje przez przesunięcie punktu leżącego między $\mathbf{x_i^t}$ a losowo wybranym punktem z populacji $\mathbf{r_1}$ o przeskalowany współczynnikiem F wektor między dwoma losowymi osobnikami z populacji (Rysunek 2.3).

current-to-best [13] – ten schemat zakłada zmianę schematu 2.1 na:

$$\mathbf{v_i^t} = K \cdot \mathbf{x_i^t} + (1 - K) \cdot \mathbf{x_{best}^t} + F \cdot (\mathbf{r_1} - \mathbf{r_2}) \tag{2.10}$$

Schemat ten jest analogiczny do schematu *current-to-rand*, jednak oparty o przesłankę związaną z najlepszym punktem populacji. Procedura ta zakłada przesuwanie wygenerowanym wektorem punktu leżącego między odpowiadającemu danemu mutantowi osobnikiem z populacji podstawowej a punktem o najlepszej dotąd wartości funkcji optymalizowanej.



Rysunek 2.3: Schemat mutacji różnicowej *current-to-rand* na przykładzie przestrzeni dwuwymiarowej. Jeśli czerwony odcinek ma długość l, to odcinek niebieski ma długość $l \cdot K$

rand-to-best [13] – ten schemat zakłada zmianę schematu 2.1 na:

$$\mathbf{v_i^t} = K \cdot \mathbf{x_{best}^t} + (1 - K) \cdot \mathbf{r_1} + F \cdot (\mathbf{r_2} - \mathbf{r_3}) \tag{2.11}$$

Schemat ten, podobnie jak dwa poprzednie, opiera się o przypuszczenie, że istnieje możliwość poprawienia skuteczności algorytmu przez zwiększenie rozrzutu punktów początkowych w przestrzeni przeszukiwań. W tym jednak wypadku, mamy do czynienia z połączeniem koncepcji znanych ze schematów *rand* i *best* w celu (w miarę możliwości) uzyskania najlepszych cech obu z nich. Warunki eksploracyjne tego schematu są poprawione względem schematu *best*, ze względu na możliwość odciągania punktów od lokalnego optimum przez osobniki znajdujące się w okolicy innego optimum. Zarazem, w przypadku dotarcia przez algorytm do fazy czysto eksploatacyjnej (skupienie wokół jednego minimum całej populacji), możliwości eksploatacyjne schematu *best* zostają zachowane.

current-to-pbest [9] – ten schemat zakłada zmianę schematu 2.1 na:

$$\mathbf{v_i^t} = \mathbf{x_i^t} + F \cdot (\mathbf{x_{r_{pbest}}^t} - x_i^t) + F \cdot (\mathbf{r_1} - \mathbf{r_2}) \tag{2.12}$$

gdzie $\mathbf{x_{r_{pbest}}^t}$ to osobnik wybrany losowo spośród $p \cdot n$ osobników o najniższej wartości funkcji optymalizowanej, $p \in (0, 1]$. W niektórych pracach [10] proponowane jest dolne ograniczenie wartości p jako $p_{min} = \frac{2}{\mu}$, co oznacza, że osobnik $\mathbf{x_{r_{pbest}}^t}$ jest zawsze losowany spośród co najmniej dwóch najlepszych osobników.

Rozwiązanie to ma za zadanie dołączyć do zalet schematu *current-to-best* dodatkowe rozrzucenie osobników w przeszukiwanej przestrzeni i osłabienie siły przyciągania jednego najmocniejszego osobnika, a więc zmniejszenie elitaryzmu całego algorytmu.

Schemat krzyżowania

Krzyżowanie w klasycznym algorytmie ewolucji różnicowej, pełni kluczową rolę – zapewnia powiązanie osobników mutantów z populacji $\mathbf{V}$ w pary z osobnikami z populacji bazowej $\mathbf{X}$. W ten sposób tworzy przyczynek do wykorzystywania par osobników do selekcji przez ich bezpośrednie porównanie. W bardziej wysublimowanych schematach mutacji opisanych w poprzednim podrozdziale, osobnik z bieżącej populacji bywa wykorzystywany już na etapie mutacji. Nie jest to regułą ani założeniem niezbędnym dla skonstruowania poprawnych operatorów dla ewolucji różnicowej, stąd też nawet w odbiegających od podstawowego wariantu wersjach algorytmów ewolucji różnicowej, krzyżowanie jest niezbędne dla poprawnego zachowania algorytmu.

Typowo w literaturze spotyka się następujące schematy krzyżowania:

bin – podstawowy schemat krzyżowania zgodny z równością 2.3.

exp – schemat krzyżowania zamieniający równość 2.3, na procedurę:

Algorithm 3 Schemat krzyżowania **exp**

```
1: j ← 1
2: while r < Cr ∧ j ⩽ D do
3:     z^t_{i,j} ← v^t_{i,j}
4:     j ← j + 1
5: end while
6: while j < D do
7:     z^t_{i,j} ← x^t_{i,j}
8: end while
```

Tego rodzaju krzyżowanie powoduje dążący (gdy $D \to \infty$) do wykładniczego rozkład położenia pierwszego elementu wektora pochodzącego z osobnika $\mathbf{x_i^t}$. Rozkład ten nie traktuje wszystkich pozycji w wektorze osobników identycznie, ponieważ dalsze pozycje w wektorze będą rzadziej zastępowane przez wartości z osobników mutantów. W związku z tym stosuje się także wariant, który wybiera ilość elementów (n_x) pochodzących z osobnika rodzica zgodnie z opisanym rozkładem a następnie wybiera z równym prawdopodobieństwem n_x elementów wektora, które będą pochodziły z osobnika $\mathbf{x_i^t}$.

Notacja

Mając na względzie, że typowo algorytmy ewolucji różnicowej rozróżniane są między sobą na podstawie trzech wyżej wymienionych zmiennych - a więc: ilości par różnicowanych, schematu mutacji różnicowej i schematu krzyżowania, wprowadzono notację pozwalającą w sposób zwięzły opisać konkretną instancję algorytmu z grupy algorytmów ewolucji różnicowej [13]:

$$\mathbf{DE/M/D/C}$$

gdzie:

- **M** - schemat mutacji (np. *rand*, *current-to-pbest*).
- **D** - ilość par różnicowanych (zwykle *1* lub *2*).
- **C** - schemat krzyżowania (np. *bin*, *exp*).

I tak na przykład klasyczny algorytm ewolucji różnicowej (2.2.2) wg tej notacji byłby opisany jako: DE/*rand*/*1*/*bin*. Notacja ta będzie wykorzystywana w dalszej części pracy.

2.2.5 Strategie adaptacji

Dla każdego z opisanych schematów istnieją prace opisujące proces adaptacji parametrów na podstawie dotychczas uzyskiwanych rezultatów [15, 9, 10]. Nierzadko proces adaptacji wykorzystanie informacje historii H o szerokości przekraczającej wielkość pojedynczego pokolenia stąd metody adaptacji mogą być traktowane jako jedna z prób poprawy działania algorytmów ewolucji różnicowej przez poszerzenie okna archiwum.

Opublikowane prace jednoznacznie wskazują, że proces adaptacji znacząco poprawia uzyskiwane w ramach ewolucji różnicowej wyniki. Stanowi to kolejną przesłankę pozwalającą oczekiwać poprawy rezultatów przez wykorzystanie szerszego okna historii.

Dodatkowo należy odnotować, że adaptacyjne algorytmy ewolucyjne otrzymują najlepsze rezultaty w testach typu *black-box* (spośród algorytmów klasy DE) [8]. Fakt ten staje się przyczynkiem do tego by sprawdzić, czy proponowany algorytm *NADE* będzie utrzymywał swoją charakterystykę jakości także względem tych modyfikacji.

2.2.6 Punkt środkowy populacji

Niedawno w badaniach nad algorytmami ewolucji różnicowej [14] pojawiła się koncepcja wykorzystania inforamcji niesionej przez punkt środkowy populacji do poprawy sposobu działania algorytmu. W pracy [14] podjęto próbę zbudowania nowego schematu mutacji, w którym punktem bazowym jest punkt środkowy populacji (DE/*mid*/k). Wnioskiem z tej pracy było stwierdzenie, iż dla większości funkcji zwłaszcza w wielowymiarowych przestrzeniach, wybór strategii *mid* poprawia w sposób statystycznie istotny działanie algorytmu ewolucji różnicowej. W pracy zwrócono także uwagę, że punkt środkowy nie przynosi znaczącej poprawy lub wręcz jest mylący dla zadania, w przypadku pewnych funkcji. Jako przykład takiej funkcji wymieniono funkcję Weierstrassa wchodzącą w skład zestawu testowego CEC2013 (3.2.5).

Praca ta jednoznacznie wskazała, że informacja zawarta w punkcie środkowym dla wielu problemów może być użyteczna. Intuicja podpowiada, że w szczególności w fazie eksploatacji okolic lokalnego optimum punkt środkowy może być o wartości wyższej niż najlepsza. Taka własność punktu środkowego stanowiła by wytłumaczenie przewagi algorytmu DE/*mid*/*1* nad DE/*best*/*1*. Choć początkowo zamierzano włączyć w skład pracy badanie dotyczące tej tezy, to ze względu na dużą ilość innych tez wymagających sprawdzenia w ostatecznej wersji rezygnowano z badania tej tezy.

2.2.7 Metoda uwzględniania ograniczeń

Praca [16], pokazała że metoda uwzględniania ograniczeń przeszukiwanej przestrzeni ma istotny wpływ na jakość rezultatów uzyskiwanych przez algorytm ewolucji różnicowej.

Praca wyszczególnia wiele typów dopasowywania punktów mutantów do zakresu:

Reinicjalizacja (ang. *Reinitialization* – Powszechnie używana w algorytmach ewolucji różnicowej strategia [16], która polega na zamianie punktu, który znalazł się poza ograniczeniami, na losowo wybrany (z rozkładem jednostajnym) punkt mieszczący się w ograniczeniach.

Ponowne próbkowanie (ang. *Resampling*) – Metoda, która okazała się bardzo skuteczna, w szczególności w przypadku dużej ilości punktów nie mieszczących się w ograniczeniach. Polega ona na powtarzaniu procedury tworzenia mutanta na podstawie losowych wektorów różnicowych, do czasu osiągnięcia osobnika, który mieści się w ograniczeniach.

Odbijanie (ang. *Reflection*) – Metoda polegająca na odbijaniu punktu względem wszystkich przekroczonych linii ograniczeń aż do czasu, gdy punkt znajdzie się wewnątrz przeszukiwanego obszaru przestrzeni.

Rzutowanie (ang. *Projection*) – Metoda ta polega na rzutowaniu prostopadłym punktu na wszystkie przekroczone ograniczenia.

Zawijanie (ang. *Wrapping*) – w tej metodzie, zakłada się, że obszar przeszukiwań jest skończonym wycinkiem hiperpłaszczyzny. Mając to na względzie, nakłada się tą skończoną podprzestrzeń na hipertorusa w ten sposób zyskując przestrzeń posiadającą wyłącznie punkty dopuszczalne a opisującą cały zbiór $\mathbb{R}^D$. Wadą tego rozwiązania jest fakt, że jeśli funkcja f jest ciągła, to jej w nowej przestrzeni jej ciągłość może zostać zerwana.

Zachowywanie (ang. *Conservatism*) – metoda ta polega na odrzuceniu dziecka z_i jeśli znajdzie się ono poza obszarem przeszukiwania i w czasie selekcji wybranie osobnika x_i. Podejście to jest równoważne nadaniu funkcji kary za przejście poza granicę obszaru przeszukiwania, równej ∞ w każdym punkcie poza ograniczeniami.

Wyniki otrzymane w ramach [16] sugerują, że najlepiej sprawdzają się metodą uwzględniania ograniczeń jest *resampling*. Poważnymi wadami tego rozwiązania jest znaczne skomplikowanie struktury algorytmu, ze względu na konieczne nawroty, a także koszta wynikające z dodatkowych obliczeń (zwłaszcza w przypadku dużego odsetka odrzucanych punktów). Z tych powodów zdecydowano, by odrzucić tę metodę i na czas pracy skorzystać z jednej z dwóch znacznie prostszych a dających podobne, obiecujące rezultaty – *odbijania* i *rzutowania*. Spośród tych dwóch wariantów arbitralnie wybrano *odbijanie* i ten sposób uwzględniania ograniczeń będzie wykorzystywany przy wszystkich testach w ramach pracy.

2.2.8 Archiwum

W literaturze spotyka się niezliczone próby jawnego lub niejawnego wykorzystania dodatkowego archiwum w algorytmach ewolucji różnicowej. Poniżej opisano podejście do archiwum w kilku pracach, które w jawny sposób powołują się na poprawę jakości działania algorytmu ewolucji różnicowej dzięki zastosowaniu archiwum. Niestety żadna z tych prac nie bada skuteczności archiwum w oderwaniu od innych metod usprawniania algorytmów.

Archiwum w ADE [3, 17]

W pracy [3] opracowano algorytm *ADE* (ang. *Archived Differential Evolution*) wykorzystujący archiwum mające przechowywać informacje o najlepszych osobnikach we wszystkich następujących po sobie pokoleniach. Archiwum tego rodzaju pełniło w algorytmie trzy funkcje:

Wskaźnika wysycenia konkretnej populacji – przyjęto tu założenie, że jeśli najlepszy punkt w populacji nie zmienił się od pewnej definiowalnej ilości pokoleń, należy przyjąć, że obecna populacja nie ma perspektyw na odnalezienie znacząco lepszych punktów. W algorytmie *ADE* był to jeden z warunków wyzwalających adaptację populacji.

Wskaźnika zbliżenia się populacji do lokalnego optimum – w algorytmie podjęto próbę wykorzystania informacji zawartych w punkcie środkowym populacji. Jednym z warunków wyzwalających adaptację populacji, było bowiem zmniejszenie odległości euklidesowej:

$$d = \sqrt{\sum_{i=1}^{D}(x_{best,i} - x_{mean,i})^2} \tag{2.13}$$

gdzie $\mathbf{x_{mean}}$ stanowi punkt środkowy populacji definiowany jako:

$$x_{mean,i} = \frac{1}{\mu} \cdot \sum_{j=1}^{\mu} x_{j,i} \tag{2.14}$$

Wspomniany warunek wyrażony jest jako:

$$d \leqslant \epsilon \tag{2.15}$$

gdzie ϵ to parametr algorytmu określający graniczną odległość między wspomnianymi punktami. Jeśli warunek jest spełniony, następuje adaptacja populacji.

Punktu odniesienia dla adaptacji populacji – w algorytmie w wyniku kilku warunków następowała adaptacja popualacji mająca na celu poprawę właściwości eksporacyjnych algorytmu. Adaptacja odbywała się wg. formuły zmieniającej osobniki w populacji [3]:

$$x_{i,j,G} = \lambda \cdot x_{i,Best,A} + (1 - \lambda) \cdot (x_{i,Best,A} - x_{i,j,G-1}) \tag{2.16}$$

gdzie $\lambda \sim U[0, 1]$ natomiast $x_{i,j,G}$ oznacza wartość i elementu wektora osobnika j w populacji G, a $x_{i,Best,A}$ oznacza wartość i elementu wektora najlepszego osobnika w archiwum.

Poważną wadą samego algorytmu jest wykorzystywanie informacji wykraczających poza te pozyskiwalne z definicji (rozdział 1.1). W szczególności algorytm wykorzystuje trudności opisane w rozdziale 3.2.4 wykorzystując parametr ϵ a nawet sugerując jego uniwersalną arbitralną wartość 10^{-3}. Tego rodzaju podejście przeczy założeniom opisywanego problemu.

Algorytm *ADE* został przez autorów oceniony pozytywnie w kontekście badanych problemów. Wykorzystuje on jednak archiwum w niewielkim stopniu a ponadto nie bada tego

zastosowania w odcięciu od pozostałych technik poprawy jakości działania algorytmu. Sama metoda reinicjalizacji oparta o brak poprawy w kolejnych pokoleniach stanowi przykład przeniesienia znanych z innych algorytmów technik na grunt ewolucji różnicowej.

Archiwum w CDDE_Ar [18]

Algorytm *CDDE_Ar* operaty jest o idee podzielenia populacji na mniejsze grupy (klastrowanie) przy użyciu algorytmu *k-średnich* [19]. Tego rodzaju grupy, traktowane są oddzielnie i podlegają złożonym procesom adaptacji, usuwania i kreacji opisanych szczegółowo w [18].

Choć modyfikacja ta znacząco odbiega od podstawowego wariantu ewolucji różnicowej *DE/rand/1/bin*, to jednak istotnym z punktu jej widzenie jest wykorzystanie archiwum, w celu selektywnego przechowywania informacji o dobrych jakościowo punktach odnalezionych przez nieistniejące już klastry. W tym podejściu wykorzystuje się znane i dobre jakościowo punkty do wzbogacania nowo tworzonych lub adaptowanych klastrów.

Algorytm *CDDE_Ar*, nie stanowi podstawy dla tej pracy, jednak pokazuje potencjał, drzemiący w historycznych osobnikach, który w podstawowym wariancie ewolucji różnicowej jest zaniedbywany.

Archiwum w innych algorytmach

Wiele innych algorytmów ewolucji różnicowej wdraża lub jest modyfikowana o elementy związane z zarządzanym archiwum. W szczególności klasyczne archiwum rozszerzające ilość elementów, z których generowane są wektory różnicowe, było przedmiotem badań przy algorytmach z rodziny *JADE* [9, 10, 20] opisanych szczegółowo w rozdziale 2.2.9.

Wnioski z dotychczasowych prac

Liczność algorytmów ewolucji różnicowej wykorzystujących koncepcję archiwum do różnych celów (w szczególności do generowania wektorów różnicowych), sugeruje, że wykorzystanie archiwum pozytywnie wpływa na działanie algorytmów. Można także zauważyć, że archiwum w algorytmie ewolucji różnicowej, jest cennym źródłem dodatkowych informacji, wspierających adaptację i inne zaawansowane procesy w ich obrębie. Warto zwrócić uwagę, że wspomniane prace traktują archiwum łącznie z innymi koncepcjami poprawy działania algorytmów ewolucji różnicowej. Nie można w związku z tym wysnuwać wniosków na temat niezależności wpływu wykorzystania archiwum od innych technik usprawniania algorytmów. Tego rodzaju badania są jednym z przedmiotów tej pracy.

2.2.9 Zaawansowne modyfikacje

Do celów pracy wybrano dwie modyfikacje stosujące adaptację parametrów (rozdział 2.2.5), jak również powiększone okno historii (rozdział 2.2.8). Algorytm *SHADE* [10] został wybrany, jako najlepiej sprawdzająca się modyfikacja ewolucji różnicowej [8]. Algorytm *JADE* [9] został wybrany jako pierwowzór *SHADE*, który opiera się o prostsze acz ideologicznie podobne koncepcje. Specyfikę obu algorytmów opisano poniżej:

JADE [9] – algorytm *JADE* jest algorytmem ewolucji różnicowej, wykorzystującym schemat mutacji *current-to-pbest* (zalecane $p \in [0.05, 0.2]$). Algorytm cechuje także indywidualna wartość współczynnika F oraz Cr dla każdego mutowanego i krzyżowanego osobnika w każdej kolejnej iteracji (oznaczenie F_i^t, Cr_i^t lub po prostu F_i, Cr_i). Taka mnogość prób wykorzystywana jest, do adaptacji tych współczynników. Wartości współczynników zmieniane są na podstawie sukcesów bądź porażek przy stosowaniu ich konkretnych wartości. Ponadto, algorytm ten wykorzystuje archiwum punktów do generowania wektorów różnicowych. Stanowi więc on dalekie rozszerzenie podstawowej idei ewolucji różnicowej. Poniżej opisano kluczowe modyfikacje względem *DE/rand/1/bin* występujące w algorytmie.

Archiwum

W algorytmie *JADE* zastosowano archiwum o wielkości dwóch populacji – w postaci populacji bieżącej $\mathbf{X}$ i archiwum $\mathbf{A}$. Do archiwum trafiają punkty z populacji $\mathbf{X}$ zastąpione przez osobniki dzieci z populacji $\mathbf{Z}$. Ograniczenie wielkości archiwum $|A| \leqslant \mu$ jest wymuszane na końcu każdej iteracji algorytmu, przez losowe usuwanie osobników z archiwum – aż do uzyskania jego porządanej wielkości. Archiwum wykorzystywane jest w połączeniu z bieżącą populacją $\mathbf{X}$ wyłącznie do generowania wektorów różnicowych.

Adaptacja F_i

Mechanizm adaptacji współczynników F_i opiera się o generowanie losowych wartości wokół zaadaptowanej wartości F_{mid} z rozkładem $F \sim Cauchy(F_{mid}, 0.1)$ z ograniczeniem wartości $F_i \in [0, 1]$. Dla wartości poniżej tego zakresu wartość F_i jest ponownie generowana. Dla wartości powyżej tego zakresu przyjmowane jest $F_i = 1$.

Model adaptacji wymaga wprowadzenia pojęcia sukcesu.

Definition 4. *Sukcesem próby i w iteracji t nazywamy sytuację, gdy* $\mathbf{x_i^{t+1}} = \mathbf{z_i^t}$.

Oznaczmy także jako S^t zbiór prób z iteracji t, które zakończyły się sukcesem.

Wartość F_{mid} inicjalizowana jest przez wartość 0.5. W kolejnych pokoleniach jest ona zmieniana na średnią ważoną F_{mid} i wartości F_{S^t} zdefiniowanej jako średnia Lehmera z wartości F_i^t prób zakończonych sukcesem:

$$F_{S^t} = \frac{\sum_{s \in S^t} {F_s}^2}{\sum_{s \in S^t} F_s} \tag{2.17}$$

Jako wagi przyjmuje się odpowiednio c i $1 - c$, gdzie $c \in [0, 1]$ jest dodatkowym parametrem algorytmu. Zachodzi więc:

$$F_{mid}^{t+1} = c \cdot F_{mid}^t + (1 - c) \cdot F_{S^t} \tag{2.18}$$

Autorzy [9] sugerują wykorzystanie wartości $c \in [0.05, 0.2]$.

Adaptacja Cr_i

Mechanizm adaptacji współczynników Cr_i w algorytmie *JADE* zbudowany jest na podobnej zasadzie jak mechanizm adaptacji współczynników F_i. Podobnie jak w przypadku F_{mid}, Cr_{mid} jest generowany na podstawie średniej z wartości Cr_i dla sukcesów. Tym razem jednak wykorzystana jest średnia arytmetyczna zamiast średniej Lehmera. Ponadto do generowania wartości Cr_i dla poszczególnych wartości i wykorzystywany jest rozkład normalny zamiast rozkładu cauchiego – $Cr_i \sim N(Cr_{mid}, 0.1)$. Do adaptacji wartości Cr_{mid} wykorzystywana jest ta sama stała c co w przypadku adaptacji F_{mid}.

SHADE [10] – algorytm *JADE* doczekał się wielu modyfikacji i rozwinięć [10, 21, 15, 22, 23, 24, 20]. Spośród nich, wyjątkowo obiecujące rezultaty w ramach testu sprawności *CEC 2013 benchmark* (rozdział 3.2.5) uzyskał algorytm *SHADE*. Autorzy tego algorytmu zauważyli, że ze względu na niedeterministyczny charakter badanych algorytmów pewne wartości adaptowanych parametrów mimo, iż lokalnie przyniosły korzyść, w perspektywie globalnego krajobrazu badanej funkcji nie przynoszą oczekiwanej poprawy. Autorzy wysuwają tezę, iż w związku z tym algorytm jest podatny na tego typu lokalne zaburzenia wartości adaptowanych i nie posiada skutecznego mechanizmu ochrony przed tym zjawiskiem.

Modyfikacja adaptacji względem JADE

Idea stojąca za algorytmem *SHADE* opiera się o silniejsze (niż w przypadku formuły 2.18) wygładzanie lokalnych zaburzeń adaptowanych wartości F_i^t oraz Cr_i^t. By uzyskać ten efekt, autorzy zaproponowali archiwum wartości adaptowanych M_F oraz M_{Cr} o parametryzowalnej wielkości $H \in \mathbb{N}$ (zalecane $H \in [30, 100] \cap \mathbb{N}$). Oba archiwa wypełniane są początkowo stałą wartością podstawową ($F_{init} = Cr_{init} = 0.5$), a następnie w każdej iteracji najstarsza wartość w archiwum zastępowana jest wartością uzyskaną na podstawie sukcesów w danej iteracji. W przypadku tego algorytmu wartości te (F_{S^t} oraz Cr_{S^t}) zdefiniowane zostały zgodnie z formułami z [21]:

$$F_{S^t} = \frac{\sum_{k=1}^{|S^t|} w_k \cdot {F_{S^t,k}}^2}{\sum_{k=1}^{|S^t|} w_k \cdot F_{S^t,k}} \tag{2.19}$$

oraz

$$Cr_{S^t} = \sum_{k=1}^{|S^t|} w_k \cdot Cr_{S^t,k} \tag{2.20}$$

gdzie

$$w_k = \frac{\Delta f_k}{\sum_{j=1}^{|S^t|} \Delta f_j} \tag{2.21}$$

oraz

$$\Delta f_k = |f(z_k) - f(x_k)| \tag{2.22}$$

Jak widać mechanizm ten posługuje się podobnymi konstrukcjami matematycznymi jak w przypadku *JADE* jednak dodatkowo bierze pod uwagę wagę, wynikającą ze

stopnia w jakim dana wartość adaptowanego parametru poprawiła wartość funkcji optymalizowanej.

Na początku każdej iteracji podobnie jak w *JADE* wybierana jest indywidualna wartość parametrów dla każdego osobnika. Analogicznie do *JADE* wykorzystywane są rozkłady $Cr_i \sim N(M_{Cr,r_i}, 0.1)$ oraz $F_i \sim Cauchy(M_{F,r_i}, 0.1)$, gdzie $r_i \in [1, H] \cap \mathbb{N}$, to losowo wygenerowany dla każdego osobnika w każdym pokoleniu indeks archiwum. W przypadku gdy wygenerowana wartość $Cr_i \notin [0, 1]$ lub $F_i \notin [0, 1]$, stosowane są procedury naprawcze jak w przypadku algorytmu *JADE*.

Zmienna wartość p w schemacie current-to-pbest

Dodatkową modyfikacją wprowadzoną w ramach algorytmu *SHADE* jest uzmiennienie wartości p wykorzystywanej w schemacie *p-to-best*. Mechanizm ten polega na wygenerowaniu w każdym pokoleniu dla każdego osobnika losowej wartości $p_i \sim U(p_{min}, 0.2)$. Pewną wadą tego rozwiązania jest brak mechanizmu weryfikującego w czasie działania algorytmu, czy sukces danego osobnika nie jest spowodowany wygenerowaną wartością p_i a nie parą Cr_i, F_i. Warto jednak zauważyć, że u podstaw idei tego algorytmu leży jego odporność na tego rodzaju lokalne zaburzenia, w związku z czym można oczekiwać, że tego rodzaju sytuacje będą miały marginalny wpływ na ostateczny rezultat adaptacji. Prawidłowość ta nie została jednak zbadana.

Rozdział 3

Metodyka badań

3.1 Modelowanie statystyczne

Modelowanie statystyczne to dziedzina zajmująca się tworzeniem modeli charakterystyki zmiennych losowych w oparciu o statystyki z próby losowej. W przypadku niedeterministycznych algorytmów optymalizujących, wynik algorytmu jest zmienną losową. Z tego powodu, porównanie pojedynczych wyników różnych algorytmów, nawet w kontekście tego samego problemu niesie bardzo niewielką informację.

Narzędzia modelowania statystycznego pozwalają przy użyciu skończonej liczby próbek zbudować przybliżoną charakterystykę otrzymywanych w kolejnych próbach wyników. Porównanie tego rodzaju charakterystyk niesie już znacznie większą informację, choć zawsze wyznaczaną z pewnym tylko prawdopodobieństwem. W tym rozdziale zostaną opisane metody modelowania statystycznego wykorzystane przy badaniu i porównywaniu algorytmów w ramach pracy.

3.1.1 Testy statystyczne

Testy statystyczne służą do estymacji prawdopodobieństwa prawdziwości pewnej hipotezy dotyczącej losowego modelu na podstawie próby losowej. Innymi słowy testy te, pozwalają stwierdzić z pewnym prawdopodobieństwem, że pewna własność zmiennych losowych zachodzi lub nie zachodzi.

W przypadku testów statystycznych, testowana hipoteza nosi nazwę hipotezy zerowej, a hipoteza uznawana za prawdziwą w przypadku odrzucenia hipotezy zerowej nazywana jest hipotezą alternatywną.

Omawiane testy statystyczne, opierają się o wyznaczenie pewnej statystyki testowej (będącej zmienną losową). Następnie, przy użyciu narzędzi modelowania statystycznego, jest wyznaczane prawdopodobieństwo, że przy prawdziwości hipotezy zerowej, uzyskana wartość statystyki będzie taka sama lub bardziej odległa od oczekiwanej – w ten sposób definiowana jest $p - wartosc$ (poziom istotności). Samo określenie, która wartość jest bardziej odległa od hipotezy zerowej zależy od konstrukcji konkretnych testów.

W przypadku, gdy na podstawie testu chcemy wysnuć (choćby z pewnym prawdopodobieństwem) twierdzenie na temat modelu losowego, często przyjmuje się próg istotności (graniczny poziom istotności), dla którego odrzuca się hipotezę zerową. Najczęściej pojawiającym się w literaturze granicznym istotności jest 0.05 [25], taka wartość będzie przy-

jęta również w tej pracy. Tego rodzaju wartość oznacza, że średnio co dwudziesta próba udowodnienia hipotezy zerowej zostanie niepoprawnie odrzucona mimo jej poprawności. Tego rodzaju zaburzenia nie powinny jednak wpływać na ogólne wnioski.

Testy statystyczne w kontekście algorytmów optymalizacyjnych

W przypadku niedeterministycznych algorytmów optymalizacyjnych, zmienną losową, która może zostać poddana testowaniu statystycznemu i nieść informację na temat jakości algorytmu jest minimalna wartość funkcji optymalizowanej uzyskiwana w kolejnych powtórzeniach algorytmu. Dla celów przykładów w dalszych częściach rozdziału zmienną tę oznaczono $R_{algorytm}$.

Próbki z tej zmiennej zebrane dla różnych algorytmów i tego samego problemu pozwalają na modelowanie i porównywanie między sobie algorytmów w kontekście uzyskiwanych przez nie wyników. To z kolei pozwala na wysnuwanie statystycznych wniosków na temat przewagi algorytmu w kontekście pojedynczego zadania. W rozdziale 3.2 wskazano jak poszerzyć zastosowanie testów statystycznych przy użyciu dodatkowych narzędzi, w celu umożliwienia badania jakości uzyskiwanych przez algorytm wyników w oderwaniu od konkretnych problemów.

3.1.2 Testy parametryczne

Statystyczne testy parametryczne służą do weryfikacji pewnych hipotez w oparciu o próbę oraz znaną postać ogólną dystrybuanty statystyki testowej.

Często przyjmowanym w statystyce rozkładem zmiennej losowej o której nie ma żadnych dodatkowych informacji, jest rozkład normalny. Ze względu na prawo wielkich liczb oraz centralne twierdzenie graniczne [26], rozkład ten występuje bowiem w realnych problemach nadzwyczaj często. Rozkład ten jest więc często dobrym punktem wyjścia dla pierwszych badań nad modelem losowym.

Test T-Studenta dla prób niezależnych

Jednym z testów zakładających normalny rozkład badanych zmiennych losowych, jest test T-Studenta dla prób niezależnych. Test ten, weryfikuje hipotezę zerową o równości średnich rozkładów dwóch zmiennych losowych na podstawie prób z nich pochodzących przy założeniu rozkładu normalnego tychże rozkładów.

Średnia wartość zmiennej $R_{algorytm}$ jest uniwersalnym (choć nie jedynym możliwym) sposobem oceny algorytmu w kontekście problemu. W związku z tym, sprawdzenie hipotezy o nierówności średnich, z dwóch prób pochodzących z różnych algorytmów, pozwala wysnuć poprawny statystycznie wniosek na temat przewagi uzyskiwanych przez algorytm wyników. Wniosek ten, obarczony będzie błędem na poziomie granicznego poziomu istotności.

Z tym testem wiążą się jednak dwa podstawowe problemy:

Założenie o normalności rozkładu – w przypadku generowanych przez różne algorytmy optymalizacyjne wyników prób na zmiennej losowej $R_{algorytm}$, rozkład tej zmiennej zależy zarówno od algorytmu jak i od rozwiązywanego problemu. Założenie o dążeniu rozkładu rzeczywistego $R_{algorytm}$ do rozkładu normalnego jest w

niektórych wypadkach poprawne, jednak istnieją funkcje i algorytmy tego założenia niespełniające. W związku z tym test ten w zaproponowanym zastosowaniu, zdaje się być obarczony nieznanym błędem metodycznym, co wyklucza jego zastosowanie w tych badaniach.

Niestabilność średniej – średnia arytmetyczna jako statystyka próby podlega znacznemu wpływowi wartości odstających. Odpowiedź na pytanie, czy jest to cecha pożądana, zależy ściśle od tego, co jest rzeczywiście obiektem naszych badań. W tym konkretnym wypadku, przy użyciu wybieranej statystyki staramy się dowiedzieć, jak algorytm wypada w typowym przypadku. Wydaje się, że mediana jest statystyką znacznie lepiej pasującą do tego ogólnego opisu. Warto zauważyć, że choć na potrzeby zawartych w pracy badań wybrano medianę, to nie jest to jedyny właściwy wybór i jest on podyktowany wyłącznie intuicją badacza (i dostępnością narzędzi do testowania hipotez).

3.1.3 Testy nieparametryczne

W odróżnieniu od testów parametrycznych, w testach nieparametrycznych nie czyni się założeń dotyczących rozkładu badanych zmiennych. W tym wypadku, bada się więc zachowania pewnych statystyk zmiennej losowej w oderwaniu od jej rzeczywistego rozkładu.

Test Wilcoxona dla prób niezależnych

Test Wilcoxona [27] to nieparametryczny test statystyczny, badający hipotezę o równości median dwóch liczb losowych. Porównuje on dwie próby $\mathbf{r_{algorytm1}}$ i $\mathbf{r_{algorytm2}}$ pochodzące ze zmiennych kolejno $R_{algorytm1}$ i $R_{algorytm2}$ przy założeniu:

- Ciągłości rozkładu $R_{algorytm1}$ i $R_{algorytm2}$.
- Zmienne $r_{algorytm1,i}$ i $r_{algorytm2,i}$ są parowane losowo.
- Zmienne $R_{algorytm1}$ i $R_{algorytm2}$ są zmiennymi losowymi na zbiorze ze zdefiniowaną relacją częściowego porządku.

Wszystkie powyższe założenia są spełnione dla badanego przypadku. W związku z tym, można przy użyciu tego testu wysnuwać statystycznie poprawne wnioski dotyczące przewagi konkretnych algorytmów (mediany wyników) w konkretnych problemach, bez obarczania wyników błędem metodycznym. To rozwiązanie wydaje się odpowiadać w pełni wymaganiom poniższych badań, w związku z tym test Wilcoxona dla prób niezależnych został w nich wykorzystany.

3.1.4 Dystrybuanta empiryczna

Dystrybuanta to funkcja zmiennej losowej opisująca prawdopodobieństwo wystąpienia poszczególnych jej wartości. Dla zmiennej losowej L dystrybuanta $D_L(x)$ jest zdefiniowana następująco:

$$D_L(x) = P(L \leqslant x) \tag{3.1}$$

Funkcja ta dokładnie opisuje rozkład zmiennej pozwalając wyznaczyć prawdopodobieństwo wystąpienia wartości x dla dowolnego zbioru. Wykres tej funkcji jest także często wykorzystywana jako wizualizacja rozkładu.

Dystrybuantą empiryczną zmiennej losowej L nazywamy funkcję stanowiącą aproksymację funkcji D_L na podstawie uzyskanych wartości x. Uzyskana w ten sposób krzywa, ma zawsze charakter schodkowy (ze względu na skończoną liczbę prób), jednak wraz ze wzrostem wielkości próby, średni błąd dystrybuanty empirycznej względem dystrybuanty rzeczywistej, dąży do zera. Z tego powodu, dystrybuanta empiryczna, przy licznych próbach, pełni nadal funkcję wizualizacyjną.

Na rysunku **??** zaprezentowano wykres rzeczywistej dystrybuanty rozkładu normalnego i nałożony na niego wykres dystrybuanty empirycznej wygenerowanej ze stu elementowej próby.

3.2 Testy sprawności typu *black-box*

3.2.1 Testy sprawności (ang. *benchmark*)

W rozdziale 3.1.1 wskazano metodę sprawdzenia jakości heurystycznego algorytmu rozwiązującego problem optymalizacji w kontekście konkretnego problemu. Ze względu na charakter optymizowanej funkcji f i kształty przeszukiwanej podprzestrzeni wyznaczanej przez funkcje $g_1, g_2, \ldots g_n$ różne algorytmy zachowują się różnie dla różnych instancji problemu optymalizacyjnego. Z tego powodu ocena algorytmu w kontekście konkretnego problemu nie może być traktowana jako rzetelna algorytmu.

Naturalną konsekwencją powyższego twierdzenia, jest próba skonstruowania mechanizmu umożliwiającego ocenę algorytmów niezależnie od konkretnego algorytmu. Testy sprawności (*benchmark*) stanowią pod wieloma względami odpowiedź na to zapotrzebowanie. Ich podstawowym założeniem, jest teza, że właściwą oceną dla algorytmu średnia jakość jego działania dla wszystkich możliwych problemów danej klasy.

Przyjmując to założenie, autorzy testów tego typu starają się dostarczyć reprezentatywnej próbki problemów. Dzięki temu, oceniając średnią jakość algorytmu w kontekście tych problemów, oczekujemy podobnej średniej jakości co w przypadku wszystkich dopuszczalnych problemów. Oznacza to, że średnia jakość algorytmu w kontekście algorytmów z testu, jest estymatorem średniej jakości algorytmu we wszystkich rozważanych problemach. Zgodnie z pierwotnym założeniem, można więc stwierdzić, że jakość algorytmu w kontekście problemów testów sprawnościowych, stanowi estymator jakości algorytmu.

Dodatkowo, należy zauważyć, że ze względu na reprezentatywny charakter problemów zebranych w ramach testów tego typu, można traktować poszczególne funkcje za reprezentantów pewnych klas problemów. Pozwala to wysnuwać wnioski dotyczące zachowania się algorytmów optymalizacyjnych w kontekście problemów posiadających określoną charakterystykę. Taka możliwość bywa bardzo cenna w procesie modyfikacji i strojenia algorytmów.

3.2.2 Testy typu *black-box*

Określenie *black-box* odnosi się do charakteru realizacji problemów i dostępu do nich. Założeniem, towarzyszącym testom określanym jako *black-box* jest zgodność problemu z punktu widzenia użytkownika z definicją z rozdziału 1.1. Oznacza to, że użytkownik testów nie możliwości skorzystania przy implementowania algorytmów z innych informacji dt. problemu niż:

definicji funkcji ograniczeń $g_1, g_2, \ldots g_m$

wartości funkcji f **w zadanym przez użytkownika zbiorze** X_B – gdzie X_B stanowi skończony zbiór zadanych przez algorytm punktów w przestrzeni X_D.

Tego rodzaju konstrukcja testu pozwala na zakwalifikowanie rozwiązanego przez użytkownika problemu do zdefiniowanego w sekcji 1.1 problemu.

W praktyce, ilość informacji udostępnianych przez twórców testów sprawności typu *black-box* prawie zawsze jest większa niż opisana wyżej. Ideą tych testów i warunkiem koniecznym dla uzyskania pewności oferowanych przez nie wyników jest jednak takie wykorzystanie dodatkowych danych, by nie wpłynęły one na przebieg algorytmu. Dodatkowe dane mają stanowić cenną wskazówkę przy analizie konkretnych rozwiązań lub uproszczenie przy implementacji (np. raportowania).

Klasycznym przykładem tego rodzaju dodatkowej informacji jest podanie przez twórców testu wartości f_{min} czyli wartości funkcji f w punkcie zgodnym z najlepszym możliwym rozwiązaniem. Wartość ta pomaga skonstruować wykresy oparte o błąd zdefiniowany jako $e(x) = f_{min} - f(x)$. Przewagą tej funkcji nad funkcją $f(x)$ jest przeciwdziedzina $D^{-1} \in [0, \infty]$ która pozwala np. na skonstruowanie wykresu w skali logarytmicznej.

3.2.3 Warunki dodatkowe testów

Wiele testów sprawności oprócz podstawowych elementów dostarcza pewnych wskazówek sugerujących sposób testowania algorytmów przy ich użyciu. Wskazówki te nie mają wiążącego charakteru i zwykle nie zastosowanie się do nich nie powoduje spadku jakości badań, jednak ich stosowanie niesie ze sobą następujące korzyści:

- Użytkownik zestawu testów zwolniony jest z obowiązku dostrajania, definiowania warunków i analizowania ich wpływu na otrzymywane wyniki.
- Użytkownik zestawu testów ma możliwość skonfrontowania swoich wyników z wynikami innych badań wykorzystujących dany zestaw testów – można oczekiwać, że większość opublikowanych wyników eksperymentów prowadzonych przy użyciu danego testu sprawności będzie przeprowadzona z zastosowaniem warunków dodatkowych.
- Użytkownik zwolniony jest z konieczności obsługi części sytuacji trudnych lub atypowych.

Popularnymi warunkami dodaktowymi są **budżet** i **ilość wykonań**.

budżet – definiuje X_B^{max} jako liczbę naturalną taką, że $|X_B| \leqslant X_B^{max}$. Wyznaczenie przez zestaw testowy budżetu zwalnia użytkownika z konieczności definiowania deterministycznego warunku stopu algorytmu – stanowi w tym kontekście warunek awaryjny.

Ponadto zastosowanie budżetu pozwala na nakierowanie badań przy użyciu danego zestawu testów na odwzorowanie realiów konkretnych problemów. Przykładem mogą być testy przy bardzo ograniczonym budżecie, które testują jakość algorytmów przy założeniu braku możliwości dokładnego zbadania przeszukiwanej przestrzeni.

ilość wykonań – dotyczy algorytmów niedeterministycznych i określa jaka ilość prób jaką należy wykonać na danym problemie by statystyki wyznaczane na otrzymanych wynikach były reprezentatywne.

3.2.4 Wady

Podstawowym problemem związanym z testami sprawności typu *black-box* jest niska weryfikowalność wyników otrzymywanych w ramach poszczególnych badań. W szczególności możliwe są następujące uchybienia przy prowadzeniu badań przy użyciu większości testów tego rodzaju:

- Wielokrotne powtarzanie eksperymentów przy użyciu niedeterministycznych algorytmów w celu uzyskania możliwie najlepszego wyniku dla poszczególnych zdań.
- Wykorzystanie dodatkowych informacji udostępnionych przez twórców zestawu w konstrukcji algorytmu.
- Przekraczanie budżetu.

Cześć z tych problemów zostało rozwiązanych np. w testach *BBComp* [28]. W ramach konkursu przed konferencją *CEC 2015*, mającego stanowić bazę dla oceny poszczególnych rozwiązań, użytkownicy mają dostęp jedynie do interfejsów umożliwiających wykonywanie operacji zgodnych z założeniami testu *black-box*. Dopuszczalne jest więc tylko zapytanie o wartość funkcji w punkcie. Ponadto do skorzystania z interfejsu wymagane jest indywidualne konto, do którego przypisany jest budżet. Przekroczenie budżetu nie powinno być więc możliwe.

Niestety rozwiązanie zastosowane w *BBComp* ma pewne wady. Po pierwsze wartości uzyskiwane w pojedynczych wykonaniach dla konkretnych problemów są nieodporne na przypadkowe niepowodzenia (lub sukcesy) wynikające ze stochastycznego charakteru badanych algorytmów. Po drugie, nadal udostępniana jest informacja pomocnicza w postaci wartości optymalnej dla każdej z dostępnych w benchmarku funkcji. Ponadto z punktu widzenia bezpieczeństwa systemu, problemem jest metoda weryfikacji nowych użytkowników (brak weryfikacji) a także sama technologiczna konstrukcja testu pozostawiająca możliwości przekraczania budżetu jak i analizy poszczególnych funkcji.

Poza problemami związanymi z weryfikowalnością, występują jeszcze inne problemy charakterystyczne dla testów sprawności typu *black-box*:

precyzja cyfrowa – problem wynikający z binarnego charakteru reprezentacji i skończoności pamięci maszyn cyfrowych obsługujących testy. Zbiór liczb rzeczywistych w reprezentacji cyfrowej jest skończony a zgodnie z modelem matematycznym powinien

mieć liczność $\mathfrak{c}$. Konsekwencją tego faktu jak również typowo stosowanego standardu reprezentacji liczb zmiennoprzecinkowych IEEE-754 [29], jest pewien spodziewany przedział wartości ϵ na przeszukiwanej przestrzeni X_D, dla którego zmienność badanej funkcji jest pomijalna. Tego rodzaju wnioskowanie nie jest możliwe w oparciu o matematyczny model 2.1.1 a zarazem jest powszechnie wykorzystywane np. w algorytmie *ADE* opisanym w rozdziale 2.2.8.

znany charakter problemów – problemy tworzone w ramach testów sprawności, są często sztucznymi funkcjami skonstruowanymi wg. najprostszych dla danych właściwości matematycznych formuł. Pewne elementarne własności charakterystyczne dla tych problemów (dla których nie ma naturalnego wytłumaczenia w modelu matematycznym) mogą być wykorzystane przez algorytmy w celu poprawy jakości ich działania. Walka z tym zjawiskiem powinna być możliwa, ale znacząco utrudnia konstruowanie testów. Pewne próby w tym kierunku poczyniono w ramach testów *CEC 2013 benchmark* opisanych w rozdziale 3.2.5. Jak pokazują wyniki tych testów, zjawisko to ma wpływ na jakość wyników – problem ten jest więc rzeczywiście obecny w badaniach.

przeuczenie – Proces tworzenia i modyfikowania algorytmów oparty jest o badanie jak zmiany w algorytmie wpływają na wyniki testów sprawności uzyskiwane przez algorytm. W związku z tym, przy próbie oceny jakości algorytmu przy użyciu tego samego lub podobnego zestawu testów, może występować znany z uczenia maszynowego problem przeuczenia. Choć podejście twórców algorytmów, oparte jest o założenie o reprezentatywności zestawu testów w kontekście wszystkich problemów, to praktyczna nieprawdziwość tego założenia odbija się w błędzie wyników testów sprawności, rozumianych jako estymator jakości algorytmu. Ponieważ w czasie tworzenia algorytmu staramy się maksymalizować jakość algorytmu oceniając ją przez pryzmat obarczonego błędem estymatora, po pierwsze pomijamy błąd estymatora w badaniach, ale co gorsza generujemy algorytm, który najprawdopodobniej będzie w praktyce jakościowo słabszy.

Niestety uniknięcie tego problemu jest problematyczne, głównie ze względu na fakt, że większość testów sprawności zbudowana jest w oparciu o podobne funkcje, a zatem wykorzystanie innego zestawu testów do tworzenia i badania algorytmu, prawdopodobnie tylko w niewielkim stopniu zredukuje zaistniały problem. Tezę tę należało by jednak poddać weryfikacji w toku dalszych badań.

3.2.5 *CEC 2013 benchmark*

Przykładem testu sprawności typu *black-box* jest zestaw testów opracowany w ramach konferencji *CEC 2013 - IEEE Congress on Evolutionary Computation – CEC2013 benchmark*. Pełen opis testu, funkcji, zakresów przeszukiwania jak i warunków dodatkowych został zawarty w [11].

Zestaw testów *CEC 2013 benchmark* składa się z 28 funkcji podzielonych na 3 grupy:

funkcje jednomodalne – funkcje zawierające w przeszukiwanej przestrzeni X_D dokładnie jedno minimum lokalne.

funkcje wielomodalne – funkcje zawierające w przeszukiwanej przestrzeni X_D wiele minimów lokalnych.

funkcje złożone – funkcje utworzone przez złożenie funkcji z dwóch poprzednich grup. Mają one najbardziej złożony charakter i jedno albo wiele minimum lokalne w przestrzeni X_D.

Wszystkie funkcje mają publiczną definicję matematyczną, jednak analityczna analiza ich formuły nie jest zgodna z opisem problemu ani z założeniami samego zestawu testów w związku z tym ta ewentualność zostanie zaniedbana.

Wszystkie funkcje dostarczone w ramach testu są dostępne dla wymiarowości przestrzeni: 2, 10, 30, 50. Wymiarowości 10, 30 i 50 są zalecane jako zestaw testowy. Przestrzeń dwuwymiarowa przeznaczona jest do celów testowych i wizualizacyjnych.

Dla wszystkich funkcji zdefiniowano wartość minimum globalnego. Ponadto zdefiniowano, że wartości błędu poniżej 10^{-8} nie powinny być rozróżniane i powinny być traktowane jako optimum globalne $\mathbf{x_{best}}$ takie, że $e(\mathbf{x_{best}}) = 0$.

Cechą charakterystyczną tego zestawu testów jest zastosowanie pewnych funkcji w wariantach standardowych jak i obróconych. Pozwala to na wykrycie na etapie testów preferencji algorytmu dla funkcji, które mają wyróżnioną charakterystykę wzdłuż osi. Wyniki eksperymentów wielu algorytmów sugerują znaczący wpływ takiej preferencji [9, 10], która zdaje się nie mieć silnego poparcia w realnych problemach.

W ramach opisu testu *CEC 2013 benchmark* dostarczono szeregu wskazań opisujących sposób zebrania danych. Wyszczególniono:

- budżet – $X_B^{max} = D \cdot 10'000$
- ilość wykonań – $R = 51$
- Punkty charakterystyczne na przestrzeni budżetu dla których należy zbierać informacje o najlepszym dotąd odnalezionym punkcie. Pozwala to na sprawdzenie jak zachowuje się dany algorytm dla zadanego budżetu, ale także dla zdefiniowanych mniejszych budżetów – dla wszystkich badań podążających za wytycznymi.

Ewolucja różnicowa – pozycja w *CEC 2013 benchmark*

Wartym odnotowania jest fakt, że w ramach badań w oparciu o testy *CEC 2013 benchmark* opisano wiele algorytmów z klasy algorytmów ewolucji różnicowej. Najlepsze rezultaty opisane w końcowym raporcie z konferencji [8] uzyskały algorytmy z rodziny *CMAES*, jednak czołowe pozycje uzyskały także algorytmy ewolucji różnicowej. I tak w pierwszej dziesiątce znalazły się:

- *SHADE* [10] – pozycja 4.
- *SMADE* [30] – pozycja 7.
- *TLBSaDE* [31] – pozycja 8.
- *DEcfbLS* [32] – pozycja 9.

Ponieważ algorytm *SHADE* uzyskał najlepszy wynik pośród algorytmów ewolucji różnicowej, został on wybrany jako baza do zaawansowanego algorytmu *ANADE*.

Motywacja wyboru zestawu testów

Zróżnicowane funkcje testowe, precyzyjne wytyczne co do zbierania danych, liczne badania algorytmów ewolucji różnicowej oraz dostępność implementacji w języku wykorzystanym do implementacji sprawiły, że właśnie test *CEC 2013 benchmark* został wybrany jako właściwy do pokazania właściwości badanych w ramach pracy algorytmów.

Rozdział 4

Algorytmy *NADE* i *ANADE*

4.1 Cele i założenia

Główną ideą stojącą za projektowaniem algorytmu *NADE* (ang. *Normalized Archive Differential Evolution*), było skonstruowanie algorytmu wykorzystującego archiwum punktów odbiegającego w możliwie niewielkim stopniu od typowych wariantów ewolucji różnicowej. Skupiono się tu przede wszystkim na próbie dołączenia archiwum do klasycznego algorytmu *DE/rand/1/bin*.

Tego rodzaju podejście pozwala zbadać użyteczność archiwum w oderwaniu od innych metod poprawy jakości działania algorytmu. Istnieje możliwość, że uzyskane rezultaty będą przynosiły korzyści zanikające w przypadku bardziej wysublimowanych wariantów algorytmu. W związku z tym, w rozdziale 4.4 zaproponowano algorytm *ANADE* (ang. *Adaptive Normalized Archive Differential Evolution*), bazujący na algorytmie *SHADE*. W rozdziale 5, każdy z zaprojektowanych algorytmów zostanie porównany ze swoim niewykorzystującym archiwum odpowiednikiem.

W rozdziale tym, w przypadku odwołań do wcześniej istniejących wariantów ewolucji różnicowej, stosowane będą oznaczenia z rozdziału 2.2.

4.2 Uproszczony schemat badań

W czasie projektowania algorytmu sprawdzenia wymagały pewne tezy dotyczące możliwych dróg rozwoju algorytmu. Ponieważ przeprowadzenie kompletnego schematu badań zestawu testów sprawności *CEC 2013 benchmark* jest czasochłonne, zdecydowano się do celów wstępnych badań zastosować procedurę ustaloną w oparciu o następujące założenia:

- Wykorzystanie wszystkich funkcji testowych f_1 do f_{28}.
- Wykonywanie badań tylko na jednej wymiarowości przestrzeni $D = 30$.
- Wykonywanie 25 powtórzeń dla każdej z funkcji testowych (zamiast 51).
- Bazowe wartości parametrów DE wg [33],
  - $F = 0.5$
  - $Cr = 0.9$

- $\mu = Max(50, 2 \cdot D)$

- Wykorzystanie testu Wilcoxona na zebranych danych w celu weryfikacji określonych tez. Przyjęto standardowy poziom istotności 0.05.

W dalszym ciągu tego rozdziału będą prezentowane całościowe lub częściowe wyniki badań uproszczonych w celu uzasadnienia podejmowanych w czasie projektowania algorytmu decyzji.

Istnieje ryzyko przeuczenia algorytmów wynikające z projektowania algorytmu w oparciu o zestaw testów go oceniających. Testy *CEC 2013 benchmark* są jednak traktowane jako reprezentatywne dla wszystkich możliwych problemów. Zarazem decyzje w czasie projektowania algorytmów będą podejmowane wyłącznie w oparciu o jasne przesłanki (tzn. wyraźna przewaga konkretnego wariantu). W związku z tym, wydaje się, że problem przeuczenia nie powinien mieć wpływu na wynik badań lub też zgodnie z tezą postawioną w rozdziale 3.2.4 uniknięcie tego problemu nie było by możliwe przez użycie innego zestawu testów do badań niż do projektowania.

4.3 Algorytm *NADE*

4.3.1 Podejścia do wykorzystania archiwum

Podstawowym celem wykorzystania archiwum w procesie ewolucji różnicowej, było zwielokrotnienie liczby możliwych do wygenerowania wektorów różnicowych bez konieczności powiększania populacji. Intuicyjnym sposobem na zbudowanie tego rodzaju algorytmu jest włączenie w skład populacji poddawanej mutacji różnicowej osobników pochodzących z poprzednich iteracji, które zostały już wyeliminowane z bieżącego pokolenia. W ten sposób uzyskiwana jest możliwość sterowania wielkością takiego archiwum za pomocą nowego parametru H oznaczającego wielkość populacji (archiwum) używanego do generowania wektorów różnicowych w mutacji różnicowej.

Tego rodzaju podejście stanowiło punkt wyjścia dla algorytmu *NADE*. Dla poszczególnych badanych wariantów zastosowano $H \in \{1.5 \cdot \mu, 2 \cdot \mu, 4 \cdot \mu\}$. Takie wartości pozwalają na badanie algorytmu w porównaniu do klasycznego algorytmu ewolucji różnicowej przy standardowych nastawach podanych w rozdziale 4.2, jednocześnie są to wartości bliskie stosowanym do tej pory w innych algorytmach ewolucji różnicowej wykorzystujących archiwum (rozdział 2.2.8).

4.3.2 Wybór punktów do archiwum

Kluczowym przy projektowaniu mechanizmu archiwum jest wybranie mechanizmu doboru elementów wchodzących w jego skład. W przypadku algorytmu *NADE* zastosowano jako punkt wyjścia najprostszy (opisany w rozdziale 2.1.2) mechanizm przesuwnego okna archiwum. W przypadku tego algorytmu, do archiwum $\mathbf{A}$ dodawane są punkty, które w czasie selekcji zostały dodane do populacji $\mathbf{X}$. Archiwum inicjalizowane jest pierwszym pokoleniem X. W przypadku przepełnienia archiwum ($|\mathbf{A}| + \mu > H$) usuwane są z niego osobniki dodane w najwcześniejszym pokoleniu. W przypadku wielu osobników w archiwum pochodzących z tego samego pokolenia, kolejność usuwania elementów jest losowa.

Tak zdefiniowany mechanizm doboru elementów do archiwum został dobrany do algorytmu *NADE* jako podstawowy punkt wyjścia. Jednocześnie w toku dalszych prac nie zostały dostrzeżone przesłanki wskazujące, że zmiana tego mechanizmu na inny cechujący się podobnym stopniem złożoności mogła by przyczynić się do statystycznie istotnej poprawy uzyskiwanych przez algorytm rezultatów.

4.3.3 Normalizacja archiwum

Pierwsze próba porównania algorytmu *NADE* z *DE/rand/1/bin* przy użyciu uproszczonego schematu badań pozwoliła uzyskać wyniki zaprezentowane w tabeli 8.1 w rozdziale 8 (podsumowanie w tabeli 4.1 poniżej). Wyniki te sugerują, iż przyjęta metoda może mieć potencjał dla wielu funkcji, ale zarazem, że uzyskiwana korzyść zanika w miarę jak rośnie wartość H.

Tablica 4.1: Podsumowanie wyników uproszczonego porównania algorytmu *NADE* bez normalizacji z *DE/rand/1/bin* dla różnych wartości H przy użyciu testu Wilcoxona. W tablicy ilość funkcji, dla których wyniki uległy statystycznej istotnej poprawie (+), pogorszeniu (-) i nie uległy statystycznej zmianie (=)

H	+	-	=
90	8	4	16
120	7	13	8
240	1	19	8

Warto odnotować, że w przypadku ewolucji różnicowej, oczekiwana jest coraz mniejsza wariancja osobników w kolejnych pokoleniach. Przetestowana wersja algorytmu mogła przyczynić się do zaburzenia tej cechy, ponieważ wariancja osobników w każdym pokoleniu zależy od wielu poprzednich – innymi słowy – wraz ze wzrostem wartości H rośnie wariancja bieżącej populacji.

W związku z tym, podjęto próbę modyfikacji algorytmu w celu wyeliminowania tej jego właściwości. W ten sposób powstał mechanizm adaptujący archiwum do bieżącego pokolenia. Mechanizm ten składa się z dwóch elementów:

Normalizacji – archiwum znormalizowane $\mathbf{A_{normalized}} = Normalize(\mathbf{A})$, gdzie funkcja $Normalize(\mathbf{A})$ przesuwa elementy archiwum $\mathbf{A}$ o wektor między punktem środkowym a punktem $\mathbf{0^D}$, a następnie dzieli wszystkie współrzędne wszystkich elementów archiwum przez średnią długość wektora miedzy elementami w $\mathbf{A}$, zgodnie z procedurą:

Algorithm 4 Schemat normalizacji w algorytmie *NADE*

1: **function** NORMALIZE(A)
2: $distance_{mean} \leftarrow$ MEANDISTANCE(A)
3: **for all** i $\in$ 1:$|A|$ **do**
4: $A[i] \leftarrow \frac{A[i]-M_A}{distance_{mean}}$
5: **end for**
6: **return** A
7: **end function**

gdzie $\mathbf{M_A}$ zdefiniowano w taki sposób, że:

$$M_{A,j} = \frac{1}{|A|} \sum_{a \in A} a_j \tag{4.1}$$

natomiast $MeanDistance(A)$ to średnia odległość euklidesowa:

$$MeanDistance(A) = \frac{2}{(|A|^2 - |A|)} \sum_{i=1}^{|A|} \sum_{j=(i+1)}^{|A|} \sqrt{\sum_{k=1}^{D} (A_{i,k} - A_{j,k})^2} \tag{4.2}$$

Denormalizacji – archiwum zdenormalizowane $\mathbf{A} = Denormalize(\mathbf{A_{normalized}}, \mathbf{X})$, gdzie funkcja $Denormalize(\mathbf{A})$ stanowi odwrotność funkcji $Normalize$ w kontekście bieżącego pokolenia $\mathbf{X}$. Zachodzi więc:

$$\mathbf{A} = Denormalize(Normalize(\mathbf{A}), \mathbf{A}) \tag{4.3}$$

a procedura opisująca funkcję $Deormalize(\mathbf{A}, \mathbf{X})$ wygląda następująco:

Algorithm 5 Schemat denormalizacji w algorytmie *NADE*

```
1: function DENORMALIZE(A, X)
2:     distance_mean ← MEANDISTANCE(X)
3:     for all i ∈ 1:|A| do
4:         A[i] ← distance_mean · A[i] + M_X
5:     end for
6:     return A
7: end function
```

Ten sposób normalizacji i denormalizacji, wymusza w każdym pokoleniu istnienie archiwum $\mathbf{A}$, ułożonego wokół punktu środkowego populacji ze średnią długością wektora taką, jak w populacji $\mathbf{X}$. Archiwum $\mathbf{A}$ wykorzystywane może być więc do mutacji różnicowej, natomiast do jego wyznaczenia w każdym pokoleniu wykorzystywane będzie archiwum $\mathbf{A_{normalized}}$.

Przy tak zdefiniowanym algorytmie *NADE* uzyskano rezultaty jak w tabeli 8.2 w rozdziale 8. W tabeli 4.2 zaprezentowanie podsumowanie testu Wilcoxona dla porównania *DE/rand/1/bin* z poszczególnymi nastawami *NADE*.

Tablica 4.2: Podsumowanie wyników porównania algorytmu *NADE* z *DE/rand/1/bin* dla różnych wartości H przy użyciu testu Wilcoxona. W tablicy ilość funkcji, dla których wyniki uległy statystycznej istotnej poprawie (+), pogorszeniu (-) i nie uległy statystycznej zmianie (=)

H	+	-	=
90	8	8	12
120	8	12	8
240	7	13	8

4.3.4 Wielkość populacji

Zebrane dane w dalszym ciągu nie sugerowały możliwości statystycznie istotnej poprawy wyników uzyskiwanych przez algorytm *DE/rand/1/bin* a zarazem uzyskane wyniki wskazują na poprawę względem algorytmu *NADE* bez normalizacji. W tej sytuacji zdecydowano się przetestować dodatkowo hipotezę, o możliwości poprawy jakości uzyskiwanych wyników przez zmniejszenie populacji, a zatem zwiększenie ilości iteracji algorytmu w ramach tego samego budżetu. Dodatkowe archiwum, mogło by w tej sytuacji zapewniać równoważną ilość wektorów różnicowych przy mniejszej populacji.

W związku z powyższym, w ramach wstępnego badania, zestawiono algorym *DE/rand/1/bin* dla $\mu = 40$ z algorytmem NADE dla $\mu = 40$ i $H = 60$. Wyniki tego eksperymentu przedstawiono w tabeli 8.3 w rozdziale 8 (skrócona wersja w tabeli 4.3 poniżej). Dodatkowo, by stwierdzić, czy wydłużenie czasu obliczeń może wpłynąć pozytywnie na ogół wyników uzyskiwanych w problemach z *CEC 2013 benchmark*, zestawiono algorytm NADE dla powyższych ustawień, z algorytmem $DE/rand/1/bin$ dla ustawień podstawowych ($\mu = 60$) wyniki tych eksperymentów zawarto w tablicach 8.4 oraz 4.4.

Tablica 4.3: Podsumowanie wyników porównania algorytmu *NADE* ($H = 60$ i $\mu = 40$) z *DE/rand/1/bin* ($\mu = 40$) przy użyciu testu Wilcoxona. W tablicy ilość funkcji, dla których wyniki uległy statystycznej istotnej poprawie (+), pogorszeniu (-) i nie uległy statystycznej zmianie (=)

H	+	-	=
60	10	6	12

Tablica 4.4: Podsumowanie wyników porównania algorytmu *NADE* ($H = 60$ i $\mu = 40$) z *DE/rand/1/bin* ($\mu = 60$) przy użyciu testu Wilcoxona. W tablicy ilość funkcji, dla których wyniki uległy statystycznej istotnej poprawie (+), pogorszeniu (-) i nie uległy statystycznej zmianie (=)

H	+	-	=
60	4	8	16

Wyniki wskazują, że dla małych populacji dodanie znormalizowanego archiwum pozwalającego na wygenerowanie większej liczby różnych wektorów różnicowych pozwala uzyskać lepsze rezultaty. Z drugiej strony, nadal istnieje podejrzenie, że dla właściwych wartości parametrów *DE/rand/1/bin* tylko nieliczne funkcje mogą być zbadane dokładniej za sprawą zyskanej kosztem liczebności populacji ilości iteracji algorytmu.

4.3.5 Wstępna hipoteza

Zebrane wyniki sugerują, iż istnienie archiwum zarówno w znormalizowanej jak i zdenormalizowanej nie wpływają korzystnie w statystycznie istotny sposób na wyniki uzyskiwane w większości funkcji z *CEC2013 benchmark*. Istnieje także podejrzenie, że w rzeczywistości istnienie archiwum w takiej formie może powodować pogorszenie uzyskiwanych wyników. Ponieważ jednak wyniki, zwłaszcza w przypadku pomniejszonej populacji

są niejednoznaczne, algorytm NADE będzie poddany kompletnym badaniom zgodnym z wytycznymi z [11] w rozdziale 5.

4.4 Algorytm *ANADE*

Podstawowy wariant algorytmów ewolucji różnicowej, *DE/rand/1/bin* został wielokrotnie pokonany pod kątem możliwości przez inne bardziej wysublimowane warianty tych algorytmów. Z tego powodu, wykazanie możliwości lub braku możliwości poprawy jakości uzyskiwanych przez *DE/rand/1/bin* wyników przez wprowadzenie znormalizowanego archiwum, nie świadczy jednoznacznie o użyteczności tego rozwiązania w kontekście poprawy wyników uzyskiwanych przez algorytmy ze wspomnianej grupy.

Argumentacja ta, doprowadziła do próby skonstruowania algorytmu *ANADE* opartego ściśle o algorytm *SHADE*. Algorytm SHADE został wybrany jako reprezentant zaawansowanych algorytmów ewolucji różnicowej, ze względu na najlepsze w grupie algorytmów ewolucji różnicowej wyniki, jakie uzyskał w teście *CEC2013 benchmark* [8].

Algorytm *ANADE* należy więc rozumieć jako taką modyfikację algorytmu *SHADE*, która normalizuje i denormalizuje archiwum (istniejące już w SHADE) wg. schematów znanych z *NADE*. Tę modyfikację poddano weryfikacji przez zestawienie uzyskiwanych przez nią wyników z wynikami uzyskiwanymi przez SHADE. Dla obu algorytmów zastosowano nastawy zgodne z [10]. Jako bazę implementacji wykorzystano kod źródłowy udostępniony przez autorów *SHADE* w ramach konferencji *CEC2013*. W takim zestawieniu uzyskano wyniki jak w tablicy 4.5 poniżej (pełne wyniki w załączniku, tablica 8.3).

Tablica 4.5: Podsumowanie wyników porównania algorytmu *ANADE* z *SHADE* dla proponowanych w [10] wartości parametrów, przy użyciu testu Wilcoxona. W tablicy ilość funkcji, dla których wyniki uległy statystycznej istotnej poprawie (+), pogorszeniu (-) i nie uległy statystycznej zmianie (=)

+	-	=
1	4	23

Uzyskane wyniki wskazują na trzy hipotezy do przetestowania w pełnym schemacie badań:

- Wpływ znormalizowanego archiwum na wyniki uzyskiwane przez algorytm zanika w zaawansowanym wariancie ewolucji różnicowej – *SHADE*.
- Średni wpływ znormalizowanego archiwum na działanie algorytmów ewolucji różnicowej jest negatywny.
- Istnieją problemy, dla których znormalizowane archiwum poprawia uzyskiwane przez *SHADE* wyniki.

Rozdział 5

Badania

5.1 Cel

W poprzednim rozdziale przy użyciu uproszczonego schematu badań wysunięto hipotezy dotyczące wpływu znormalizowanego wg. wskazanej procedury archiwum na wyniki uzyskiwane w różnych wariantach ewolucji różnicowej. W celu zweryfikowania tych tez, przeprowadzono badania w oparciu o schemat zgodny z zalecaniami twórców *CEC2013 benchmark*.

Weryfikacji poddane zostaną następujące tezy:

1. Archiwum znormalizowane wg. procedury algorytmu *NADE* powoduje poprawę jakości uzyskiwanych przez ewolucję różnicową dla niektórych typów funkcji.
2. Tego rodzaju archiwum nie jest perspektywiczne jeśli chodzi o jakość algorytmu dla nieznanego charakteru problemu.
3. Wpływ takiej modyfikacji na jakość wyników zanika w przypadku zastosowania go w zaawansowanym algorytmie ewolucji różnicowej (*SHADE*).
4. Archiwum niewielkich rozmiarów (ok. $1.5 \cdot \mu$) daje największe korzyści przy jednoczesnym zminimalizowaniu liczby problemów, dla których przynosi ono skutek negatywny.

Szczególnie istotne jest sprawdzenie postawionych tez w kontekście innych niż pierwotnie badano wymiarowości problemu.

5.2 Plan eksperymentów

- Wykorzystanie wszystkich funkcji testowych f_1 do f_{28}.
- Wykonywanie badań dla wymiarowości $D \in \{10, 30, 50\}$.
- Wykonywanie 51 powtórzeń dla każdej z funkcji testowych.
- Bazowe wartości parametrów DE wg [33] oraz [10],
  - $F = 0.5$

  - $Cr = 0.9$
  - $\mu = Max(50, 2 \cdot D)$ – w przypadku *DE/rand/1/bin* i *NADE*
  - $\mu = 100$ – w przypadku *SHADE* i *ANADE*

- Wykorzystanie testu Wilcoxona na zebranych danych w celu weryfikacji określonych tez. Przyjęto standardowy poziom istotności 0.05.

W celu weryfikacji tez, wg. powyższego scenariusza zostaną wykonane następujące porównania:

- Porównanie algorytmu *DE/rand/1/bin* i *NADE* dla $H \in \{1.5 \cdot \mu, 2 \cdot \mu, 4 \cdot \mu\}$
- Porównanie algorytmu *SHADE* i *NADE* dla standardowych parametrów *SHADE*.

Ze względu na koszta czasowe i niskie rokowania, nie będzie przeprowadzone porównanie algorytmu *NADE* z *DE/rand/1/bin* przy pomniejszonej w stosunku do przyjętej zgodnie z [33] liczebności populacji.

5.3 Sposób analizy

Dla każdego zestawu parametrów, dla każdego algorytmu i każdej funkcji testowej, algorytm zostanie wykonany 51 krotnie. Uzyskane wartości optymalizowanej funkcji w najlepszych punktach w kolejnych próbach będą przedmiotem analizy.

W przypadku porównywania algorytmów, serie danych będą porównywane dla każdej funkcji i każdego zestawu parametrów osobno przy użyciu testu Wilcoxona. Dodatkowo wyznaczona więc będzie średnia wartość błędu z serii informująca o rzeczywistym rzędzie wielkości wyników uzyskiwanych przez algorytmy.

Tak zebrane dane pozwolą na przeanalizowanie otrzymywanych przez różne algorytmy wyników w kontekście konkretnych problemów, a także niezależne porównanie wyników z innymi opublikowanymi pracami wykorzystującymi ten sam zestaw testów.

5.4 Skrót wyników

Przeprowadzone eksperymenty pozwoliły na uzyskanie wyników przedstawionych w tablicach 8.6 do 8.11 w załączniku pracy, poniżej zaprezentowano ogólny poziom wyników w postaci ilości statystycznie istotnych różnic między porównywanymi algorytmami przy konkretnych parametrach (tablica 5.1).

5.5 Wnioski

Analizując wyniki eksperymentów, należy zwrócić uwagę, na fakt, że w uzyskanych wynikach wyraźnie widać, że dla pewnych funkcji dodanie archiwum (szczególnie w przypadku algorytmu NADE) niezależnie od przyjętych parametrów przynosi korzyść. Wstępne badania sugerowały, że może mieć to związek z charakterem funkcji i dotyczyć w szczególności wielu funkcji jedno modalnych (f_1 do f_5) oraz złożonych (f_{21} do f_{28}). W tych

Tablica 5.1: Podsumowanie wyników kompletnych badań algorytmów *NADE* i *ANADE* w oparciu o zestaw testów *CEC2013 benchmark*. W tablicy ilość funkcji, dla których wyniki uległy statystycznej istotnej poprawie (+), pogorszeniu (-) i nie uległy statystycznej zmianie (=), względem odpowiedników algorytmów nie zawierającego znormalizowanego archiwum, dla różnych wymiarowości D.

D	**NADE** $H = 1.5 \cdot \mu$			**NADE** $2 \cdot \mu$			**NADE** $H = 4 \cdot \mu$			**ANADE**		
	+	-	=	+	-	=	+	-	=	+	-	=
10	4	11	13	4	12	12	5	13	10	3	2	23
30	8	8	12	8	11	9	7	13	8	1	4	23
50	6	13	9	3	14	11	0	17	11	2	4	22

grupach podsumowanie testu Wilcoxona prezentowało się jak w tablicy 5.2 poniżej. Sugeruje to, że algorytm poszerzony o niewielkie archiwum w najlepszym razie nie traci na możliwościach przeszukiwania w przypadku funkcji tego rodzaju (i to tylko dla $H = 1.5 \cdot \mu$. Obala to hipotezę pierwszą, szczególnie dla większych wymiarowości problemów. Istnieją funkcje, dla których znormalizowane archiwum przynosi poprawę jednak jest to nieliczna i nieregularna podgrupa wszystkich analizowanych problemów.

Tablica 5.2: Podsumowanie wyników kompletnych badań algorytmów *NADE* i *ANADE* w oparciu o zestaw testów *CEC2013 benchmark* dla funkcji jedno modalnych i złożonych ($f_1 - f_5$ oraz $f_{21} - f_{28}$). W tablicy ilość funkcji, dla których wyniki uległy statystycznej istotnej poprawie (+), pogorszeniu (-) i nie uległy statystycznej zmianie (=), względem odpowiedników algorytmów nie zawierającego znormalizowanego archiwum, dla różnych wymiarowości D.

D	**NADE** $H = 1.5 \cdot \mu$			**NADE** $2 \cdot \mu$			**NADE** $H = 4 \cdot \mu$			**ANADE**		
	+	-	=	+	-	=	+	-	=	+	-	=
10	2	2	9	2	3	8	2	2	9	1	1	11
30	5	3	5	5	2	6	4	4	5	1	1	11
50	4	4	5	2	5	6	0	7	8	0	1	12

Jednocześnie, ważnym wnioskiem jest jednak fakt, że przyjmując próbę problemów z testu *CEC2013* za reprezentatywną dla wszystkich możliwych problemów, dodanie archiwum powoduje średnio (dla całkowicie nieznanego problemu) pogorszenie wyników uzyskiwanych przez algorytm. Potwierdza to hipotezę drugą, dla wszystkich wymiarowości problemu i analizowanych wariantów ewolucji różnicowej.

Trzecia z hipotez, również wydaje się być potwierdzona. Dla innych wymiarowości i bardziej reprezantatywnej liczby powtórzeń wykonań algorytmu, wyniki nadal wskazują na zanikający wpływ normalizacji archiwum w przypadku zastosowania jej do algorytmu *SHADE*. W takim przypadku zanikają również wzorce o których mówi hipoteza pierwsza, co dodatkowo wpływa na jej obalenie.

Ostatnią z hipotez najłatwiej potwierdzić sumując wartości z kolumny + i − dla wszystkich wymiarowości i kolejnych wielkości H. Tego typu operacja ujawnia rezultat widoczny w tabeli 5.3 poniżej. Wynik ten ze względu na zdecydowanie najlepszy bilans dla $H = 1.5 \cdot \mu$ co potwierdza ostatnią z hipotez. Warto jednak zaznaczyć, że ponieważ

bilans we wszystkich przypadkach był ujemny, ten wniosek można też przypisać faktowi, że najmniejsze archiwum najmniej ingeruje w podstawowy wariant algorytmu.

Tablica 5.3: Podsumowanie ilościowe popraw i pogorszeń uzyskiwanych przez *NADE* w stosunku do *DE/rand/1/bin* w zależności od wartości H.

H	+	**-**	**bilans**
$1.5 \cdot \mu$	18	32	-14
$2 \cdot \mu$	15	37	-22
$4 \cdot \mu$	12	43	-31

Rozdział 6

Obserwacje dotyczące metodyki

Poza celem pracy, prowadzenie badań doprowadziło do licznych obserwacji dotyczących metodyki badań, które stanowią potencjalny obszar rozwoju w celu lepszego dostosowania się do dalszego prowadzenia badań. Ten rozdział zawiera opis obserwacji i wstępne propozycje rozwiązania niektórych z napotkanych problemów.

6.1 Specyfika algorytmów ewolucji różnicowej

6.1.1 Stabilność algorytmów

Podstawowym elementem, który należy rozważyć przy omawianiu metodyki pracy nad algorytmami ewolucji różnicowej, jest podatność wyników przez nie uzyskiwanych na zmiany w algorytmie. Praktyka pokazuje, że niewielkie modyfikacje algorytmów w sposób istotny statystycznie wpływają na otrzymywane wyniki. Analogicznie sytuacja prezentuje się w kwestii ustawień dostępnych w algorytmach parametrów.

Tak niski poziom stabilności badanych algorytmów powoduje dwie ważne konsekwencje:

podatność na błędy – popełnienie drobnego błędu w implementacji założonej koncepcji może znacząco wpłynąć na końcowe wyniki. W praktyce gdy błąd doprowadzi do pogorszenia wyników, może to spowodować przedwczesne zakończenie badań nad konkretną koncepcją. Gdy z kolei błąd doprowadzi do poprawy wyników, wysunięte wnioski a przede wszystkim uargumentowanie co spowodowało poprawę rezultatów będą błędne. Zwłaszcza ta druga sytuacja jest niebezpieczna, gdyż autorzy kolejnych prac w oparciu o wcześniejsze badania mogą brnąć w gałąź nauki opartą na błędnych przesłankach.

dużą ilość stopni swobody – ze względu na ogromny wpływ każdego elementu na ogólny wynik działania algorytmu, a także na zachodzące między elementami zależności, ilość wariantów algorytmów które należy poddać badaniu by potwierdzić pewne teorie nie pozostawiając pola na wątpliwości jest niezliczona. Z tego powodu, częstą praktyką jest przyjmowanie pewnych wartości parametrów czy koncepcji ze wcześniejszych prac i domniemanie, że w połączeniu z proponowanymi modyfikacjami będą one stanowiły reprezentatywną próbę. Analogicznie postąpiono w tej

pracy. Ponieważ badanie wszelkich kontr teorii dotyczących zmian w wynikach uzyskiwanych przez algorytm jest przy dzisiejszym stanie wiedzy niemożliwe. Próba wyeliminowania tego problemu zdaje się być bardzo ciekawym, choć niewątpliwie dalece złożonym problemem, godnym dalszych badań.

6.1.2 Czas obliczeń

Na brak stabilności algorytmów z grupy algorytmów ewolucyjnych, nakłada się dodatkowo fakt, że ich rzetelne przetestowanie wymaga wielokrotnego powtarzania i tak często długotrwałych obliczeń. Przykładowo przeprowadzenie kompletnego badania algorytmu *DE/rand/1/bin* w 30 wymiarach przy tylko jednym zestawie parametrów zgodnie z wytycznymi [33] trwało 2-4h na współczesnym komputerze klasy PC.

Tego rzędu czasy obliczeń zmuszają do uproszczeń w schematach badań przynajmniej na wstępnym etapie prac, a także dodatkowo potęgują problematyczność związaną ze stabilnością wyników. Każdy bowiem błąd jak i każda dodatkowa teza wymagająca sprawdzenia powodują nie tylko konieczność poświęcenia czasu na obliczenia, ale także przestój w pracy badawczej.

Wydaje się więc, że podejmowane w literaturze próby zmniejszania ilości parametrów i wariantów algorytmu, za których przetestowanie odpowiedzialny jest projektant nowego algorytmu, jest drogą słuszną. W przypadku braku przełomowego rozwiązania, umożliwiającego pozbycie się tego rodzaju ograniczeń, dalsze zgłębianie charakteru badanych algorytmów w celu ustalenia, jakie stopnie swobody nie będą wymagały dodatkowych testów przy nowych algorytmach, wydaje się więc kolejnym istotnym i już trwającym kierunkiem badań.

6.1.3 Zbieranie danych

Innym problemem związanym z metodyką badań nad algorytmami wymagającymi wielokrotnego powtarzania eksperymentów, jest konieczność zbierania danych do analizy cech algorytmów. W podstawowym wariancie dla każdego uruchomienia algorytmu zbierana jest tylko końcowa wartość funkcji przystosowania. Przy bardziej złożonych analizach może okazać się jednak konieczne zebranie licznych serii danych dla każdego uruchomienia, a to z kolei powoduje, że sam sposób gromadzenia tych danych staje się złożonym zagadnieniem z dziedziny baz danych. Ta kwestia także pozostawia wiele pola do rozwoju w przyszłości.

6.2 Implementacja

6.2.1 Język implementacji

Mimo teoretycznego charakteru pracy, za uzyskiwanymi wynikami stały konkretne implementacje algorytmów. Pierwotne implementacje utworzone były w języku R [34]. Język ten wydawał się odpowiedni, ze względu na wsparcie dla operacji macierzowych, szybkość działania, dostępność implementacji *CEC2013 benchmark* oraz dużą zwięzłość pozwalającą na szybką implementację zakładanych funkcjonalności.

Język ten, choć pozwala na sprawne prototypowanie, niestety posiada szereg wad w kontekście stawianego przed nim zadania spośród których najpoważniejsze to:

słaba typizacja – ze względu na słabą kontrolę typów i brak możliwości organizacji typów wektorowych i macierzowych w struktury logiczne bez utraty wydajności obliczeń, język nie stanowi bariery dla potencjalnych błędów ze strony programisty.

brak kontroli rozmiarów macierzy – macierze i wektory w R poddają się operacją arytmetycznym niezależnie od wymiarowości. W pewnych sytuacjach prowadzi to do ostrzeżeń interpretera, jednak w większości wypadków kończy się powodzeniem. Analogicznie jak w przypadku typizacji, problemem jest brak mechanizmów ochrony użytkownika języka przed popełnieniem błędów.

Cechy te, w kontekście postawionych tez dotyczących stabilności algorytmów ewolucji różnicowej sprawiają, że język ten wydaje się nie być odpowiedni do tworzenia końcowych implementacji, ze względu na zbyt duże ryzyko błędu i rozbieżności między planowaną ideą a rzeczywistą interpretacją.

Ponadto, przy wstępnych badaniach, wykorzystanie tego języka może sugerować wstępne wnioski niezgodne z rzeczywistością, co jest szczególne dotkliwe z powodu możliwości odrzucenia potencjalnie interesujących gałęzi rozwoju, na wstępnym etapie badań. Wydaje się, że zysk płynący z szybkości implementacji jest niewspółmiernie mały.

Z drugiej strony, język R wydaje się nadal znakomitym narzędziem analizy danych, pozwalającym przy wykorzystaniu wbudowanych funkcjonalności wykonywać złożone operacje statystyczne na danych, bez konieczności wnikania użytkownika w ich implementację. Dodatkowo język R udostępnia szereg mechanizmów wizualizacji, które dodatkowo zwiększają jego atrakcyjność na etapie analizy danych. Stosowność jego zastosowania, jest więc tu podważana tylko w kontekście obliczeń symulacyjnych – takich jak testy algorytmów optymalizacyjnych.

Tego rodzaju rozumowanie prowadzi do poszukiwania odpowiedzi na pytanie – jaki język implementacji byłby najbardziej właściwy. W późniejszej części badań w ramach pracy wykorzystano do implementacji silnie typowany język obiektowy – *C++*. Spowodowało to znaczne zwiększenie tempa prac, mimo spowolnienia tempa implementacji. Mniejsza ilość błędów i większy wachlarz możliwości jeśli chodzi o ich odnajdywanie, spowodowały, że ilość kosztownych uruchomień symulacji znacząco zmalała przy jednoczesnym niewątpliwym wzroście wiarygodności badań. Dodatkową korzyścią płynącą z wykorzystania tego języka był wzrost wydajności obliczeń, a więc skrócenie czasu oczkowania na wyniki. Czas ten nie był dokładnie zmierzony ponieważ nie był on przedmiotem badań, jednak przyspieszenie obliczeń można szacować na 2–3 krotne.

Fakt, że język *C++* sprawdził się w tym zastosowaniu dobrze, nie przesądza o tym, czy jest to najlepsze z możliwych rozwiązań. Należy również wziąć pod uwagę, że zaobserwowany spadek liczby błędów w kodzie może być także spowodowany dużym doświadczeniem autora pracy w tej właśnie technologi.

Sprawą otwartą i wymagającą dalszych badań (a także w dużej mierze zależną od autora konkretnych badań) jest dobranie języka implementacji odpowiedniego do opisywanych badań. Język taki niewątpliwie powinien posiadać mechanizmy kontrolujące logiczną zwartość przetwarzanych informacji i operacji, a także pozwalać na modularyzację algorytmów. Wydaje się więc, że najlepszymi kandydatami będą silnie typowane języki obiektowe oraz silnie typowane języki funkcyjne (np. *Haskell).*

6.2.2 Modularyzacja

Ważnym aspektem, który należy wziąć pod uwagę, przy projektowaniu algorytmów, jest możliwość podzielenia ich na niezależne moduły, funkcje czy też podalgorytmy. Takie podejście pozwala nie tylko na analizę i testowanie kodu w częściach, ale także umożliwia dostrzeżenie podobieństw między różnymi stosowanymi algorytmami i wychwycenie rzeczywistych różnic między nimi występujących.

Podział algorytmu, powinien prawdopodobnie przebiegać wzdłuż kolejnych elementów operujących na danych w sposób możliwie niezależny od siebie. Wydaje się, że koncepcją którą może się sprawdzić (i dla której *proof of concept* stanowiła implementacja w *C++* algorytmów w ramach badań w tej pracy), jest podział algorytmu na niezależne części, które operują na stanie algorytmu i przekazują jego zmodyfikowaną wersję dalej. Pozwala to w idealnym wypadku na konstruowanie algorytmów z niezależnych, raz zaimplementowanych bloków. To z kolei prowadzi do powtarzalności implementacji i daje możliwość poświęcenia większej ilości czasu na badanie poprawności tych właśnie bloków.

Warto odnotować, że do takiego charakteru przetwarzania idealnie zdają się pasować języki funkcyjne. To właśnie one wydają się być godną zbadania alternatywą dla rozwiązań obiektowych w kontekście algorytmów ewolucji różnicowej. Elementem oczekującym na dalsze zbadanie i takim, który może okazać się kluczowy dla rozwinięcia tej idei, jest poszukiwanie rozsądnych linii podziału między kolejnymi modułami w algorytmie.

Innym ale równie istotnym aspektem wymagającym modularyzacji, jest konieczność możliwe dużej separacji wykonywanego algorytmu od mechanizmów zbierania i przetwarzania danych wynikowych. Częstą praktyką jest łączenie tych dwóch elementów, co może prowadzić do utraty czytelności implementacji jak i utrudniać katalogowanie i analizowanie danych zebranych przy badaniach. Ponieważ dane wynikowe dla większości algorytmów z badanej grupy będą miały podobny charakter, wydaje się również że uniezależnienie ich zbierania od samego algorytmu, pozwoliło by wielokrotnie wykorzystywać te same rozwiązania – co dodatkowo oszczędziło by czas badaczom.

Testowanie

Programowanie jest dziedziną, która wiąże się z dużą ilością błędów. Ze względu na to, że niezależnie od tego co jest implementowane, błędy w kodzie są jego nierozłącznym elementem, informatyka wypracowała mechanizmy pozwalające ograniczyć ich problematyczność do minimum.

Poza silnie typowanymi, chroniącymi przed błędami już na poziomie składni językami, o których mowa w rozdziale 6.2.1, informatyka proponuje szereg rozwiązań operujących na gotowym kodzie. Dwa najważniejsze to:

testowanie – testy aplikacji polegają na przetwarzaniu z góry przygotowanych scenariuszy użycia kodu i porównywaniu uzyskanych wyników ze znanymi oczekiwanymi. Popularne testy regresji, polegają na testowaniu po każdej zmianie kodu, czy do tej pory przewidziane scenariusze nadal funkcjonują w sposób prawidłowy. Ważnym jest też opisywanie w formie scenariuszy testowych takich danych wejściowych, które w przeszłości doprowadzały do wystąpienia błędów. Z punktu widzenia testowania modularyzacja zdaje się być kluczowa. Ze względu bowiem na złożony charakter algorytmów, ich całościowe testowanie jest problematyczne. Testy jednostkowe, wy-

dzielające pod problemy do testowania, pozwalają dokładnie zbadać właściwości kolejnych operatorów nakładanych na dane i w ten sposób znacząco poprawić ich jakość.

Kwestią wymagającą oddzielnych badań wydają się jednak być testy integracyjne – a więc takie polegające na sprawdzeniu całości algorytmu. Z jednej strony, wydaje się, że mechanizmy łączenia modułów w kompletny algorytm można przetestować na przykładzie prostego i deterministycznego algorytmu. Z drugiej jednak nieoczywistym jest, że proste algorytmy będą ujawniały te same błędne cechy kodu co algorytmy bardziej złożone. Zbadanie możliwości wykorzystania różnych istniejących technik (lub stworzenia nowych) tworzenia testów integracyjnych, jawi się więc jako jeden z istotniejszych kierunków badań służących poprawie zaufania do uzyskiwanych w zastosowanej metodyce wyników.

Odnotować należy również możliwość prowadzenia badań nad algorytmami z wykorzystaniem technik *Test Driven Development*, jako że takie rozwiązanie w wielu źródłach podawane jest jako sposób na znaczącą poprawę jakości kodu [35].

dowodzenie poprawności – informatyka wypracowała również metody matematycznego dowodzenia poprawności kodu. Choć techniki te nie są doskonałe, i jednocześnie są czasochłonne (i z tych powodów rzadko stosuje się je w rzeczywistych rozwiązaniach), wydają się one doskonale odpowiadać potrzebom tego rodzaju badań. Sposób skorzystania z tych możliwości informatyki, to kolejny z otwartych kierunków badań nad metodyką pracy przy omawianych problemach.

6.3 Narzędzia wsparcia

Ostatnią wartą odnotowania obserwacją dotyczącą stosowanej metodyki pracy, jest niska dostępność narzędzi wsparcia. Oznacza to, że mimo sporej społeczności badaczy, zajmujących się tematyką, a także dużej powtarzalności narzędzi wykorzystywanych w badaniach, nie został skonstruowany wspólny ogólnodostępny ich zestaw. W praktyce powoduje to, że każdy z badaczy, poświęca własny czas badań, na przygotowanie pobocznych narzędzi nie będących ich przedmiotem. Jednocześnie należy zwrócić uwagę na fakt, że zwiększa to także ryzyko błędu w implementacji.

Choć współczesne pakiety umożliwiające analizę danych jak *R*, dostarczają łatwo dostępnych wysokopoziomowych funkcji analizy danych, to nie jest to jedyna wspólna część narzędzi dla różnych badaczy. Spośród najistotniejszych braków w bazie narzędziowej należało by bowiem wymienić:

Narzędzie planowania eksperymentów – elementem niezmiennym dla wszystkich omawianym badań jest planowanie eksperymentów w oparciu o dostarczony algorytm. Niezależnie od parametrów i typu algorytmu, wiele części planowania wygląda podobnie. Dobierane są konkretne zestawy parametrów i zbierane podobne dane. Wydaje się więc możliwym, stworzenie systemu korzystającego ze wspólnych cech badanych algorytmów, i dostarczającego podstawowych operacji takich jak tworzenie zestawu testów i przeglądanie ich wyników.

Narzędzie zbieranie danych – oprócz planowania eksperymentów istotny jest także sposób składowania danych, który co do zasady powinien być podobny dla większości algorytmów. Dostarczając rozbudowaną implementację zbierania danych i łącząc ją z narzędziem do planowania eksperymentów, można się spodziewać, że cały proces badawczy przetwarzający algorytm na gotowe analizy danych wymagające wyłącznie ręcznego opisania i wstawienia do treści artykułu, mógłby zostać zautomatyzowany.

Narzędzie budowy algorytmów – ostatnim elementem, który mógłby powstać w oparciu o ideę modularnych algorytmów, jest narzędzie pozwalające tworzyć algorytmy w bardziej przejrzystej formie wykorzystując gotowe bloki i dopisując własne. Narzędzie takie by było bardziej użyteczne, niż przejrzyste API posiadające podobne właściwości musiało by być rozbudowane pod kątem interfejsu użytkownika. Stworzenie takiego narzędzia nie wydaje się więc być sprawą priorytetową, jednak na pewno jest to warta odnotowania droga rozwoju dla społeczności badaczy.

Odnosząc się raz jeszcze do tematu narzędzi wsparcia, można domniemywać, że tego rodzaju zestaw narzędzi stanowił by istotne wsparcie dla badaczy, a zarazem poprawił by jakość i spójność badań. Ponadto można podejrzewać, że uogólnienie niektórych problemów do postaci wystarczającej dla większości badaczy, jest dużym wyzwaniem teoretycznym, mogącym się przyczynić do rozwoju teorii i praktyki różnych dziedzin informatyki.

Rozdział 7

Podsumowanie

Cel pracy został osiągnięty. Skonstruowany został algorytm *NADE*, dodający do klasycznego schematu ewolucji różnicowej zewnętrzne archiwum wykorzystywane w procesie mutacji różnicowej. Algorytm *NADE* został porównany z klasycznym algorytmem *DE/rand/1/bin* pod kątem jakości uzyskiwanych wyników, dla różnych wartości nowego parametru H. Tym samym zbadano wpływ zaproponowanego archiwum na jakość uzyskiwanych wyników w mutacji różnicowej.

Uzyskane wyniki wskazały, że archiwum w zaproponowanej formie wpływa w sposób statystycznie istotny na otrzymywane wyniki. Zostało stwierdzone, że choć istnieją problemy, dla których zaproponowana modyfikacja może prowadzić do poprawy uzyskiwanych wyników, to nie ma jasnych przesłanek, według których można by wskazać te problemy. Zgodnie z otrzymanymi rezultatami, należy także przyjąć, iż dla większości funkcji z testu *CEC2013 benchmark*, algorytm *NADE* działa mniej skutecznie niż podstawowy wariant ewolucji różnicowej *DE/rand/1/bin*. Wskazano, że choć wpływ archiwum na uzyskiwane wyniki zależy od badanej funkcji, to traktując funkcje z testu *CEC2013 benchmark* jako reprezentatywną próbkę problemów, archiwum w opisanej formie nie jest pożądaną modyfikacją algorytmu *DE/rand/1/bin* dla całkowicie nieznanego problemu.

Zbadana została możliwość zastosowania znormalizowanego archiwum w wypracowanej formie w zaawansowanej wersji algorytmu ewolucji różnicowej *SHADE* [10]. Został skonstruowany algorytm *ANADE* stanowiący rozszerzenie algorytmu *SHADE* o takie właśnie archiwum, który jakością rozwiązań w ogólnym przypadku nieznacznie odbiegał od swojego pierwowzoru w sposób negatywny. Skłonność proponowanej modyfikacji do statystycznie istotnej zmiany uzyskiwanych wyników w sposób zależny od problemu została potwierdzona. W przypadku algorytmu *ANADE* było to jednak mniej wyraźne zjawisko.

Mając na względzie powyższe wnioski, wydaje się, że dalsze badanie archiwum jako niezależnej od wariantu ewolucji różnicowej metody poprawy jakości jej działania, nie są interesującym kierunkiem rozwoju.

Dodatkowo w ramach pracy w rozdziale 6 poruszono wątki dotyczące metodyki badań i możliwości ich dalszego rozwoju. Omówiono trudności związane z weryfikowalnością implementacji algorytmów, dokładnością oraz czasochłonnością badań. Ten poboczny wątek, który nie był pierwotnym celem pracy, wskazuje kluczowe problemy przy badaniach tego rodzaju i sugeruje cechy jakie powinny posiadać rozwiązania tychże problemów. Metodyka pracy nad niedeterministycznymi algorytmami optymalizującymi, zdaje się być

ważnym punktem dla rozwoju dziedziny, a zarazem okazją do rozwijających technologię informacyjną badań zarówno o charakterze teoretycznym jak i inżynierskim.

Rozdział 8

Załącznik - wyniki eksperymentów

Tablica 8.1: Średnie błędy algorytmu *DE/rand/1/bin* oraz pierwszej wersji algorytmu *NADE* (bez normalizacji punktów w archiwum) dla $H \in \{90, 120, 240\}$ i funkcji z *CEC2013 benchmark*. Dodatkowo podano wyniki testu Wilcoxona dla par, dla uzyskiwanych wyników względem *DE/rand/1/bin*. Wymiarowość przestrzeni przeszukiwań $D = 30$.

	DE/rand/1/bin	**NADE H=90**	**NADE H=120**	**NADE = 240**
F1	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)
F2	1.76e+05 (=)	3.44e+05 (-)	4.46e+05 (-)	4.88e+06 (-)
F3	1.88e+06 (=)	9.75e+04 (+)	4.11e+04 (+)	7.34e+05 (=)
F4	7.76e+02 (=)	7.01e+02 (=)	1.55e+03 (-)	2.61e+04 (-)
F5	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)
F6	2.04e+01 (=)	1.44e+01 (+)	1.40e+01 (+)	1.46e+01 (+)
F7	1.57e+00 (=)	1.73e-01 (+)	6.41e-02 (+)	1.53e+01 (-)
F8	2.09e+01 (=)	2.10e+01 (=)	2.09e+01 (=)	2.09e+01 (=)
F9	1.06e+01 (=)	7.40e+00 (+)	2.66e+01 (-)	3.81e+01 (-)
F10	7.15e-02 (=)	1.31e-02 (+)	4.20e-03 (+)	7.80e-01 (-)
F11	1.87e+01 (=)	1.75e+01 (=)	1.17e+02 (-)	1.81e+02 (-)
F12	1.59e+02 (=)	1.70e+02 (=)	1.83e+02 (-)	2.00e+02 (-)
F13	1.70e+02 (=)	1.72e+02 (=)	1.75e+02 (-)	1.98e+02 (-)
F14	2.90e+03 (=)	3.35e+03 (=)	5.56e+03 (-)	6.66e+03 (-)
F15	7.10e+03 (=)	7.09e+03 (=)	7.11e+03 (=)	7.23e+03 (-)
F16	2.41e+00 (=)	2.45e+00 (=)	2.43e+00 (=)	2.60e+00 (=)
F17	1.10e+02 (=)	1.27e+02 (-)	1.74e+02 (-)	2.14e+02 (-)
F18	2.01e+02 (=)	2.01e+02 (=)	2.10e+02 (-)	2.34e+02 (-)
F19	9.19e+00 (=)	1.29e+01 (-)	1.46e+01 (-)	1.65e+01 (-)
F20	1.19e+01 (=)	1.20e+01 (=)	1.21e+01 (-)	1.26e+01 (-)
F21	3.09e+02 (=)	2.93e+02 (=)	2.91e+02 (=)	2.91e+02 (=)
F22	2.56e+03 (=)	2.82e+03 (=)	5.24e+03 (-)	6.75e+03 (-)
F23	7.08e+03 (=)	7.05e+03 (=)	7.06e+03 (=)	7.24e+03 (-)
F24	2.13e+02 (=)	2.01e+02 (+)	2.02e+02 (+)	2.15e+02 (=)
F25	2.44e+02 (=)	2.39e+02 (+)	2.39e+02 (+)	2.52e+02 (-)
F26	2.17e+02 (=)	2.50e+02 (-)	2.54e+02 (-)	2.08e+02 (-)
F27	4.75e+02 (=)	4.05e+02 (+)	3.48e+02 (+)	8.30e+02 (-)
F28	3.00e+02 (=)	3.00e+02 (=)	3.00e+02 (=)	3.00e+02 (=)

Tablica 8.2: Średnie błędy algorytmu *DE/rand/1/bin* algorytmu *NADE* dla $H \in \{90, 120, 240\}$ i funkcji z *CEC2013 benchmark*. Dodatkowo podano wyniki testu Wilcoxona dla par, dla uzyskiwanych wyników względem *DE/rand/1/bin*. Wymiarowość przestrzeni przeszukiwań $D = 30$.

	DE/rand/1/bin	**NADE H=90**	**NADE H=120**	**NADE H=240**
F1	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)
F2	1.76e+05 (=)	3.78e+05 (-)	2.97e+05 (-)	7.06e+05 (-)
F3	1.88e+06 (=)	1.30e+06 (+)	6.45e+00 (+)	3.60e-03 (+)
F4	7.76e+02 (=)	4.35e+02 (+)	2.90e+02 (+)	6.53e+02 (=)
F5	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)
F6	2.04e+01 (=)	1.92e+01 (+)	1.53e+01 (+)	9.37e+00 (+)
F7	1.57e+00 (=)	3.09e-01 (+)	1.86e-01 (+)	3.10e-01 (+)
F8	2.09e+01 (=)	2.10e+01 (=)	2.09e+01 (=)	2.09e+01 (=)
F9	1.06e+01 (=)	1.19e+01 (=)	2.36e+01 (-)	3.87e+01 (-)
F10	7.15e-02 (=)	2.76e-02 (+)	1.22e-02 (+)	0.00e+00 (+)
F11	1.87e+01 (=)	8.85e+01 (-)	1.60e+02 (-)	1.70e+02 (-)
F12	1.59e+02 (=)	1.71e+02 (=)	1.78e+02 (-)	1.87e+02 (-)
F13	1.70e+02 (=)	1.73e+02 (=)	1.78e+02 (-)	1.91e+02 (-)
F14	2.90e+03 (=)	6.46e+03 (-)	6.53e+03 (-)	6.65e+03 (-)
F15	7.10e+03 (=)	7.15e+03 (=)	7.21e+03 (-)	7.16e+03 (=)
F16	2.41e+00 (=)	2.50e+00 (=)	2.52e+00 (=)	2.40e+00 (=)
F17	1.10e+02 (=)	1.92e+02 (-)	1.93e+02 (-)	2.03e+02 (-)
F18	2.01e+02 (=)	2.01e+02 (=)	2.06e+02 (=)	2.20e+02 (-)
F19	9.19e+00 (=)	1.42e+01 (-)	1.52e+01 (-)	1.59e+01 (-)
F20	1.19e+01 (=)	1.21e+01 (-)	1.21e+01 (-)	1.23e+01 (-)
F21	3.09e+02 (=)	3.20e+02 (=)	3.08e+02 (=)	3.01e+02 (=)
F22	2.56e+03 (=)	6.55e+03 (-)	6.73e+03 (-)	6.72e+03 (-)
F23	7.08e+03 (=)	7.15e+03 (=)	7.08e+03 (=)	7.19e+03 (-)
F24	2.13e+02 (=)	2.02e+02 (+)	2.02e+02 (+)	2.01e+02 (+)
F25	2.44e+02 (=)	2.41e+02 (+)	2.41e+02 (+)	2.41e+02 (+)
F26	2.17e+02 (=)	2.39e+02 (-)	2.42e+02 (-)	2.29e+02 (-)
F27	4.75e+02 (=)	3.70e+02 (+)	3.77e+02 (+)	4.48e+02 (+)
F28	3.00e+02 (=)	3.00e+02 (=)	3.00e+02 (=)	3.00e+02 (=)

Tablica 8.3: Średnie błędy algorytmu *DE/rand/1/bin* ($\mu = 40$) algorytmu *NADE* dla $\mu = 40$, $H = 60$ i funkcji z *CEC2013 benchmark*. Dodatkowo podano wyniki testu Wilcoxona dla par, dla uzyskiwanych wyników względem *DE/rand/1/bin*. Wymiarowość przestrzeni przeszukiwań $D = 30$.

	DE/rand/1/bin	**NADE H=60**
F1	8.00e-04 (=)	0.00e+00 (=)
F2	1.35e+07 (=)	1.11e+07 (=)
F3	2.11e+01 (=)	3.40e+00 (=)
F4	9.88e+00 (=)	1.99e+00 (=)
F5	1.22e+01 (=)	6.74e+00 (+)
F6	2.35e+01 (=)	1.18e+01 (+)
F7	1.45e+02 (=)	1.64e+02 (+)
F8	6.98e+03 (=)	7.09e+03 (=)
F9	5.84e+01 (=)	1.84e+02 (+)
F10	4.36e+00 (=)	1.22e+01 (+)
F11	2.84e+02 (=)	3.44e+02 (+)
F12	6.84e+03 (=)	6.98e+03 (-)
F13	2.55e+02 (=)	2.44e+02 (=)
F14	6.32e+02 (=)	4.35e+02 (-)
F15	8.00e-04 (=)	0.00e+00 (=)
F16	1.35e+07 (=)	1.11e+07 (=)
F17	2.11e+01 (=)	3.40e+00 (-)
F18	9.88e+00 (=)	1.99e+00 (=)
F19	1.22e+01 (=)	6.74e+00 (-)
F20	2.35e+01 (=)	1.18e+01 (=)
F21	1.45e+02 (=)	1.64e+02 (-)
F22	6.98e+03 (=)	7.09e+03 (-)
F23	5.84e+01 (=)	1.84e+02 (=)
F24	4.36e+00 (=)	1.22e+01 (+)
F25	2.84e+02 (=)	3.44e+02 (+)
F26	6.84e+03 (=)	6.98e+03 (=)
F27	2.55e+02 (=)	2.44e+02 (+)
F28	6.32e+02 (=)	4.35e+02 (+)

Tablica 8.4: Średnie błędy algorytmu *DE/rand/1/bin* ($\mu = 60$) algorytmu *NADE* dla $\mu = 40$, $H = 60$ i funkcji z *CEC2013 benchmark*. Dodatkowo podano wyniki testu Wilcoxona dla par, dla uzyskiwanych wyników względem *DE/rand/1/bin*. Wymiarowość przestrzeni przeszukiwań $D = 30$.

	DE/rand/1/bin	**NADE**
F1	0.00e+00 (=)	0.00e+00 (=)
F2	1.88e+06 (=)	1.11e+07 (=)
F3	0.00e+00 (=)	3.40e+00 (-)
F4	1.57e+00 (=)	1.99e+00 (=)
F5	1.06e+01 (=)	6.74e+00 (-)
F6	1.87e+01 (=)	1.18e+01 (=)
F7	1.70e+02 (=)	1.64e+02 (=)
F8	7.10e+03 (=)	7.09e+03 (=)
F9	1.10e+02 (=)	1.84e+02 (+)
F10	9.19e+00 (=)	1.22e+01 (-)
F11	3.09e+02 (=)	3.44e+02 (+)
F12	7.08e+03 (=)	6.98e+03 (=)
F13	2.44e+02 (=)	2.44e+02 (+)
F14	4.75e+02 (=)	4.35e+02 (-)
F15	0.00e+00 (=)	0.00e+00 (=)
F16	1.88e+06 (=)	1.11e+07 (=)
F17	0.00e+00 (=)	3.40e+00 (-)
F18	1.57e+00 (=)	1.99e+00 (=)
F19	1.06e+01 (=)	6.74e+00 (-)
F20	1.87e+01 (=)	1.18e+01 (+)
F21	1.70e+02 (=)	1.64e+02 (=)
F22	7.10e+03 (=)	7.09e+03 (-)
F23	1.10e+02 (=)	1.84e+02 (=)
F24	9.19e+00 (=)	1.22e+01 (=)
F25	3.09e+02 (=)	3.44e+02 (=)
F26	7.08e+03 (=)	6.98e+03 (-)
F27	2.44e+02 (=)	2.44e+02 (=)
F28	4.75e+02 (=)	4.35e+02 (=)

Tablica 8.5: Średnie błędy algorytmu *SHADE* i *ANADE* dla ustawień zgodnych z [10] i funkcji z *CEC2013 benchmark*. Dodatkowo podano wyniki testu Wilcoxona dla par, dla uzyskiwanych wyników względem *SHADE*. Wymiarowość przestrzeni przeszukiwań $D = 30$.

	SHADE	**ANADE**
F1	0.00e+00 (=)	0.00e+00 (=)
F2	8.92e+03 (=)	9.52e+03 (=)
F3	7.09e+04 (=)	3.60e+05 (=)
F4	0.00e+00 (=)	8.00e-04 (=)
F5	0.00e+00 (=)	0.00e+00 (=)
F6	1.06e+00 (=)	1.06e+00 (=)
F7	2.38e+00 (=)	2.21e+00 (=)
F8	2.08e+01 (=)	2.09e+01 (-)
F9	2.79e+01 (=)	2.72e+01 (=)
F10	7.60e-02 (=)	9.36e-02 (=)
F11	0.00e+00 (=)	0.00e+00 (=)
F12	2.22e+01 (=)	2.31e+01 (=)
F13	4.78e+01 (=)	5.19e+01 (=)
F14	1.50e-02 (=)	4.07e-01 (-)
F15	3.07e+03 (=)	3.18e+03 (=)
F16	9.72e-01 (=)	9.54e-01 (=)
F17	3.04e+01 (=)	3.04e+01 (=)
F18	7.16e+01 (=)	7.15e+01 (=)
F19	1.33e+00 (=)	1.34e+00 (=)
F20	1.06e+01 (=)	1.09e+01 (-)
F21	2.85e+02 (=)	2.95e+02 (=)
F22	9.19e+01 (=)	1.13e+02 (-)
F23	3.40e+03 (=)	3.47e+03 (=)
F24	2.06e+02 (=)	2.04e+02 (=)
F25	2.59e+02 (=)	2.62e+02 (=)
F26	2.04e+02 (=)	2.00e+02 (=)
F27	3.78e+02 (=)	3.49e+02 (+)
F28	2.92e+02 (=)	3.00e+02 (=)

Tablica 8.6: Średnie błędy algorytmu *DE/rand/1/bin* ($\mu = 50$), algorytmu *NADE* dla $H \in \{75, 100, 200\}$ i funkcji z *CEC2013 benchmark*. Dodatkowo podano wyniki testu Wilcoxona dla par, dla uzyskiwanych wyników względem *DE/rand/1/bin*. Wymiarowość przestrzeni przeszukiwań $D = 10$.

	DE/rand/1/bin	**NADE H=75**	**NADE H=100**	**NADE H=200**
F1	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)
F2	5.06e+03 (=)	1.16e+02 (+)	0.00e+00 (+)	0.00e+00 (+)
F3	2.64e+02 (=)	1.57e-01 (=)	4.02e-01 (=)	9.16e-02 (=)
F4	6.52e+01 (=)	5.21e+00 (+)	2.80e-03 (+)	0.00e+00 (+)
F5	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)
F6	4.35e+00 (=)	5.05e+00 (=)	4.13e+00 (=)	6.28e+00 (=)
F7	5.68e-03 (=)	0.00e+00 (+)	0.00e+00 (+)	0.00e+00 (+)
F8	2.04e+01 (=)	2.04e+01 (=)	2.04e+01 (=)	2.03e+01 (+)
F9	1.33e+00 (=)	6.85e-01 (+)	5.89e-01 (+)	5.89e-01 (+)
F10	5.53e-02 (=)	2.41e-01 (-)	3.99e-01 (-)	4.95e-01 (-)
F11	2.56e+00 (=)	1.37e+01 (-)	1.66e+01 (-)	2.02e+01 (-)
F12	1.24e+01 (=)	2.12e+01 (-)	2.49e+01 (-)	2.58e+01 (-)
F13	1.38e+01 (=)	2.08e+01 (-)	2.44e+01 (-)	2.65e+01 (-)
F14	4.24e+02 (=)	9.91e+02 (-)	1.03e+03 (-)	1.03e+03 (-)
F15	1.07e+03 (=)	1.30e+03 (-)	1.27e+03 (-)	1.36e+03 (-)
F16	1.06e+00 (=)	1.13e+00 (=)	1.14e+00 (=)	1.17e+00 (-)
F17	1.67e+01 (=)	2.96e+01 (-)	3.05e+01 (-)	3.06e+01 (-)
F18	3.14e+01 (=)	3.45e+01 (-)	3.34e+01 (-)	3.59e+01 (-)
F19	1.09e+00 (=)	1.83e+00 (-)	1.85e+00 (-)	1.87e+00 (-)
F20	2.31e+00 (=)	2.36e+00 (=)	2.30e+00 (=)	2.52e+00 (-)
F21	3.92e+02 (=)	4.00e+02 (=)	4.00e+02 (=)	4.00e+02 (=)
F22	3.10e+02 (=)	9.12e+02 (-)	9.58e+02 (-)	1.01e+03 (-)
F23	9.81e+02 (=)	1.14e+03 (=)	1.27e+03 (-)	1.28e+03 (-)
F24	2.01e+02 (=)	2.03e+02 (=)	2.04e+02 (=)	2.03e+02 (=)
F25	2.02e+02 (=)	2.02e+02 (=)	2.02e+02 (=)	2.02e+02 (=)
F26	1.45e+02 (=)	1.84e+02 (-)	1.72e+02 (-)	1.50e+02 (=)
F27	3.30e+02 (=)	3.50e+02 (=)	3.21e+02 (=)	3.07e+02 (=)
F28	2.84e+02 (=)	3.00e+02 (=)	3.00e+02 (=)	3.00e+02 (=)

Tablica 8.7: Średnie błędy algorytmu *SHADE* i *ANADE* dla ustawień zgodnych z [10] i funkcji z *CEC2013 benchmark*. Dodatkowo podano wyniki testu Wilcoxona dla par, dla uzyskiwanych wyników względem *SHADE*. Wymiarowość przestrzeni przeszukiwań $D = 10$.

	SHADE	**NADE**
F1	0.00e+00 (=)	0.00e+00 (=)
F2	0.00e+00 (=)	0.00e+00 (=)
F3	2.80e-03 (=)	3.10e-01 (=)
F4	0.00e+00 (=)	0.00e+00 (=)
F5	0.00e+00 (=)	0.00e+00 (=)
F6	8.24e+00 (=)	7.85e+00 (=)
F7	3.08e-03 (=)	5.36e-03 (=)
F8	2.04e+01 (=)	2.03e+01 (=)
F9	3.32e+00 (=)	3.62e+00 (=)
F10	1.29e-02 (=)	1.35e-02 (=)
F11	0.00e+00 (=)	0.00e+00 (=)
F12	3.94e+00 (=)	3.26e+00 (+)
F13	3.27e+00 (=)	3.23e+00 (=)
F14	5.00e-03 (=)	2.22e-03 (-)
F15	4.06e+02 (=)	4.29e+02 (=)
F16	7.36e-01 (=)	6.95e-01 (=)
F17	1.01e+01 (=)	1.01e+01 (=)
F18	1.74e+01 (=)	1.70e+01 (=)
F19	3.17e-01 (=)	3.32e-01 (=)
F20	2.26e+00 (=)	2.06e+00 (+)
F21	4.00e+02 (=)	4.00e+02 (=)
F22	4.02e+00 (=)	1.82e+01 (-)
F23	3.97e+02 (=)	4.43e+02 (=)
F24	1.98e+02 (=)	1.85e+02 (+)
F25	2.00e+02 (=)	1.97e+02 (=)
F26	1.24e+02 (=)	1.28e+02 (=)
F27	3.00e+02 (=)	3.00e+02 (=)
F28	3.00e+02 (=)	3.00e+02 (=)

Tablica 8.8: Średnie błędy algorytmu *DE/rand/1/bin* ($\mu = 100$), algorytmu *NADE* dla $H \in \{90, 120, 240\}$ i funkcji z *CEC2013 benchmark*. Dodatkowo podano wyniki testu Wilcoxona dla par, dla uzyskiwanych wyników względem *DE/rand/1/bin*. Wymiarowość przestrzeni przeszukiwań $D = 30$.

	DE/rand/1/bin	**NADE H=90**	**NADE H=120**	**NADE H=240**
F1	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)
F2	1.72e+05 (=)	3.75e+05 (-)	2.88e+05 (-)	6.89e+05 (-)
F3	2.91e+06 (=)	1.32e+06 (+)	8.53e+00 (+)	3.33e-03 (+)
F4	7.42e+02 (=)	4.33e+02 (+)	2.96e+02 (+)	6.55e+02 (=)
F5	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)
F6	2.04e+01 (=)	1.88e+01 (+)	1.51e+01 (+)	9.37e+00 (+)
F7	1.52e+00 (=)	2.96e-01 (+)	1.76e-01 (+)	3.34e-01 (+)
F8	2.09e+01 (=)	2.10e+01 (=)	2.09e+01 (=)	2.09e+01 (=)
F9	1.06e+01 (=)	1.23e+01 (=)	2.34e+01 (-)	3.88e+01 (-)
F10	7.09e-02 (=)	2.79e-02 (+)	1.26e-02 (+)	0.00e+00 (+)
F11	1.87e+01 (=)	9.14e+01 (-)	1.59e+02 (-)	1.69e+02 (-)
F12	1.60e+02 (=)	1.71e+02 (=)	1.78e+02 (-)	1.86e+02 (-)
F13	1.71e+02 (=)	1.74e+02 (=)	1.78e+02 (-)	1.90e+02 (-)
F14	2.88e+03 (=)	6.46e+03 (-)	6.56e+03 (-)	6.67e+03 (-)
F15	7.12e+03 (=)	7.10e+03 (=)	7.18e+03 (=)	7.18e+03 (=)
F16	2.43e+00 (=)	2.51e+00 (=)	2.52e+00 (=)	2.40e+00 (=)
F17	1.08e+02 (=)	1.92e+02 (-)	1.94e+02 (-)	2.02e+02 (-)
F18	2.01e+02 (=)	2.01e+02 (=)	2.07e+02 (-)	2.19e+02 (-)
F19	9.32e+00 (=)	1.43e+01 (-)	1.52e+01 (-)	1.59e+01 (-)
F20	1.19e+01 (=)	1.21e+01 (-)	1.21e+01 (-)	1.23e+01 (-)
F21	3.04e+02 (=)	3.18e+02 (=)	3.13e+02 (=)	2.97e+02 (=)
F22	2.61e+03 (=)	6.55e+03 (-)	6.73e+03 (-)	6.73e+03 (-)
F23	7.09e+03 (=)	7.14e+03 (=)	7.08e+03 (=)	7.19e+03 (-)
F24	2.13e+02 (=)	2.02e+02 (+)	2.02e+02 (+)	2.01e+02 (+)
F25	2.45e+02 (=)	2.41e+02 (+)	2.41e+02 (+)	2.41e+02 (+)
F26	2.20e+02 (=)	2.40e+02 (-)	2.39e+02 (=)	2.27e+02 (-)
F27	4.80e+02 (=)	3.79e+02 (+)	3.85e+02 (+)	4.37e+02 (+)
F28	3.00e+02 (=)	3.00e+02 (=)	3.00e+02 (=)	3.00e+02 (=)

Tablica 8.9: Średnie błędy algorytmu *SHADE* i *ANADE* dla ustawień zgodnych z [10] i funkcji z *CEC2013 benchmark*. Dodatkowo podano wyniki testu Wilcoxona dla par, dla uzyskiwanych wyników względem *SHADE*. Wymiarowość przestrzeni przeszukiwań $D = 30$.

	SHADE	**NADE**
F1	0.00e+00 (=)	0.00e+00 (=)
F2	9.92e+03 (=)	9.76e+03 (=)
F3	6.57e+04 (=)	3.34e+05 (=)
F4	0.00e+00 (=)	7.41e-04 (=)
F5	0.00e+00 (=)	0.00e+00 (=)
F6	1.96e+00 (=)	1.96e+00 (=)
F7	2.68e+00 (=)	2.12e+00 (=)
F8	2.08e+01 (=)	2.10e+01 (-)
F9	2.78e+01 (=)	2.73e+01 (=)
F10	7.80e-02 (=)	9.33e-02 (=)
F11	0.00e+00 (=)	0.00e+00 (=)
F12	2.20e+01 (=)	2.34e+01 (=)
F13	4.80e+01 (=)	5.20e+01 (=)
F14	1.62e-02 (=)	4.12e-01 (-)
F15	3.10e+03 (=)	3.20e+03 (=)
F16	9.72e-01 (=)	9.49e-01 (=)
F17	3.04e+01 (=)	3.04e+01 (=)
F18	7.18e+01 (=)	7.11e+01 (=)
F19	1.32e+00 (=)	1.33e+00 (=)
F20	1.05e+01 (=)	1.09e+01 (-)
F21	2.86e+02 (=)	2.95e+02 (=)
F22	9.30e+01 (=)	1.14e+02 (-)
F23	3.44e+03 (=)	3.46e+03 (=)
F24	2.05e+02 (=)	2.04e+02 (=)
F25	2.60e+02 (=)	2.62e+02 (=)
F26	2.04e+02 (=)	2.00e+02 (=)
F27	3.76e+02 (=)	3.50e+02 (+)
F28	2.93e+02 (=)	3.00e+02 (=)

Tablica 8.10: Średnie błędy algorytmu *DE/rand/1/bin* ($\mu = 100$), algorytmu *NADE* dla $H \in \{150, 200, 400\}$ i funkcji z *CEC2013 benchmark*. Dodatkowo podano wyniki testu Wilcoxona dla par, dla uzyskiwanych wyników względem *DE/rand/1/bin*. Wymiarowość przestrzeni przeszukiwań $D = 50$.

	DE/rand/1/bin	**NADE H=150**	**NADE H=200**	**NADE H=400**
F1	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)
F2	2.83e+06 (=)	9.21e+06 (-)	2.32e+07 (-)	8.06e+07 (-)
F3	1.35e+06 (=)	2.39e+05 (+)	3.28e+04 (+)	2.15e+07 (-)
F4	1.76e+04 (=)	2.54e+04 (-)	3.27e+04 (-)	5.07e+04 (-)
F5	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)	0.00e+00 (=)
F6	4.35e+01 (=)	4.36e+01 (-)	4.35e+01 (-)	4.34e+01 (=)
F7	3.42e+00 (=)	1.37e+00 (+)	3.54e+00 (=)	3.22e+01 (-)
F8	2.11e+01 (=)	2.11e+01 (-)	2.11e+01 (=)	2.11e+01 (=)
F9	7.21e+01 (=)	7.20e+01 (=)	7.19e+01 (=)	7.23e+01 (=)
F10	5.44e-02 (=)	3.54e-02 (+)	1.39e-02 (+)	1.11e+00 (-)
F11	2.09e+02 (=)	3.20e+02 (-)	3.45e+02 (-)	3.73e+02 (-)
F12	3.56e+02 (=)	3.61e+02 (-)	3.66e+02 (-)	3.97e+02 (-)
F13	3.51e+02 (=)	3.63e+02 (-)	3.70e+02 (-)	3.95e+02 (-)
F14	1.12e+04 (=)	1.28e+04 (-)	1.28e+04 (-)	1.30e+04 (-)
F15	1.38e+04 (=)	1.39e+04 (=)	1.37e+04 (=)	1.39e+04 (=)
F16	3.35e+00 (=)	3.34e+00 (=)	3.28e+00 (=)	3.40e+00 (=)
F17	3.38e+02 (=)	3.90e+02 (-)	3.99e+02 (-)	4.27e+02 (-)
F18	3.99e+02 (=)	4.08e+02 (-)	4.22e+02 (-)	4.38e+02 (-)
F19	2.95e+01 (=)	3.13e+01 (-)	3.20e+01 (-)	3.38e+01 (-)
F20	2.21e+01 (=)	2.22e+01 (=)	2.22e+01 (-)	2.24e+01 (-)
F21	6.31e+02 (=)	7.95e+02 (=)	6.00e+02 (=)	8.35e+02 (=)
F22	1.08e+04 (=)	1.30e+04 (-)	1.28e+04 (-)	1.31e+04 (-)
F23	1.38e+04 (=)	1.38e+04 (=)	1.37e+04 (=)	1.39e+04 (=)
F24	2.32e+02 (=)	2.12e+02 (+)	2.03e+02 (+)	2.38e+02 (=)
F25	2.78e+02 (=)	2.70e+02 (+)	2.78e+02 (=)	3.17e+02 (-)
F26	2.89e+02 (=)	3.35e+02 (-)	3.71e+02 (-)	4.40e+02 (-)
F27	6.93e+02 (=)	5.96e+02 (+)	8.60e+02 (-)	1.74e+03 (-)
F28	5.19e+02 (=)	4.00e+02 (=)	4.00e+02 (=)	4.00e+02 (=)

Tablica 8.11: Średnie błędy algorytmu *SHADE* i *ANADE* dla ustawień zgodnych z [10] i funkcji z *CEC2013 benchmark*. Dodatkowo podano wyniki testu Wilcoxona dla par, dla uzyskiwanych wyników względem *SHADE*. Wymiarowość przestrzeni przeszukiwań $D = 50$.

	SHADE	**NADE**
F1	0.00e+00 (=)	0.00e+00 (=)
F2	2.99e+04 (=)	3.00e+04 (=)
F3	1.72e+06 (=)	1.78e+06 (=)
F4	1.20e-03 (=)	3.60e-03 (=)
F5	0.00e+00 (=)	0.00e+00 (=)
F6	4.34e+01 (=)	4.24e+01 (=)
F7	2.19e+01 (=)	1.60e+01 (+)
F8	2.09e+01 (=)	2.10e+01 (-)
F9	5.54e+01 (=)	5.61e+01 (=)
F10	1.09e-01 (=)	1.49e-01 (-)
F11	0.00e+00 (=)	0.00e+00 (=)
F12	5.84e+01 (=)	5.60e+01 (=)
F13	1.53e+02 (=)	1.35e+02 (+)
F14	3.05e-02 (=)	2.72e+00 (-)
F15	6.86e+03 (=)	6.81e+03 (=)
F16	1.27e+00 (=)	1.25e+00 (=)
F17	5.08e+01 (=)	5.08e+01 (=)
F18	1.38e+02 (=)	1.37e+02 (=)
F19	2.63e+00 (=)	2.62e+00 (=)
F20	1.95e+01 (=)	1.96e+01 (=)
F21	7.95e+02 (=)	8.95e+02 (=)
F22	1.15e+01 (=)	2.22e+01 (-)
F23	7.57e+03 (=)	7.29e+03 (=)
F24	2.33e+02 (=)	2.30e+02 (=)
F25	3.34e+02 (=)	3.31e+02 (=)
F26	2.68e+02 (=)	2.62e+02 (=)
F27	8.77e+02 (=)	9.10e+02 (=)
F28	5.18e+02 (=)	4.00e+02 (=)

Rozdział 9

Bibliografia

[1] Xinjie Yu; Mitsuo Gen. *Introduction to Evolutionary Algorithms*. Decision Engineering. Springer, 2010.

[2] Jarosław Arabas; Paweł Cichosz. *Searching for Intelligent Behavior*.

[3] Zhangjun Huang, Mingxu Ma, and Chengen Wang. An archived differential evolution algorithm for constrained global optimization. In *Smart Manufacturing Application, 2008. ICSMA 2008. International Conference on*, pages 255–260, April 2008.

[4] Swagatam Das; Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION,*, 15(1):4–31, feb 2011.

[5] Jan Palczewski. *Matematyka Stosowana - Optymalizacja II*. Uniwersytet Warszawski, Wydział Matematyki, Informatyki i Mechaniki, 2014.

[6] Rainer Storn; Kenneth Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces, 1995.

[7] Karol Opara. *Analiza algorytmu ewolucji różnicowej i jego zastosowanie w wyznaczaniu zależności statystycznych*. PhD thesis, Instytut Badań Systemowych Polskiej Akademii Nauk, 2014.

[8] Tianjun Liao Ilya Loshchilov, Thomas Stuetzle. Ranking results of cec'13 special session & competition on real-parameter single objective optimization. 2013.

[9] Jingqiao Zhang and A.C. Sanderson. JADE: Adaptive differential evolution with optional external archive. *Evolutionary Computation, IEEE Transactions on*, 13(5):945–958, Oct 2009.

[10] R. Tanabe and A. Fukunaga. Success-history based parameter adaptation for differential evolution. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 71–78, June 2013.

[11] J. J. Liang; B-Y. Qu; P. N. Suganthan; Alfredo G. Hernández-Díaz. Problem definitions and evaluation criteria for the cec2013 special session on real-parameter optimization, 2013.

[12] Jarosław Arabas. *Wykłady z algorytmów ewolucyjnych*. Wydawnictwo WNT, Warszawa, 2004.

[13] Daniela Zaharie. *Differential Evolution: from Theoretical Analysis to Practical Insights*. 2012.

[14] Adam Stelmaszczyk. DE/mid – nowy wariant algorytmu ewolucji różnicowej wykorzystujący punkt środkowy populacji. Master's thesis, Warsaw University of Technology, 2014.

[15] J. Zhang and A.C. Sanderson. *Adaptive Differential Evolution: A Robust Approach to Multimodal Problem Optimization*. Adaptation, Learning, and Optimization. Springer, 2009.

[16] Jarosław Arabas; Adam Szczepankiewicz; Tomasz Wroniak. Experimental comparison of methods to handle boundary constraints in differential evolution. *Lecture Notes in Computer Science*, 6239:411–420, 2010.

[17] Zhangjun Huang, Chengen Wang, and Mingxu Ma. A robust archived differential evolution algorithm for global optimization problems. *JCP*, 4(2):160–167, 2009.

[18] U. Halder, S. Das, and D. Maity. A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments. *Cybernetics, IEEE Transactions on*, 43(3):881–897, June 2013.

[19] S. lloyd. Least square quantization in PCM. *IEEE Trans. Inform. Theory*, 28:129–137, 1982.

[20] Jingqiao Zhang and A.C. Sanderson. Self-adaptive multi-objective differential evolution with direction information provided by archived inferior solutions. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2801–2810, June 2008.

[21] Fei Peng, Ke Tang, Guoliang Chen, and Xin Yao. Multi-start jade with knowledge transfer for numerical optimization. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 1889–1895, May 2009.

[22] Wenyin Gong, Zhihua Cai, C.X. Ling, and Changhe Li. Enhanced differential evolution with adaptive strategies for numerical optimization. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 41(2):397–413, April 2011.

[23] Zhenyu Yang, Jingqiao Zhang, Ke Tang, Xin Yao, and A.C. Sanderson. An adaptive coevolutionary differential evolution algorithm for large-scale optimization. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 102–109, May 2009.

[24] Jingqiao Zhang, V. Avasarala, A.C. Sanderson, and T. Mullen. Differential evolution for discrete optimization: An experimental study on combinatorial auction problems. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2794–2800, June 2008.

[25] Richard Lowry. Concepts & applications of inferential statistics.

[26] R. M. Dudley. *Uniform Central Limit Theorems*. Cambridge, second edition, 2014. ISBN: 9780521498845.

[27] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.*, 18(1):50–60, 03 1947.

[28] Black Box Optimization Competition. *http://bbcomp.ini.rub.de/*. Dostęp: 26 kwietnia 2015.

[29] IEEE Task P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*. August 1985.

[30] F. Caraffini, F. Neri, Jixiang Cheng, Gexiang Zhang, L. Picinali, G. Iacca, and E. Mininno. Super-fit multicriteria adaptive differential evolution. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1678–1685, June 2013.

[31] S. Biswas, S. Kundu, S. Das, and A.V. Vasilakos. Teaching and learning best differential evoltuion with self adaptation for real parameter optimization. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1115–1122, June 2013.

[32] I. Poikolainen and F. Neri. Differential evolution with concurrent fitness based local search. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 384–391, June 2013.

[33] A.K. Qin and Xiaodong Li. Differential evolution on the CEC-2013 single-objective continuous optimization testbed. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1099–1106, June 2013.

[34] The R Project for Statistical Computing. https://www.r-project.org/.

[35] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1 edition, 2008.

[36] Daniela Zaharie. A comparative analysis of crossover variants in differential evolution. *Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 171–181, 2007.

[37] J. Ronkkonen, S. Kukkonen, and K.V. Price. Real-parameter optimization with differential evolution. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 506–513 Vol.1, Sept 2005.