Algorytmy metaheurystyczne

Projekt nr 4 - dokumentacja wstępna

Oskar Bartosz, Michał Matak

1 Wstęp

Tematem projektu jest dokonanie hybrydyzacji algorytmu NL-SHADE-RSP oraz DES, a następnie porównanie działania algorytmu z klasyczną wersją NL-SHADE-RSP oraz DES. W tym celu zostanie użyty benchmark CEC 2022. Oba algorytmy, zarówno DES jak i NL-SHADE-RSP są rozbudowanymi wersjami algorytmu ewolucji różnicowej i osiągają dobre wyniki w benchmarkach.

W wyniku połączenia udało się uzyskać algorytm który zawiera pozytywne cechy obu podejść, i dobrze radzi sobie w benchmarkach CEC, zmniejszając jego czas wykonywania w stosunku do NL-SHADE-RSP o niemal połowę, oraz zwiększając jakość wyników w stostunku do DES o istotną wartość.

2 Wprowadzenie Teoretyczne

Algorytmy różnicowe bazują na populacji potencjalnych rozwiązań, zwanych wektorami, które ewoluują poprzez iteracyjne procesy mutacji, krzyżowania i selekcji. Kluczowym elementem tych algorytmów jest operacja różnicowa, która generuje nowe rozwiązania poprzez kombinację istniejących wektorów populacji i różnicę między nimi. Główne kroki algorytmu różnicowego można opisać następująco:

1. **Inicjalizacja populacji**: Na początku generowana jest początkowa populacja losowych wektorów, reprezentujących różne możliwe rozwiązania problemu optymalizacji.

2. **Mutacja**: Dla każdego wektora z populacji tworzone są wektory mutantów poprzez dodanie różnicy między dwoma losowo wybranymi wektorami do trzeciego wektora.

3. **Krzyżowanie**: Następnie, wektor mutant jest mieszany z wektorem bazowym, tworząc wektor potomny, który dziedziczy cechy obu rodziców.

4. **Selekcja**: Na koniec, wektor potomny jest porównywany z wektorem bazowym, a lepsze rozwiązanie (tj. to, które minimalizuje lub maksymalizuje funkcję celu) jest zachowywane do następnej generacji.

Algorytmy różnicowe są szczególnie skuteczne w optymalizacji globalnej, gdzie celem jest znalezienie najlepszego możliwego rozwiązania w dużych, wielowymiarowych przestrzeniach poszukiwań. Dzięki zdolności do eksploracji całej przestrzeni poszukiwań oraz wykorzystaniu informacji z poprzednich iteracji, algorytmy te mogą unikać lokalnych minimów i znajdować rozwiązania bliskie optymalnym. Są także łatwo dostosowywalne do optymalizowanego problemu.

2.1 DES

Algorytm DES wprowadza zmianę do DE, polegającą na tym, że w generacji kolejnych osobników w następnych populacjach wprowadza się przesunięcie względem średniej najlepszej części populacji a średniej całej populacji. Algorytm jest pozbawiony krzyżowania, zamiast tego każdy kolejny osobnik tworzony jest z dodania do średniej najlepszych osobników różnicy między losowymi najlepszymi osobnikami z pewnego okna czasowego, pędu oraz szumu powiązanego z bezpośrednio warunkiem końca (po osiągnięciu przez populację takiej wariancji, jaka odpowiada temu szumowi, algorytm się zatrzymuje).

Pozwala to na osiągnięcie algorytmu posiadającego ciekawe cechy, jak na przykład swego rodzaju pęd w kierunku coraz lepszych rozwiązań. Istnieją także dalsze sposoby na poprawienie działania algorytmu, jakich przykładem jest zastąpienie ewolucji rand-to-rand ewolucją rand-to-best.

2.2 NL-SHADE-RSP

Ten algorytm czerpie z rozwiązań różnych algorytmów opartych na DE, takich jak algorytmy JADE, SHADE, AGSK, IMODE oraz L-SHADE. Najważniejsze elementy tych algorytmów składające się na algorytm ewolucji różnicowej NL-SHADE-RSP to:

- używanie najlepszych osobników przy wyznaczaniu nowych (inspirowane algorytmemm JADE);
- dostosowywanie parametrów mutacji i krzyżowania w czasie (inspirowane algorytmemm L-SHADE);
- zmniejszanie nielinowo ilości osobników dla późniejszych epok (inspirowane algorytmemm AGSK);
- wybieranie odpowiedniego zapisu z historii na podstawie prawdopodobieństwa jego przydatności, i ponowna weryfikacja prawdopodobieństwa co krok (inspirowane algorytmem IMODE).

Po zrozumieniu działania tych rozwiązań algorytm staje się dość prosty i podobny do DE.

3 Opis rozwiązania

Implementacja została wykonana w języku Python wykorzystując popularne biblioteki do pracy na danych, takie jak `numpy`. Do przeprowadzenia badań potrzebne były trzy różne implementacje algorytmu ewolucyjnego. Implementacje te mają wspólny interfejs, pozwalający na proste modularne łączenie oraz przenoszenie różnych cech algorytmów.

Wszystkie implementacje są oparte na klasie bazowej `BaseAgent`, która definiuje podstawowy interfejs i wspólne metody dla algorytmów ewolucyjnych. Klasa ta korzysta z abstrakcyjnych metod, które muszą być zaimplementowane w klasach pochodnych, zapewniając elastyczność i łatwość rozszerzania funkcjonalności.

Klasa `BaseAgent` zawiera następujące kluczowe elementy:

1. **Inicjalizacja**: Konstruktor klasy inicjalizuje obiekt z zadanym celem optymalizacji (`objective`), konfiguracją (`config`) oraz opcjonalną populacją początkową. Jeśli populacja nie jest dostarczona, generowana jest nowa populacja przy użyciu metody `_init_population`.
2. **Inicjalizacja populacji**: Metoda `_init_population` generuje początkową populację przy użyciu rozkładu normalnego o zadanych parametrach średniej (`mu`) i wariancji (`sigma`). Następnie populacja jest sortowana według wartości funkcji celu.

3. **Ocena populacji**: Metoda `_eval` ocenia populację, wyliczając wartość funkcji celu dla każdego osobnika.

4. **Historia populacji**: Klasa przechowuje historię ewolucji populacji, umożliwiając śledzenie zmian w populacji na przestrzeni czasu. Metody `get_history_means`, `dump_history_to_file` i `load_history_from_file` umożliwiają zarządzanie i analizę historii.

5. **Metody abstrakcyjne**: Klasa `BaseAgent` definiuje trzy metody abstrakcyjne: `_init_state`, `_make_step` oraz `run`, które muszą być zaimplementowane w klasach pochodnych. Te metody określają specyficzne dla danego algorytmu operacje, takie jak inicjalizacja stanu, wykonywanie pojedynczego kroku ewolucji oraz uruchamianie algorytmu.

3.1 Implementacja DES

Algorytm DES jest dość prostym algorytmem, którego implementacja była konceptualnie łatwa. Używa on dobrze znanych i sprecyzowanych operacji. Stanowi on dobry startowy algorytm do zrozumienia działania całej rodziny algorytmów.

3.1.1 Inicjalizacja

Na początku klasa `DES` dziedziczy po klasie `BaseAgent` i inicjalizuje swoje parametry:

- `max_n_epochs` - maksymalna liczba iteracji.
- `c` - współczynnik zmiany delty.
- `f` - współczynnik skalowania.
- `d` - mała delta, używana do skalowania losowego szumu.
- `h` - rozmiar okna historii.
- `mu` - liczba najlepszych osobników do rozważenia.
- `e` - intensywność szumu.
- `delta` - dynamiczna wielkość zmian wprowadzanych do populacji.

3.1.2 Inicjalizacja stanu

Metoda `_init_state` tworzy początkowy stan agenta z populacją i deltą ustawioną na wektor wartości początkowej.

3.1.3 Krok algorytmu

Metoda `_make_step` w algorytmie DES wykonuje pojedynczy krok optymalizacji, obejmujący następujące etapy:

1. **Obliczanie średniej populacji**: Na początku metoda oblicza średnią wszystkich osobników w populacji. Średnia ta jest wykorzystywana do oceny ogólnego postępu algorytmu.

2. **Sortowanie populacji**: Następnie populacja jest sortowana na podstawie wartości funkcji celu. Osobniki są uporządkowane od najlepszego do najgorszego w kontekście minimalizacji funkcji celu.

3. **Wyznaczanie najlepszej średniej populacji**: Metoda wyznacza średnią najlepszych osobników w populacji (w liczbie `mu`), co pozwala na uwzględnienie tylko najbardziej obiecujących rozwiązań w kolejnych krokach.

4. **Aktualizacja delty**: Nowa wartość delty jest obliczana jako kombinacja poprzedniej delty i różnicy między najlepszą średnią a obecną średnią populacji. Współczynnik `c` kontroluje tempo zmian delty.

5. **Generowanie nowej populacji**: Dla każdego osobnika w populacji losowane są dwa osobniki z poprzednich stanów historii algorytmu. Nowy osobnik jest generowany jako suma najlepszego osobnika, różnicy losowanych osobników oraz losowego szumu skalowanego deltą i małą deltą (`d`).

6. **Dodawanie szumu**: Do nowego osobnika dodawany jest dodatkowy szum o intensywności `e`, co pomaga w eksploracji przestrzeni poszukiwań.

3.2 Implementacja NL-SHADE-RSP

Algorytm NL-SHADE-RSP jest o wiele bardziej rozbudowany niż DES, stąd jego implementacja jest także bardziej złożona. Dla ułatwienia rozumienia wykonywanego kroku algorytmu, w implementacji pozostawiono komentarze oznaczające które linie pseudokodu oryginalnego artykułu są implementowane.

3.2.1 Inicjalizacja

Klasa `NLSHADERSP` dziedziczy po `BaseAgent` i inicjalizuje swoje parametry:

- `H` - rozmiar okna historii.
- `NPmin` - minimalna wielkość populacji.
- `NPmax` - maksymalna wielkość populacji.
- `max_iters` - maksymalna liczba iteracji.
- `best_part` - część najlepszych osobników w populacji.
- `memory_F` - pamięć wartości współczynnika skalowania.
- `memory_Cr` - pamięć wartości współczynnika krzyżowania.
- `archive_probability` - prawdopodobieństwo wykorzystania archiwum.
- `archive_size` - rozmiar archiwum.

3.2.2 Inicjalizacja stanu

Metoda `_init_state` tworzy początkowy stan agenta, w tym archiwum z częściowo zainicjowaną populacją, pamięci współczynników `M_F` i `M_Cr`.

3.2.3 Krok algorytmu

Metoda `_make_step` w algorytmie NL-SHADE-RSP wykonuje pojedynczy krok optymalizacji, obejmujący następujące etapy:

1. **Sortowanie populacji**: Populacja jest sortowana na podstawie wartości funkcji celu. Sortowanie pozwala na łatwe wybranie najlepszych osobników i pbest do dalszych operacji.
2. **Generowanie realizacji współczynników**: Dla każdego osobnika w populacji losowane są współczynniki `F` i `Cr` z rozkładów opartych na pamięci. Wartości te są używane w operacjach mutacji i krzyżowania.
3. **Mutacja różnicowa**: Dla każdego osobnika wykonywana jest operacja mutacji różnicowej. Wybierane są cztery różne osobniki: aktualny, `pbest`, oraz dwa losowe. Nowy wektor jest generowany jako kombinacja tych osobników z uwzględnieniem współczynnika `F`.
4. **Krzyżowanie**: Wykonywane jest krzyżowanie binarne lub eksponencjalne, w zależności od wartości losowej. Krzyżowanie łączy nowy wektor z aktualnym osobnikiem na podstawie współczynnika `Cr`.
5. **Ocena nowego osobnika**: Nowy osobnik jest oceniany przy użyciu funkcji celu. Jeśli nowy osobnik jest lepszy od aktualnego, zastępuje go w populacji, a aktualny osobnik jest dodawany do archiwum.
6. **Aktualizacja archiwum**: Jeśli archiwum osiągnie maksymalny rozmiar, losowo zastępowany jest jeden z istniejących osobników. Archiwum jest używane do zapewnienia różnorodności populacji.
7. **Dostosowanie rozmiaru populacji**: Rozmiar populacji jest dynamicznie dostosowywany na podstawie liczby iteracji, zapewniając lepszą eksplorację na wczesnym etapie i eksploatację na późniejszym.
8. **Aktualizacja pamięci współczynników**: Pamięci współczynników `M_F` i `M_Cr` są aktualizowane na podstawie sukcesów mutacji i krzyżowania, co pozwala na adaptacyjne dostosowywanie parametrów algorytmu.

3.3 Implementacja Hybrydyzacji

3.3.1 Pierwotne założenia

W dokumentacji wstępnej zostały stwierdzone, że aby skutecznie dokonać hybrydyzacji obu algorytmów, zostaną połączone rozwiązania charakterystyczne dla obu z nich. Wynikowy algorytm powinien:

- zachowywać pęd w kierunku najlepszych rozwiązań;
- zmniejszać populację wraz z czasem działania (nieliniowo, jak w NL-SHADE-RSP);
- dostosowywać parametry mutacji (być może bez krzyżowania, aby nie modyfikować aż nadto DES);
- dokonywać wyboru z okna historii na podstawie wyliczonych prawdopodobieństw przydatności.

Aby uzyskać opisywany rezultat bazowano na algorytmie NL-SHADE-RSP i dokonywano zmian jego fragmentów w taki sposób, aby odzwierciedlały one część założeń algorytmu DES. Do zmian należało przede wszystkim:

- wyliczanie średniego osobnika i w oparciu o niego generowanie nowych rozwiązań
- zachowywanie pędu populacji w kierunku lepszych rozwiązań
- usunięcie krzyżowania i powiązanych z nim parametrów

W efekcie powstał algorytm, który posiada specyficzne elementy algorytmu NL-SHADE-RSP jakimi jest losowanie i dostosowywanie parametrów mutacji, używanie archiwum najlepszych osobników oraz malejąca nieliniowo populacja, ale jest trochę wydajniejszy obliczeniowo ze względu na brak krzyżowania, a do tego zachowujący pęd w kierunku lepszych rozwiązań.

3.3.2 Inicjalizacja

Klasa `Hybridization` dziedziczy po `BaseAgent` i inicjalizuje swoje parametry:

- `H` - rozmiar okna historii
- `NPmin` - minimalna wielkość populacji
- `NPmax` - maksymalna wielkość populacji
- `max_iters` - maksymalna liczba iteracji
- `best_part` - część najlepszych osobników w populacji
- `memory_F` - pamięć wartości współczynnika skalowania
- `archive_probability` - prawdopodobieństwo wykorzystania archiwum.
- `archive_size` - rozmiar archiwum.
- `c` - współczynnik zmiany delty
- `d` - mała delta, używana do skalowania losowego szumu
- `e` - intensywność szumu

3.3.3 Inicjalizacja stanu

Metoda `_init_state` tworzy początkowy stan agenta, w tym archiwum z częściowo zainicjowaną populacją, oraz pamięć współczynnika `M_F`.

3.3.4 Krok algorytmu

Metoda `_make_step` w algorytmie zhybrydyzowanym wykonuje pojedynczy krok optymalizacji, obejmujący następujące etapy:

1. **Sortowanie populacji**: Populacja jest sortowana na podstawie wartości funkcji celu. Sortowanie pozwala na łatwe wybranie najlepszych osobników i pbest do dalszych operacji.
2. **Generowanie współczynników mutacji**: Dla każdego osobnika w populacji losowany jest współczynnik `F` z rozkładu opartego na pamięci.
3. **Wyznaczenie średniej z najlepszych osobników w populacji**: Metoda wyznacza średnią najlepszych osobników w populacji w liczbie `best_part` przemnożonej prze obecną wielkość populacji (warto zauważyć, że w tym przypadku nie można użyć parametru `mu` jak w przypadku algorytmu DES, ponieważ wielkość populacji zmienia się w czasie).

4. **Aktualizacja delty**: Nowa wartość delty jest obliczana jako kombinacja poprzedniej delty i różnicy między najlepszą średnią a obecną średnią populacji. Współczynnik `c` kontroluje tempo zmian delty.

5. **Mutacja różnicowa**: Dla każdego osobnika wykonywana jest operacja mutacji różnicowej. Wybierane są dwa losowe osobniki o indeksie różnym od indeksu aktualnie rozpatrywanego osobnika. Nowy wektor jest generowany jako suma średniej z najlepszych osobników populacji, różnicy dwóch wylosowanych osobników z uwzględnieniem przeskalowanej o współczynnik `F`, a także szumu skalowanego deltą i małą deltą (`d`).

6. **Ocena nowego osobnika**: Nowy osobnik jest oceniany przy użyciu funkcji celu. Jeśli nowy osobnik jest lepszy od aktualnego, zastępuje go w populacji, a aktualny osobnik jest dodawany do archiwum.

7. **Aktualizacja archiwum**: Jeśli archiwum osiągnie maksymalny rozmiar, losowo zastępowany jest jeden z istniejących osobników. Archiwum jest używane do zapewnienia różnorodności populacji.

8. **Dostosowanie rozmiaru populacji**: Rozmiar populacji jest dynamicznie dostosowywany na podstawie liczby iteracji, zapewniając lepszą eksplorację na wczesnym etapie i eksploatację na późniejszym.

9. **Aktualizacja pamięci współczynnika mutacji**: Pamięć współczynnika `M_F` jest aktualizowana na podstawie sukcesów mutacji.

4 Eksperymenty

4.1 Wizualizacje i dowolne funkcje celu

W ramach badań nad algorytmami optymalizacji różnicowej, wykorzystano dwie funkcje celu: `quad` oraz `almost_twin_peaks`. Funkcje te zostały zaprojektowane w celu przetestowania wydajności i skuteczności algorytmów w różnych scenariuszach optymalizacyjnych.

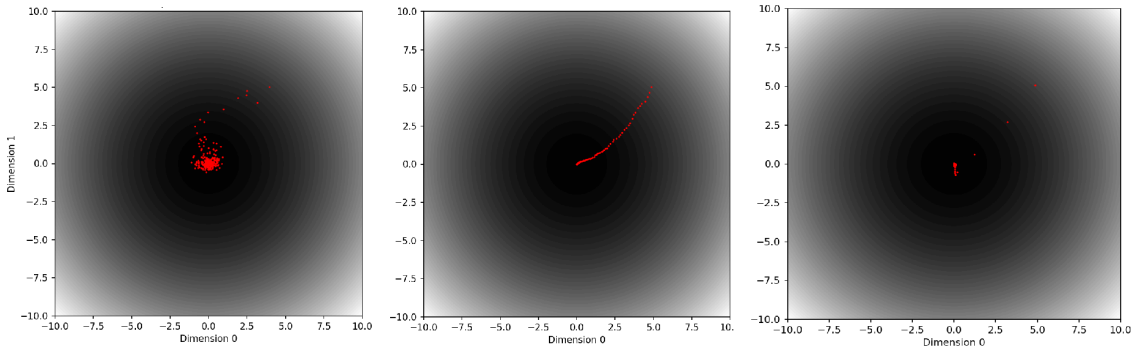
4.1.1 Funkcja `quad`

Funkcja `quad` jest klasycznym przykładem funkcji kwadratowej, często wykorzystywanej w testach algorytmów optymalizacyjnych ze względu na swoją prostotę i dobrze znane właściwości.

$$\text{quad}(x) = \sum_{i=1}^{n} x_i^2$$

gdzie $x = (x_1, x_2, \ldots, x_n)$ jest osobnikiem. Funkcja ta oblicza sumę kwadratów elementów wektora x. Funkcja ma globalne minimum w punkcie $x = 0$, gdzie osiąga wartość 0. Jest ona bardzo prostą funkcją, mającą na celu sprawdzić podstawowe cechy algorytmów.

Wyniki są zaprezentowane na rysunku1. Czerwone punkty to średia populacji w danym kroku. Problem rozwiązywany jest dla pięciu wymiarów, a tutaj zaprezentowana jest płaszczyzna będąca przekrojem pierwszego i drugiego wymiaru. Każdy algorytm poza hybridization został zainicjowany parametrami autorów, a w wypadku hybridizaton zostały wybrane najlepsze parametry startowe. Widoczne jest że każdy z algorytmów zachowuje się rozsądnie na prostym przykładzie, zmierzając do minimum globalnego



Rysunek 1: Wyniki algorytmów na funkcji quad. Od lewej: DES, NL-SHADE-RSP, Hybridization

4.1.2 Funkcja `almost_twin_peaks`

Funkcja `almost_twin_peaks` jest bardziej złożoną funkcją testową, charakteryzującą się dwoma głównymi szczytami (maksimami) o różnych wysokościach i szerokościach. Funkcja ta jest zdefiniowana jako suma dwóch funkcji Gaussa, z których każda reprezentuje jeden ze szczytów.

$$\text{almost\_twin\_peaks}(x) = -20\exp\left(-\frac{\sum(x+2)^2}{3.0}\right) - \exp\left(-\frac{\sum(x-2)^2}{10.0}\right)$$

gdzie $x = (x_1, x_2, \ldots, x_n)$ jest wektorem zmiennych decyzyjnych. Funkcja ta oblicza sumę dwóch wartości:

- **Peak 1:** $-20\exp\left(-\frac{\sum(x+2)^2}{3.0}\right)$ - Reprezentuje wysoki, wąski szczyt położony w punkcie $x = -2$. Wartość maksymalna tego szczytu wynosi -20.
- **Peak 2:** $-1\exp\left(-\frac{\sum(x-2)^2}{10.0}\right)$ - Reprezentuje niski, szeroki szczyt położony w punkcie $x = 2$. Wartość maksymalna tego szczytu wynosi -1.

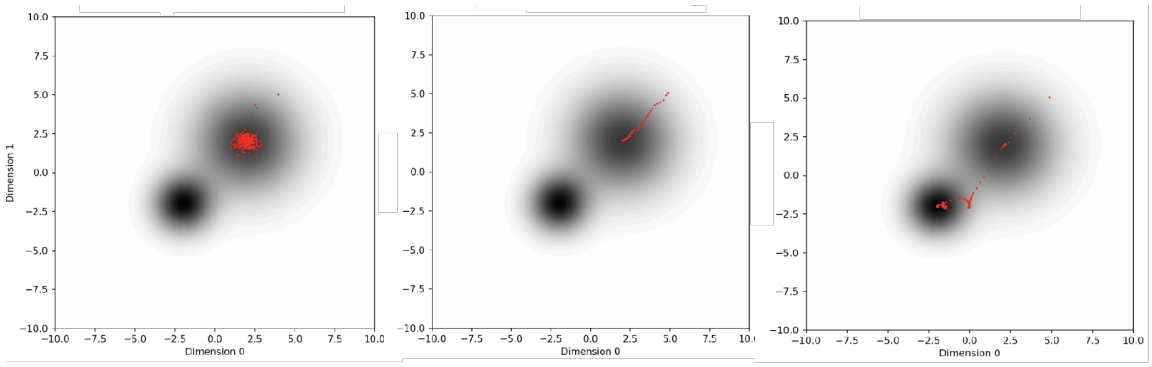
Funkcja `almost_twin_peaks` została zaprojektowana w celu testowania zdolności algorytmów optymalizacyjnych do eksploracji przestrzeni poszukiwań i znajdowania różnych lokalnych maksimów. Ze względu na obecność wielu szczytów, głównym wyzwaniem jest ominięcie lokalnego minimum i dotarcie do globalnego

Ponownie algorytmy zostały zainicjowane jak w wypadku funkcji quad. Wyniki przedstawiono na obrazku 2. Widzimy, że jako jedyny do optimum globalnego doatarł algorytm hybridization, trochę niespoidziewanie zakręcając po pokonaniu minimum lokalnego. NL-SHADE-RSP ze względu na swoją konstrukcję powinien poradzić sobie z lokalnym minimum po drodze, jednak z parametrami wskazanymi przez autorów utyka. Po zmianach parametrów początkowych, drogę znajduje także NL-SHADE-RSP.

4.2 CEC 2022

Eksperymenty zostały przeprowadzone na zbiorze funkcji CEC 2022 [3]. Zbiór ten pochodzi z konferencji *Congress of evolutionary computation*, a jego wcześniejsze wersje były używane w artykułach przedstawiających DES i NL-SHADE-RSP.

W tabeli1 przedstawiono wyniki optymalizacji na benchmarku CEC dla trzech modeli: DES, NL-SHADE-RSP oraz ich hybrydyzacji. Ze względu na długość obliczeń, zostały przeprowadzone dla dwóch wymiarów problemu. W przypadku funkcji cec-7 oraz cec-8 nie ma możliwości uruchomienia



Rysunek 2: Wyniki algorytmów na funkcji almost-twin-peaks. Od lewej: DES, NL-SHADE-RSP, Hybridization

benchmarku dla innej ilości wymiarów niż 10 lub 20, stąd jako jedyne prezentują wyniki dla dziesięciu wymiarów.

Funkcja celu	**DES**	**NL-SHADE-RSP**	**Hybrydyzacja**
cec-1	300.00008053185184	300.00001806121116	300.0000034619176
cec-2	400.0257699569012	400.0000049037666	400.0000046890257
cec-3	677.5750526866956	677.5750526866956	677.5750526866956
cec-4	812.9344324320411	812.9344324320411	812.9344324320411
cec-5	964.3280050908908	900.0000089874949	900.0002845643668
cec-6	1800.0001553734842	1800.0001150622827	1800.0002876331273
cec-7 (10 wymiarów)	2129.368930069756	2000.0000687574163	2130.7807940256284
cec-8 (10 wymiarów)	2443.0457959481746	2437.206992707528	2487.605840240031
cec-9	2639.416734343873	2600.0	2600.0000000006607
cec-10	2600.000000000968	2600.0	2600.00000000008
cec-11	2600.000048336511	2600.0000601806287	2600.0001131275058
cec-12	3111.155775991618	3165.4536443462457	3165.4536443462457

Tabela 1: Wyniki optymalizacji na benchmarku CEC dla trzech modeli

Zmierzono także długość wykonywania 5000 kroków dla wszystkich algorytmów ponieważ zauważono, że długość wykonywania programu znacznie się różni w zależności od algorytmu. W tabeli 2 przedstawiono wyniki pomiarów.

DES	**NL-SHADE-RSP**	**Hybrydyzacja**
3.28	27.11	16.09

Tabela 2: Czas wykonywania 5000 kroków algorytmu (w sekundach)

5 Podsumowanie i wnioski

Na podstawie wyników eksperymentów można wysnuć następujące wnioski:

- zaimplementowany zhybrydyzowany algorytm uzyskuje wyniki ogólnie porównywalne z pozostałymi algorytmami
- w przypadku każdej funkcji poza funkcjami 7 i 8 zaimplementowany zhybrydyzowany algorytm uzyskuje niemal takie same wartości jak NL-SHADE-RSP
- zaimplementowany jest o około 60% szybszy od algorytmu NL-SHADE-RSP
- algorytm DES uzyskuje lepsze rezultaty od pozostałych dwóch algorytmów w przypadku funkcji 12, natomiast gorsze w przypadku 5 i 9

Literatura

[1] https://github.com/GMC-DRL/psc4MetaBBO/blob/main/Classic%20BBO/Differential%20Evolution/NL-SHADE-RSP/NL-SHADE-RSP%20algorithm%20with%20adaptive%20archive%20and%20selective%20pressure%20for%20CEC%202021%20numerical%20optimization.pdf - NL-SHADE-RSP

[2] https://ieeexplore.ieee.org/abstract/document/7969529 - DES

[3] https://github.com/P-N-Suganthan/2022-SO-BO/tree/main - CEC