Algorytm inspirowany polem walki - połączenie algorytmów numerycznych z ideą roju.

Mgr inż Jan Baumgart Mgr inż Belco Sangho

[1] *Uniwersytet Kazimierza Wielkiego, Instytut Informatyki, jan.baumgart@ukw.edu.pl*
[2] *Uniwersytet Kazimierza Wielkiego, Instytut Informatyki, belco.sangho@ukw.edu.pl*

Streszczenie: *Artykuł przedstawia przygotowany algorytm na bazie połączenia idei znanych metod numerycznych z metodami opartymi na idei roju. Algorytm został przygotowany z inspiracji polem walki podczas którego w równych odstępach żołnierze przeczesują siły wroga z różnymi prędkościami zależnie od posiadanego orężu a następnie ograniczają zakres pola bitwy. Zaproponowane rozwiązanie wywodzi się właśnie ze zbliżonych założeń. Głównym założeniem pracy było przedstawienie potencjalnego zysku z połączenia metod optymalizacji oraz porównanie metody mieszanej z metodami bazującymi na idei roju pod względem prędkości działania oraz skuteczności odnajdowania optimum globalnego.Algorytm został porównany z dwoma algorytmami metaheurystycznymi pod kątem dokładności odnalezionych rozwiązań oraz prędkości. Zgodnie z wynikami eksperymentów posiada wydajność podobną w porównaniu z innymi algorytmami oraz daje zadowalające efekty w wykorzystaniu.*

Słowa kluczowe: *algorytm optymalizacyjny; inspiracja polem walki; rozwiązanie; optymalizacja; rzeczywiste problemy optymalizacji; optymalizacja funkcji; algorytm numeryczny;*

Abstract: *The article presents prepared algorithm based on the combination of the ideas of known numerical methods with methods based on the idea of a swarm. The algorithm was prepared inspired by the battlefield, during which, at equal intervals, soldiers scour enemy forces at different speeds depending on the weapon they have, and then limit the scope of the battlefield. The proposed solution is based on similar assumptions. The main assumption of the work was to present the potential profit from the combination of optimization methods and to compare the mixed method with methods based on the idea of a swarm in terms of operating speed and the effectiveness of finding the global optimum. The algorithm was compared with two metaheuristic algorithms in terms of the accuracy of the solutions found and speed. According to the results of the experiments, it has a similar performance compared to other algorithms and gives satisfactory results in use.*

Keywords: *optimization algorithm; battlefield inspired inspiration; solution; optimization; real optimization problems; function optimization; numerical algorithm;*

1. Wprowadzenie

Algorytmy Rojowe stają się coraz bardziej skuteczne w rozwiązywaniu problemów optymalizacji numerycznej. Rzeczywiste problemy niestety czasami wiążą się prawdopodobieństwem wystąpienia wielu globalnych i lokalnych optimum. Jeśli do zadania optymalizacji zastosuje się klasyczną metodę optymalizacji numerycznej punkt po punkcie, metoda klasyczna musi wielokrotnie próbować znaleźć za każdym razem inne optymalne rozwiązanie, co zajmuje dużo czasu i pracy. Z drugiej strony wykorzystywanie inspirowanych naturą algorytmów metaheurystycznych do odnalezienia rozwiązania jest często kuszące ponieważ mogą z bardzo dużym prawdopodobieństwem osiągnąć globalne optima lecz często nie dają pełnego obrazu możliwych rozwiązań, co daje efekt niepewności względem odnalezionego wyniku.[2] W tym artykule przedstawiam algorytm do optymalizacji funkcji, opartą na najlepszych praktykach z obu światów, klasycznej metody optymalizacji punkt po punkcie w połączeniu z umiejętnością przekazania najlepszych wyników do kolejnych generacji.

Algorytm ten został porównany z najlepszymi przykładami algorytmów optymalizacji, uznanych w dziedzinie takimi jak Particle Swarm Optimization oraz Artifical Bee Colony.

Pierwszy z nich to algorytm oparty na inteligencji roju zaproponowany przez Kennedy'ego i Eberharta [4], inspirowany zachowaniem społecznym stada ptaków. Algorytm ten zyskał uznanie w dziedzinie, zarówno za dobre osiągnięcia w przypadku funkcji testowych jak i za wielokrotną implementację przy problemach rzeczywistych. Główną ideą algorytmu jest każda cząstka przemieszczająca się do nowej pozycji w oparciu o aktualizację jej prędkości i pozycji, przy czym prędkość dotyczy jej najlepszej poznanej pozycji i najlepszej pozycji zespołu cząstek.

Drugi zaś został opracowany przez Karabogę w 2005 roku [3] i został zainspirowany inteligentnym zachowaniem żerowania pszczół miodnych. Idea algorytmu skrywa się za trzema zasadniczymi elementami: pracujących i bezrobotnych pszczół żerujących oraz źródeł pożywienia. Zatrudnione i bezrobotne pszczoły żerujące, szukają najlepszych pożywienia, co jest trzecim komponentem w pobliżu ich ula. Model definiuje również dwa wiodące sposoby zachowania, które są niezbędne do samoorganizacji i inteligencji zbiorowej: rekrutacja zbieraczy do bogatych źródeł pożywienia skutkująca pozytywną informacją zwrotną oraz porzucanie ubogich źródeł przez zbieraczy powodujące negatywne opinie.

Dalsza część artykułu jest zorganizowana w następujący sposób. Kolejna sekcja zawiera opis proponowanego rozwiązania, dalej zaprezentowano kolejno konfiguracje algorytmów, opis przygotowania eksperymentu, przedstawienie wyników eksperymentalnych przeprowadzonych w celu oceny proponowanego algorytmu oraz wnioski i tematy do dalszych badań.

2. Funkcje testujące

Algorytmy zostały przetestowane z wykorzystaniem przygotowanego samemu programu do benchmarku algorytmów na funkcjach matematycznych przygotowanym właśnie do testowania tych algorytmów. [1] Wykorzystanie przygotowanej aplikacji pozwala na szybsze przeprowadzenie badań w bardziej skontrolowanym środowisku. Aplikacja pozwala na przedstawienie wyników w formie wykresu oraz przygotowanie tabel z wynikami oraz mierzy prędkość wykonania kodu. Algorytmy zostały przetestowane na 5 różnych od siebie pod względem charakterystyki funkach matematycznych. Duża różnorodność funkcji pozwoliła na przetestowanie algorytmów w każdych potencjalnych warunkach. Funkcje wybrane to Michalewicz, HolderTable, Booth, CrossInTray oraz Shwefel.

2.1 Michalewicz

Reprezentacja matematyczna funkcji

$$f(\mathbf{x}) = -\sum_{i=1}^{d} sin(x_i) sin\left(\frac{ix_i^2}{\pi}\right)^{2m}$$

Minimum Globalne poszukiwane dla domyślnego zakresu

$$f(2.20, 1.57) \approx -1.8013$$

2.2 HolderTable

Reprezentacja matematyczna funkcji

$$f(x,y) = -|sin(x)cos(y)exp(|1 - \frac{\sqrt{x^2+y^2}}{\pi}|)|$$

Minimum Globalne poszukiwane dla domyślnego zakresu

$$f(pm8.05502, pm9.66459)approx - 19.2085$$

2.3 Booth

Reprezentacja matematyczna funkcji

$$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$

Minimum Globalne poszukiwane dla domyślnego zakresu

$$f(1,3) = 0$$

2.4 CrossInTray

Reprezentacja matematyczna funkcji

$$f(x,y) = -0.0001(|sin(x)sin(y)exp(|100 - \frac{\sqrt{x^2+y^2}}{\pi}|)|+1)^{0.1}$$

Minimum Globalne poszukiwane dla domyślnego zakresu

$$f(x,y) \approx -2.06261218, at$$

$x \pm 1.349406685353340, and y \pm 1.349406608602084'$

2.5 Schwefel

Reprezentacja matematyczna funkcji

$$f(x_1 \cdots x_n) = \sum_{i=1}^{n}(-x_i sin(\sqrt{|x_i|})) + \alpha \cdot n$$

Minimum Globalne poszukiwane dla domyślnego zakresu

$$f(420.968746, 420.968746, \cdots, 420.968746) = 0$$

3. Opis algorytmu

Zaproponowany algorytm bazuje na kilku etapach przeszukiwania oraz głównej funkcji sterującej. W początkowym etapie pracy następuje podział całego zakresu granic przeszukiwania tak aby ocenić od jakiej dokładności należy zacząć przeszukiwanie przestrzeni aby zawsze ograniczyć się do 10 przejść w pierwszym etapie. I tak dla zakresu od -5 do 5 przeszukiwanie w pierwszej fazie będzie następować co krok o wartości 1 a dla zakresu od -50 do 50 krok o wartości 10. Następnie budujemy listę precyzji zaczynając właśnie od określonej wartości a kończąc na wartości ustalonej jako zmienna algorytmu zwaną PrecisionOfOutput. Lista oczywiście jest posortowana w sposób malejący.

W tym momencie rozpoczynamy proces przeszukiwania przestrzeni z określoną dokładnością, podczas pierwszej iteracji przeszukiwania zaczynamy od szerokości kroku określonego poprzez funkcję inicializującą opisaną powyżej oraz dla całej zadanej przestrzeni rozwiązań. Po pierwszej iteracji budujemy listę najlepszych rozwiązań oraz ich współrzędnych, lista zawiera wszystkie unikalne współrzędne zgodnie z opisem w dalszej części.

W następnych iteracjach zasada działania jest taka sama dla każdego przebiegu. Dla wszystkich unikalnych współrzędnych odnalezionych w poprzednim przebiegu przeszukujemy z dokładnością zwiększoną o jeden stopień lecz tym razem tylko w zakresie określonym w taki sposób.

Dodatkowo sprawdzamy czy aktualnie obrane ograniczenia dla współrzędnych nie wychodzą poza nadane przez użytkownika ograniczenia globalne, jeżeli tak to ograniczamy dany zakres do wartości nadanej przez użytkownika. Na tym etapie przechodząc przez każdą wartość współrzędnych obliczamy wynik dla zadanych par. Należy również zwrócić uwagę na różnice między kodem którego zadaniem jest odnalezienie pojedynczego rozwiązania a kodem rozwiązania którego zadaniem jest odnalezienie możliwie wszystkich rozwiązań.

Algorytm którego zadaniem jest odnalezienie każdej poprawnej odpowiedzi dla każdej pary współrzędnych X i Y dopisze do listy każdą napotka wartość i współrzędne. Co pozwala na obliczenie wartości z każdych współrzędnych i następnie przefiltrowanie najlepszych na podstawie sortowania wyników oraz przekazania tylko unikalnych wpisów, tych z najlepszym wynikiem. Podczas gdy algorytm którego celem jest osiągniecie pojedynczego wyniku z wysoką prędkością będzie pamiętał globalnie osiągnięte wyniki przez co zawsze zapamięta tylko najlepszą wartość. Co pozwala na znaczne ograniczenie sprawdzanych wyników dla każdego przebiegu , niestety kosztem ilości proponowanych rozwiązań.

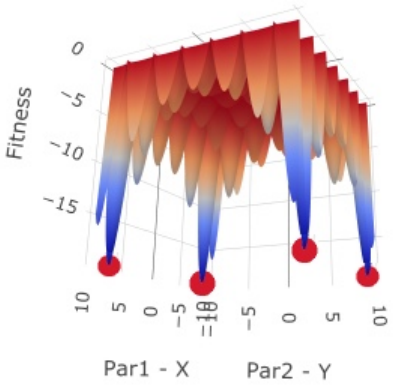
Po przejściu przez wszystkie zadane współrzędne przechodzimy do przedostatniej funkcji czyli do filtrowania wyników w następujący sposób. W pierwszym filtrujemy listę tylko do unikalnych wpisów. Na kolejnym etapie sortujemy pozostałe osiągnięte wyniki od najlepszych do najgorszych. Następnie pobieramy osiągniętą wartość na pierwszym rzędzie i filtrujemy wyniki z dokładnością do 8 znaku po przecinku aby ponownie aby zachować tylko te współrzędne z najlepszym wynikiem lub bardzo zbliżonym.

Algorytm przechodzi przez opisane kroki dla wszystkich obliczonych precyzji, od pierwszej obliczonej na podstawie zakresu aż po ostatnią zadaną przez użytkownika. W każdym testowanym przypadku w ostatniej iteracji odpowiednio zostają odnalezione wszystkie możliwe wyniki lub te które wpisały się w uproszony zakres.

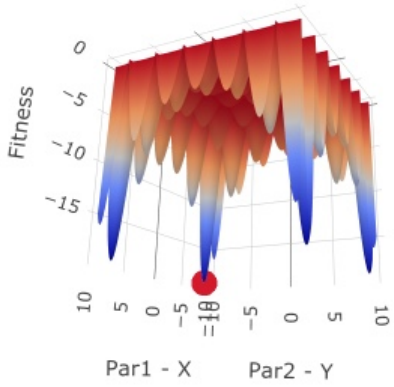
4. Wyniki badań

Przeprowadzenie badań polegało na uruchomieniu poszczególnych algorytmów z odpowiednią parametryzacją dla poszczególnych funkcji w 20 przebiegach dla każdej konfiguracji. Następnie uśredniono osiągnięte wyniki i przygotowano wykresy oraz tabele wyników.

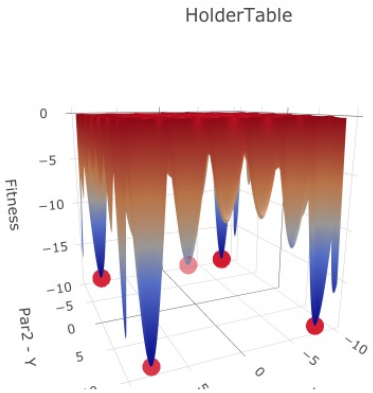
Dla zobrazowania możliwości zaproponowanego algorytmu poniżej przedstawiono przykład osiągniętych wykresów dla funkcji Holder Table.



Rysunek 1: Przedstawia wyniki zaproponowanego algorytmu przygotowanego pod kątem odnalezienia wszystkich możliwych wyników



Rysunek 2: Przedstawia wyniki algorytmu skonfigurowanego pod kątem szybkości działania.

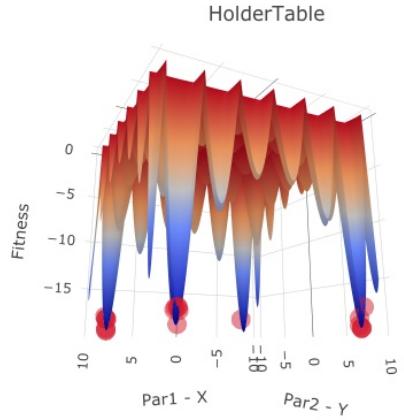


Rysunek 3: Przedstawia wyniki osiągnięte przez algorytm Artifical Bee Colony

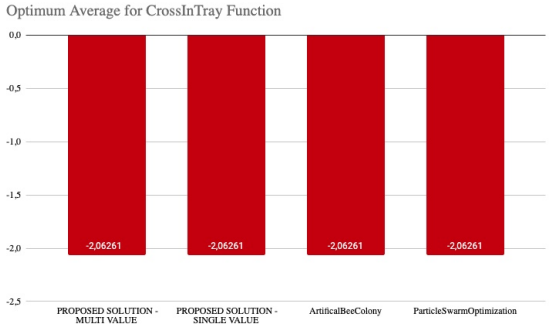
Jak widać na załączonych powyżej grafikach zaproponowany algorytm posiada delikatną przewagę nad pozostałymi algorytmami pod kątem dokładności rozwiązań w przypadku poszukiwań pod kątem szybkości odnalezienia rozwiązania jak i zarówno w przypadku próby odnalezienia wszystkich możliwych rozwiązań. Algorytm nie ma tendencji do osiadania w optimach lokalnych.

4.1 Porównanie wydajności pod kątem dokładności wyników.

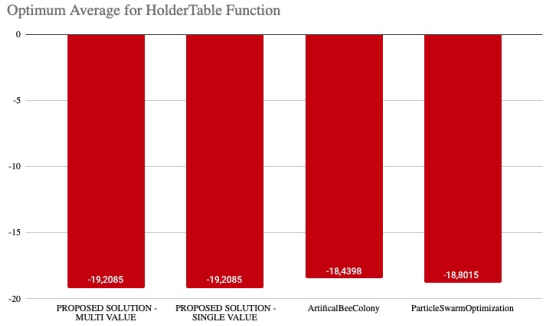
Poniżej przedstawiono wykresy odnalezionych rozwiązań, przygotowane z uśrednionych danych z 20 przebiegów dla każdego z algorytmów.



Rysunek 4: Przedstawia wyniki osiągnięte przez algorytm Particle Swarm Optmization

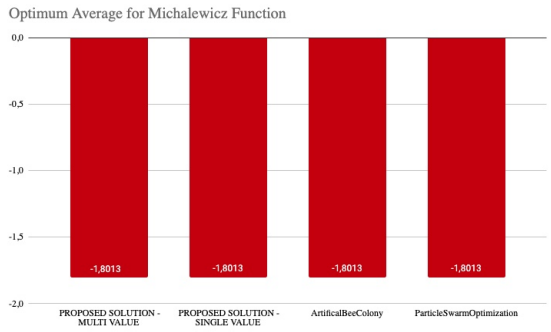


Rysunek 5: Wykres przedstawiający średnie osiągnięte wyniki dla funkcji CrossInTray przez każdy z algorytmów.

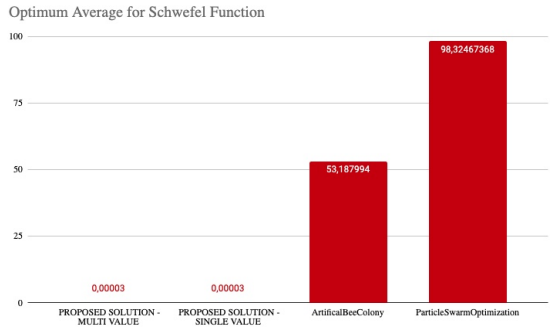


Rysunek 6: Wykres przedstawiający średnie osiągnięte wyniki dla funkcji HolderTable przez każdy z algorytmów.

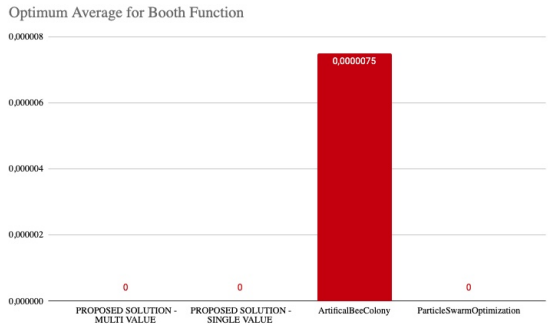
Jak widać zaproponowany algorytm idealnie odnalazł się w większości funkcji testowych. Tylko w jednej funkcji nie udało mu się osiągnąć idealnego rozwiązania. Na wykresie została zaprezentowa-



Rysunek 7: Wykres przedstawiający średnie osiągnięte wyniki dla funkcji Michalewicza przez każdy z algorytmów.



Rysunek 8: Wykres przedstawiający średnie osiągnięte wyniki dla funkcji Schwefel przez każdy z algorytmów.

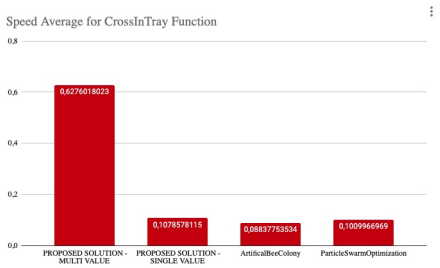


Rysunek 9: Wykres przedstawiający średnie osiągnięte wyniki dla funkcji Booth przez każdy z algorytmów.

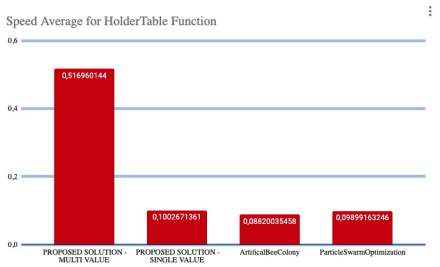
na implementacja przeszukująca wszystkie rozwiązania jak i tylko pojedyncze wyniki. Pomimo pozornego podobieństwa wyników należy zauważyć że podczas gdy algorytm drugi od lewej zawsze odnajdował to samo rozwiązania algorytm przeszukujący całe zakresy odkrywał wszystkie możliwe wyniki a mimo to nie osiadł w optimum lokalnym.

4.2 Porównanie wydajności pod kątem prędkości algorytmów.

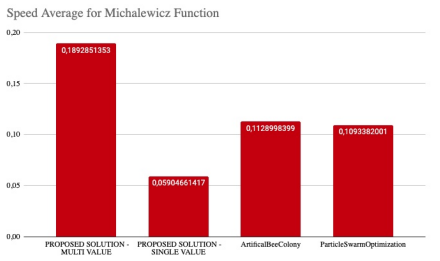
W poniższej sekcji porównano algorytmy pod kątem szybkości wykonania i odnalezienia rozwiązania. Aby rzetelnie oddać prędkości test wykonano 20 razy dla każdego algorytmu a następnie uśredniono wyniki. Wyniki prezentują się następująco:



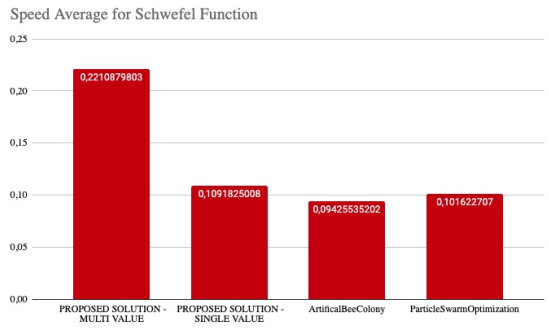
Rysunek 10: Wykres przedstawiający średnią prędkość wykonania dla funkcji CroosInTray przez każdy z algorytmów.



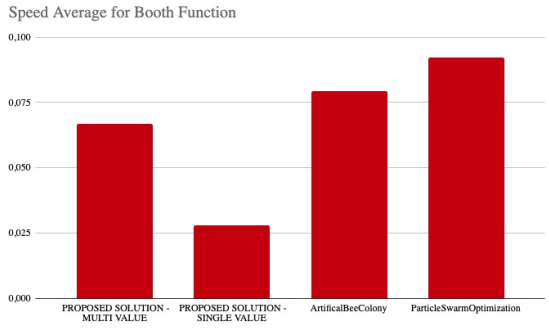
Rysunek 11: Wykres przedstawiający średnią prędkość wykonania dla funkcji HolderTable przez każdy z algorytmów.



Rysunek 12: Wykres przedstawiający średnią prędkość wykonania dla funkcji Michalewicza przez każdy z algorytmów.



Rysunek 13: Wykres przedstawiający średnią prędkość wykonania dla funkcji Schwefel przez każdy z algorytmów.



Rysunek 14: Wykres przedstawiający średnią prędkość wykonania dla funkcji Booth przez każdy z algorytmów.

Jak widać zaproponowany algorytm stwarza konkurencję godną pozostałym algorytmom. Na wykresie został zaprezentowany algorytm przeszukujący wszystkie rozwiązania jak i tylko pojedyncze najlepsze wyniki zgodnie z wcześniejszą deskrypcją. Wersja algorytmu skierowana na odnalezienie wszystkich wyników wykazała się zdecydowanie wolniejsza, jest to jednak koszt związany z wyszukiwaniem czasami nawet 4 różnych rozwiązań. Algorytm ukierunkowany na prędkość działania i odnalezienie pojedynczych globalnych rozwiązań okazał się być bardzo konkurencyjny w przypadku funkcji z małym zakresem przestrzeni rozwiązań jak Michalewicz i Booth w przypadku pozostałych funkcji różnica miedzy osiągniętymi wynikami była znikoma. Należy zwrócić uwagę że nawet dla funkcji z szerokim zakresem takimi jak Schwefel (-500, 500) wykres tylko delikatnie odchylił się na niekorzyść nowo zaproponowanego algorytmu.

5. Wnioski

Analizując wyniki można dostrzec potencjał w wykorzystaniu zaproponowanego algorytmu. Zarówno wersja skierowana na prędkość działania jak i wersja przeszukująca wszystkie możliwe optima odnalazły się w dobrym miejscu na tabeli wyników, pomimo konkurencji uznanej w dziedzinie za przełomowe. W przypadku problemów o pojedynczym optimum globalnym można uznać za zasadne skorzystanie z nowego rozwiązania, ze względu na przewagę w prędkości i mniejszego poziomu skomplikowania ze względu na tylko jeden parametr sterujący. Ponadto algorytm ten zapewnił wyższą dokładność w przypadku wyników uśrednionych niż pozostałe.

Algorytm skierowany na odnalezienie wszystkich wyników niestety nie wykazał się wyjątkową szybkością jak poprzednik, lecz nie takie było jego zadanie. Algorytm za każdym razem odnalazł wszystkie optima globalne w przypadku funkcji zawierających więcej niż 1 rozwiązanie. Tutaj należy zwrócić uwagę że pozostałe algorytmy aby odnaleźć wszystkie rozwiązania należy uruchomić kilkukrotnie podczas gdy algorytm zaproponowany wystarczy tylko raz. Daje to znaczącą przewagę w czasie wykonania zadania pomimo pozornie długiego czasu pracy. Można więc otwarcie założyć ze połączenie klasycznych metod numerycznych z umiejętnością przekazywania najlepszych wyników z grupy inspirowane algorytmami rojowymi daje zadowalające efekty.

Celem na przyszłość jest dalsze ulepszanie algorytmu. I dostosowanie go, aby można było go zastosować w większej liczbie dziedzin i rozwiązywać bardziej zaawansowane problemy optymalizacji.

Literatura

[1] J. Baumgart, *Application of OFN notation in Swarm Algorithms -basic implementation problems in the R environment*, 2019.

[2] J. Czerniak *et al.*, *(2015).* New proposed implementation of ABC method to optimization of water capsule flight, 2015.

[3] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm," *In:Journal of global optimization39*, vol. 3, pp. 459–471, 2007.

[4] J. Kennedy and R. Eberhart, "Particle swarm optimization," *In:Proceedings of ICNN95-international conference on neural networks*, vol. 4, pp. 1942–1948, 1995.