Zeszyty Naukowe WWSI, No 24, Vol. 15 2021, s. 7-34
DOI: 10.26348/znwwsi.24.7

Wykorzystanie algorytmu świetlika dla znajdowania ekstremów globalnych wybranych funkcji testowych

Jan Olszewski[*], Ewa Figielska[**]

Warszawska Wyższa Szkoła Informatyki

Abstrakt

W artykule przedstawiona została jedna z najnowszych metod inteligencji rojowej – algorytm świetlikowy zaproponowany przez Xin-She Yanga w roku 2008. Przeprowadzona została analiza działania algorytmu, zbadany został wpływ zmian wartości jego parametrów na jakość uzyskiwanych rozwiązań przy poszukiwaniu ekstremów globalnych wybranych nieliniowych funkcji jedno i wielomodalnych.

Słowa kluczowe – inteligencja rojowa, algorytm świetlikowy, optymalizacja

[*] E-mail: j_olszewski@ms.wwsi.edu.pl
[**] E-mail: efigielska@wwsi.edu.pl

Zgłoszono do druku 2 lipca 2021 r.

1. Wstęp

Problemy optymalizacyjne towarzyszą ludzkości od zawsze. Na przestrzeni dziejów powstawało wiele ich matematycznych modeli oraz metod rozwiązywania, lecz prawdziwym przełomem było pojawienie się komputerów, które dzięki dużej mocy obliczeniowej, pozwoliły szybko i skutecznie rozwiązywać wiele problemów optymalizacyjnych świata rzeczywistego.

Metody inteligencji rojowej cieszą się w ostatnim czasie dużym zainteresowaniem w dziedzinie rozwiązywania problemów optymalizacyjnych. Naśladują one czynności naturalnych bytów, zwłaszcza zwierząt, znanych z tworzenia skupisk lub stad. Podobnie jak w naturalnych grupach stworzeń, tak w algorytmach inteligencji rojowej występują oddziaływania między poszczególnymi osobnikami. Mogą być to zachowania bazujące na współpracy, rywalizacji lub atrakcyjności. Wynik działań matematycznych symulujących takie zachowania zależy często od wartości parametrów charakteryzujących populację osobników i oddziaływania między nimi. Parametry te bezpośrednio wpływają na całościową pracę algorytmu, a przez odpowiedni dobór ich wartości, możliwe jest dostrojenie algorytmu do pracy z konkretnymi problemami optymalizacyjnymi lub ich całymi klasami.

Wśród najczęściej stosowanych algorytmów inteligencji rojowej można wyróżnić algorytm mrówkowy (ang. *Ant Colony Optimization*) [1; 2], algorytm optymalizacji rojem cząstek (ang. *Particle Swarm Optimization*) [3], algorytm optymalizacji żerowania bakterii (ang. *Bacteria Foraging Optimization*) [4; 5]. Algorytmy te działają w następujący sposób:

- Algorytm mrówkowy inspirowany jest ruchem mrówek, z których każda wyruszając z gniazda w poszukiwaniu pożywienia pozostawia za sobą ślad z feromonów (substancji zapachowych). Gdy mrówce uda się znaleźć pożywienie, zanosi je do gniazda wracając po swoim śladzie i wzmacniając tym samym jego intensywność. Inne mrówki z większym prawdopodobieństwem udadzą się ścieżką mającą silniejszy ślad, z czasem mniej atrakcyjne ślady feromonów wyparowują, aż w końcu prawie wszystkie mrówki podążą najkrótszą drogą do źródła pożywienia. Pojedyncza mrówka reprezentuje potencjalne rozwiązanie problemu, dla którego wartości zmiennych ustalane są, w każdej iteracji algorytmu, z prawdopodobieństwem zależnym

od intensywności feromonów. Intensywności feromonów są to wagi możliwych do osiągnięcia wartości zmiennych wyznaczane na podstawie aktualnych wartości funkcji celu dla rozwiązań reprezentowanych przez wszystkie mrówki.

- Zasada działania algorytmu optymalizacji rojem cząstek opiera się na symulacji zachowania społecznego ptaków. Potencjalne rozwiązanie problemu reprezentowane jest przez pozycję cząstki poruszającej się z pewną prędkością w roju cząstek. Aktualna prędkość cząstki jest funkcją jej poprzedniej prędkości i najlepszej pozycji osiągniętej do tej pory oraz najlepszej pozycji osiągniętej do tej pory przez cały rój. W kolejnych iteracjach algorytmu pozycje cząstek w roju są aktualizowane na podstawie ich prędkości.
- W algorytmie optymalizacji żerowania bakterii potencjalne rozwiązania problemu reprezentowane są przez bakterie, które mogą wykonywać dwa rodzaje ruchów: ruch w losowym kierunku, co odpowiada tzw. „wirowaniu" bakterii szukającej pożywienia, oraz ruch w tym samym kierunku co ruch poprzedni, co odpowiada „płynięciu" bakterii w kierunku coraz to większej ilości pożywienia (w naturze bakterie poruszają się za pomocą wici, aby przejść do przodu, wić obraca się w kierunku przeciwnym do ruchu wskazówek zegara i bakteria „płynie", podczas gdy obrót wici w drugą stronę skutkuje losowym ruchem w nowym kierunku – bakteria „wiruje"). Bakterie najlepiej przystosowane ulegają podziałowi (reprodukcji) pozostałe wymierają. Bakterie, które utkną w lokalnych optimach są eliminowane lub przesuwane na inne pozycje.

W niniejszym artykule skupimy się na jednym z najnowszych algorytmów inteligencji rojowej, a mianowicie na algorytmie świetlikowym (ang. *Firefly Algorithm*) zaproponowanym w roku 2008 przez Xin-She Yanga [6; 7; 8]. Omówiona zostanie zasada działania tego algorytmu, a także przedstawione i przeanalizowane będą wyniki eksperymentu obliczeniowego, w którym zbadany został wpływ zmian wartości parametrów algorytmu na jego efektywność i skuteczność w znajdowaniu ekstremów globalnych wybranych nieliniowych funkcji jedno i wielomodalnych.

2. Algorytm świetlikowy

Świetliki to niewielkie skrzydlate chrząszcze zdolne do wytwarzania migoczącego światła za pomocą procesu chemicznego. Światło służy świetlikowi do przyciągania partnera, komunikacji w celu obrony terytorium oraz odstraszania drapieżników (informuje, że owad jest niesmaczny, a może i trujący), a także pomaga w zdobywaniu pożywienia (przyciąga inne owady).

2.1. Zasada działania

Algorytm świetlikowy jest procedurą iteracyjną pracującą z populacją potencjalnych rozwiązań (świetlików). Rozmiar populacji będzie oznaczany przez m. Potencjalne rozwiązanie problemu (świetlik i) jest reprezentowane za pomocą ciągu n liczb rzeczywistych x_i (nazywanego położeniem świetlika i), gdzie n jest liczbą zmiennych rozwiązywanego problemu. Z każdym świetlikiem i skojarzona jest wielkość nazywana intensywnością świecenia I_i, wyznaczana na podstawie wartości funkcji celu, $f(x_i)$; np. dla problemu minimalizacji intensywność świecenia może być wyrażona jako odwrotność funkcji celu.

Świetliki o wyższej jasności przyciągają świetliki z mniejszą intensywnością świecenia. W każdej iteracji algorytmu, świetlik i zmienia swoją pozycję x_i biorąc pod uwagę swoją poprzednią pozycję, atrakcyjność jaśniejszego od siebie świetlika j oraz pewien składnik losowy. Najlepszy świetlik w roju porusza się losowo.

Atrakcyjność jest malejącą funkcją odległości między świetlikami, r_{ij}, i wyraża się wzorem:

$$\beta_{ij} = \beta_0 e^{-\gamma r_{ij}^2}, \tag{1}$$

gdzie β_0 i γ są parametrami algorytmu, oznaczającymi, odpowiednio, maksimum atrakcyjności i współczynnik absorpcji, a odległość między świetlikami obliczana jest jako odległość Euklidesowa:

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{n} \left(x_{ik} - x_{jk}\right)^2}. \tag{2}$$

Nowe położenie świetlika i jest wyznaczane ze wzoru:

$$x_i = x_i + \beta_{ij}(x_j - x_i) + \varphi, \tag{3}$$

gdzie φ wielkością kroku losowego wyznaczoną ze wzoru:

$$\varphi = \alpha(rand\ - 0{,}5), \tag{4}$$

gdzie α jest współczynnikiem kroku losowego, $rand$ jest losową liczbą z przedziału $[0; 1]$.

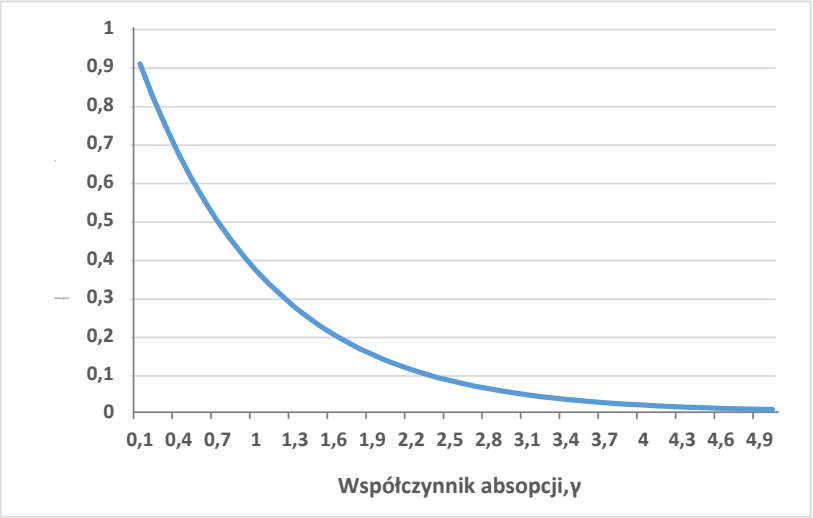
Pseudokod zastosowanego algorytmu świetlikowego został przedstawiony na rysunku 1.

1: Ustaw wartości parametrów algorytmu: m, β_0, γ, α, δ;
2: Wygeneruj początkową populację m świetlików, dla każdego świetlika oblicz intensywność świecenia, zapamiętaj położenie najlepszego świetlika, x_b, i jego intensywność świecenia, I_b;
3: **while** kryterium stopu nie jest spełnione **do:**
4: **for** i $=$ 1 **to** m **do:**
5: **for** j $=$ 1 **to** m **do:**
6: **if** $I_i < I_j$ **then**:
7: Wyznacz odległość między świetlikami: $r_{ij} = \|x_i - x_j\|$;
8: Wyznacz atrakcyjność: $\beta_{ij} = \beta_0 e^{-\gamma r_{ij}^2}$;
9: Wyznacz nowe położenie świetlika i: $x_i = x_i + \beta_{ij}\,(x_j - x_i) + \alpha(rand\ - 0{,}5)$;
10: **endif**
11: **endfor**
12: Aktualizuj intensywność świecenia I_i dla świetlika i;
13: **If** $I_i > I_b$ **then:** $x_b = x_i, I_b = I_i$;
14: **endfor**
15: Wykonaj ruch dla najlepszego świetlika: $x_b = x_b + \alpha(rand\ - 0{,}5)$ i oblicz I_b;
16: Zredukuj wartość współczynnika kroku losowego: $\propto = \delta \cdot \propto$;
17: **endwhile**

Rysunek 1. Pseudokod algorytmu świetlikowego

Jakość działania algorytmu świetlikowego zależy od ustawień jego parametrów. Parametry takie jak współczynnik absorpcji światła γ, współczynnik kroku losowego α i maksymalna atrakcyjność β_0 mają bezpośredni wpływ na ruchy świetlików.

Wartość współczynnika absorpcji ma zasadniczy wpływ na wielkość ruchu świetlika – jej wzrost powoduje zmniejszanie się atrakcyjności, a tym samym zmniejszenie długości ruchu świetlika. Zależność atrakcyjności od wartości współczynnika absorpcji przy ustalonej odległości świetlików ilustruje rysunek 2.



Rysunek 2. Wpływ współczynnika absorpcji światła na wartość atrakcyjności (dla $\beta_0 = 1$ i $r = 1$)

Krok losowy ogranicza możliwość wpadania algorytmu w pułapki lokalnych ekstremów. Pozwala na wykonywanie ruchów najlepszemu świetlikowi oraz pomaga prześcigać go reszcie roju. Ponieważ w początkowej fazie pracy algorytm powinien przeszukać jak największe obszary przestrzeni rozwiązań, to krok losowy powinien być dość duży. Z drugiej strony, za duży krok losowy w fazie końcowej może wpłynąć negatywnie na zbieżność algorytmu. Rozwiązaniem problemu jest wprowadzenie współczynnika redukcji δ ($\delta \in (0,1]$), który zmniejsza wartość współczynnika kroku losowego, α, z każdą kolejną iteracją.

Ostatnim parametrem odnoszącym się do ruchu świetlika jest maksimum atrakcyjności β_0. Gdy $\beta_0 = 0$, ruch świetlika zależy tylko od kroku losowego. Zwykle wartość β_0 zawiera się w przedziale $[0,1]$ i w większości przypadków może być równa 1.

Większy rozmiar populacji m, pozwala zazwyczaj lepiej przeszukiwać przestrzeń potencjalnych rozwiązań i zwiększa szanse na znalezienie optymalnego rozwiązania. Niestety, powoduje on również dłuższy czas działania algorytmu.

Jeżeli chodzi o kryterium stopu, to może być nim narzucona z góry maksymalna liczba iteracji – zbyt mała wartość sprawi, że algorytm nie osiągnie ani ekstremum globalnego ani dobrego ekstremum lokalnego, natomiast zbyt duża może niepotrzebnie wydłużać jego pracę. Również, algorytm może kończyć działanie, gdy nie następuje poprawa rozwiązania przez zadaną liczbę iteracji.

2.2. Zastosowania

Mimo że algorytm świetlikowy jest stosunkowo nowy, doczekał się już wielu odmian i całkiem sporej gamy zastosowań w rozwiązywaniu problemów optymalizacyjnych świata rzeczywistego [9; 10; 11; 12]. Do jego zastosowań należą między innymi: obróbka obrazów cyfrowych (mapowanie, transformacja, odszumianie, kompresja), śledzenie obiektów, grafika satelitarna, wykrywanie twarzy, obróbka elektrochemiczna, projektowanie konstrukcji stalowych, planowanie bezprzewodowych sieci komputerowych.

Główną zaletą algorytmu świetlikowego jest to, że dzięki mechanizmowi spadku atrakcyjności wraz ze zwiększaniem się odległości między świetlikami, cała populacja może podzielić się na mniejsze grupy przeszukujące odrębne obszary przestrzeni rozwiązań, zwiększając tym samym szanse na znalezienie globalnego ekstremum.

Jeśli chodzi o wady, to do największych należy złożoność obliczeniowa kwadratowo zależna od rozmiaru populacji (zagnieżdżone pętle w krokach 4, 5 na rysunku 1), która może być sporym ograniczeniem w rozwiązywaniu trudniejszych zagadnień wymagających zwiększenia liczby świetlików.

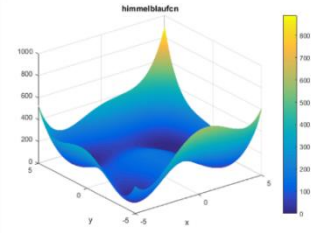
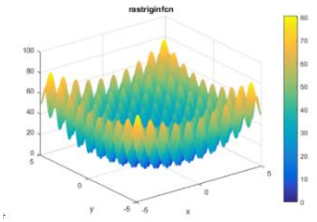
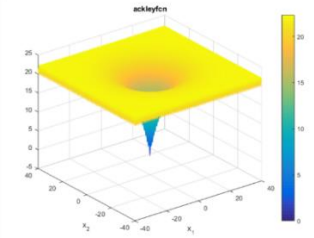
3. Funkcje testowe do optymalizacji

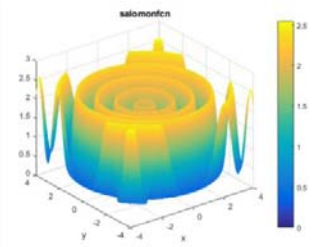
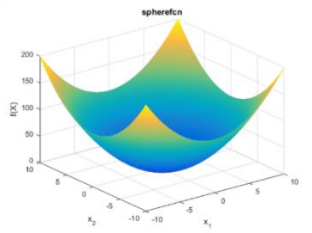
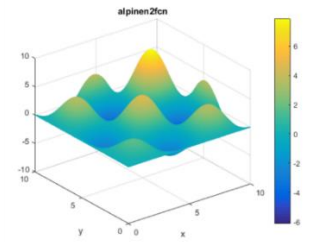
Jakość algorytmów optymalizacyjnych jest często oceniana przy użyciu standardowych funkcji testowych znanych z literatury. Występują funkcje jednomodalne, w których istnieje tylko jedno ekstremum oraz wielomodalne posiadające wiele różnych ekstremów lokalnych. Jeżeli chodzi o liczbę wymiarów, to może ona być ściśle

określona, np. funkcja dwóch zmiennych Himmelblau, trzech zmiennych Wolfe, lub nieokreślona, np. funkcja n-zmiennych Alpine N. 2 [13].

Do badania algorytmu świetlikowego opisanego w niniejszym artykule wykorzystane zostały następujące funkcje testowe: Himmelblau, Rastrigin, Ackley, Salomon, Sphere, Alpine N. 2 [13]. W tabeli 1 zostały pokazane wzory funkcji testowych oraz graficzne wizualizacje ich dwuwymiarowych wersji.

Tabela 1. Badane funkcje testowe

Nazwa funkcji	Wzór funkcji	Wykres dla dwóch zmiennych*)
Himmelblau	$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$	
Rastrigin	$f(x_1,..,x_n) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i))$	
Ackley	$f(x_1,..,x_n) = -20\exp\left(-0{,}2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n}cos(2\pi x_i)\right) + 20 + e$	

Nazwa funkcji	Wzór funkcji	Wykres dla dwóch zmiennych*)
Salomon	$f(x_1,..,x_n) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0{,}1\sqrt{\sum_{i=1}^{n} x_i^2}$	
Sphere	$f(x_1,..,x_n) = \sum_{i=1}^{n} x_i^2$	
Alpine N. 2	$f(x_1,..,x_n) = \prod_{i=1}^{n} \sqrt{x_i}\sin(x_i)$	

*) Wykorzystano rysunki z [13].

W tabeli 2 przedstawione są ekstrema funkcji, zakresy wartości zmiennych oraz liczba wymiarów.

Eksperyment polegał na dziesięciokrotnym uruchomieniu algorytmu dla każdej z wymienionych funkcji dla 27 różnych konfiguracji parametrów algorytmu, co w sumie dało 1620 uruchomień algorytmu. W tabeli 3 znajduje się zestawienie badanych wartości parametrów.

Tabela 2. Ekstrema, zakresy wartości zmiennych i liczba zmiennych dla badanych funkcji testowych

Nazwa funkcji testowej	Szukane ekstremum (minimum)	Badany zakres wartości zmiennych	Liczba zmiennych
Himmelblau	0 dla (3, 2) lub 0 dla (−2.805118,3.283186) lub 0 dla (−3.779310,−3.283186) lub 0 dla (3.584458,−1.848126)	[-6; 6]	2
Rastrigin	0 dla $x_i = 0$	[-5,12; 5,12]	4
Ackley	0 dla $x_i = 0$	[-32,768; 32,768]	6
Salomon	0 dla $x_i = 0$	[-100; 100]	6
Sphere	0 dla $x_i = 0$	[-5,12; 5,12]	8
Alpine N. 2 (zanegowana)	$-2{,}808^n$ dla $x_i = 7{,}917$	[0; 10]	8

Tabela 3. Wykaz badanych wartości parametrów algorytmu

Współczynnik kroku losowego α	Współczynnik redukcji kroku losowego δ	Rozmiar populacji m
0,8	0,980	90
0,2	0,990	60
0,1	0,998	30

Ze względu na ograniczenia sprzętowe, maksymalna liczba iteracji została uzależniona od rozmiaru populacji. Maksymalna liczba iteracji jest wyznaczona wzorem $maxGen = 8100000/m^2$. Liczba 8100000 została wyznaczona metodą prób i błędów jako wartość, dla której każda funkcja jest w stanie się wykonać we względnie przystępnym czasie. Obok maksymalnej liczby iteracji zostały zastosowane dwa dodatkowe kryteria kończące pracę algorytmu: maksymalna liczba iteracji bez poprawy najlepszego wyniku, ustawiona na 50, oraz osiągnięcie minimum globalnego dla badanej funkcji testowej.

4. Wyniki eksperymentu obliczeniowego

W tym rozdziale przedstawiono i przedyskutowano wyniki przeprowadzonego eksperymentu obliczeniowego. Tabela 4 przedstawia wykaz skrótów użytych do opisu otrzymanych wyników. Szczegółowe wyniki eksperymentu przedstawiono na kolejnych rysunkach – czerwoną przerywaną linią zaznaczono wartość minimum globalnego.

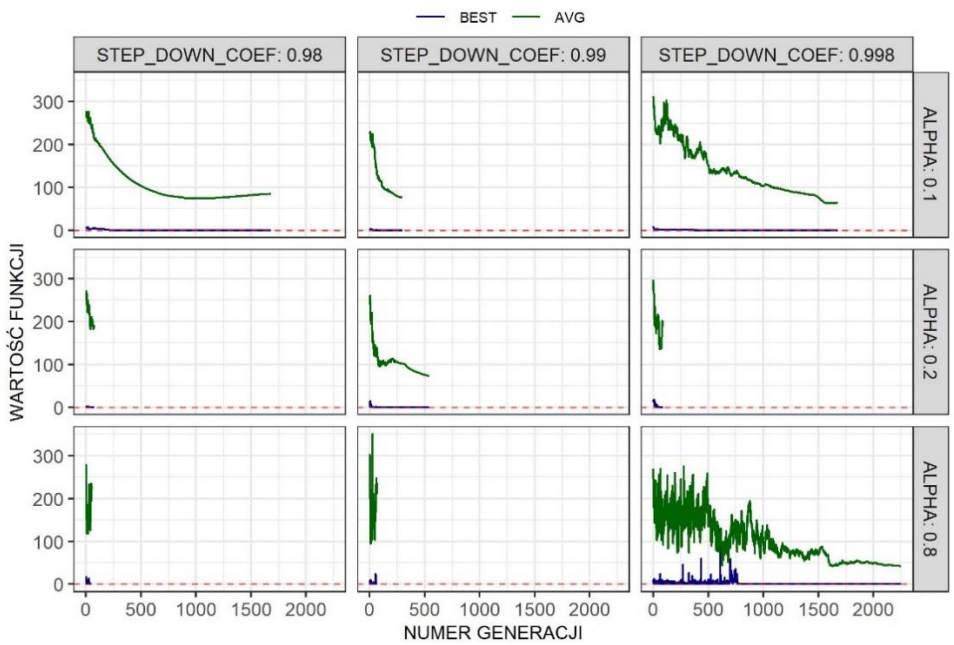
Tabela 4. Wykaz skrótów użytych podczas prezentacji wyników

Nazwa	Opis
POP_SIZE	Rozmiar populacji
ALPHA	Współczynnik kroku losowego α
STEP_DOWN_COEF	Współczynnik redukujący wartość kroku losowego δ
MIN_RESULT	Najlepszy wynik osiągnięty we wszystkich przebiegach algorytmu
MAX_RESULT	Najgorszy wynik ze wszystkich przebiegów algorytmu
AVG_RESULT	Średni wynik dla wszystkich przebiegów algorytmu
STD_RESULT	Odchylenie standardowe wyników
AVG_LAST_GEN	Średnia liczba iteracji wykonana przez algorytm
MIN_WORK_TIME	Najkrótszy czas pracy algorytmu spośród wszystkich uruchomień
MAX_WORK_TIME	Najdłuższy czas pracy algorytmu spośród wszystkich uruchomień
AVG_WORK_TIME	Średni czas pracy algorytmu
BEST	Wartość badanej funkcji dla najlepszego świetlika w bieżącej populacji
AVG	Średnia wartość badanej funkcji dla całej populacji

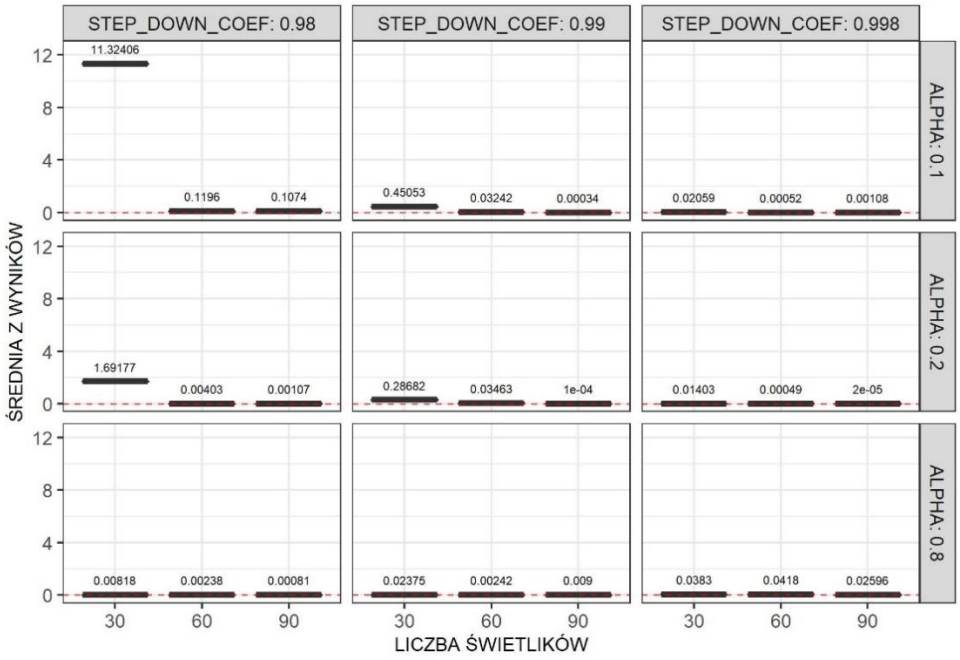
4.1. Funkcja testowa Himmelblau

Pierwsze testy algorytmu zostały przeprowadzone z wykorzystaniem funkcji dwóch zmiennych Himmelblau. Przykładowe przebiegi algorytmu pokazane są na rysunku 3. Widać tu, że już w pierwszych iteracjach algorytmu najlepszy świetlik populacji zbliżał się do globalnego minimum. Rysunek 4 przedstawia średnie wyniki dla 10 przebiegów algorytmu dla każdej badanej konfiguracji jego parametrów. Należy zwrócić uwagę, że wyniki dla 30 świetlików znacząco odbiegają od poszukiwanego ekstremum w przypadku współczynnika redukcji $\delta = 0{,}98$. Pozostałe wyniki znajdują się dość blisko poszukiwanej wartości (czerwona przerywana linia).

W tabeli 5 przedstawiono skuteczność (w procentach) znajdowania ekstremum. Nie udało się osiągnąć skuteczności równej 100% dla żadnego badanego wariantu. Najlepszą skutecznością, równą 90%, wykazały się dwa warianty $(m;\ \alpha;\ \delta) = (90;\ 0{,}2;\ 0{,}998)$ oraz $(m;\ \alpha;\ \delta) = (60;\ 0{,}1;\ 0{,}998)$. Dla większości konfiguracji przy zastosowaniu 60 i 90 świetlików udało się znaleźć minimum przynajmniej jeden raz. Spośród wszystkich wariantów, osiem uzyskało skuteczność co najmniej 50%. Dla populacji 30 świetlików skuteczność jest bardzo niska.



Rysunek 3. Przykładowe przebiegi algorytmu (przy $m = 60$) dla funkcji testowej Himmelblau



Rysunek 4. Średnie wyniki dla funkcji testowej Himmelblau

Tabela 5. Skuteczność znajdowania ekstremów dla funkcji testowej Himmelblau (w %)

POP_SIZE	ALPHA	STEP_DOWN_COEF		
		0,98	0,99	0,998
30	0,1	0	0	40
	0,2	0	0	10
	0,8	0	20	10
60	0,1	0	20	90
	0,2	30	50	40
	0,8	40	30	0
90	0,1	10	60	60
	0,2	50	80	90
	0,8	70	10	0

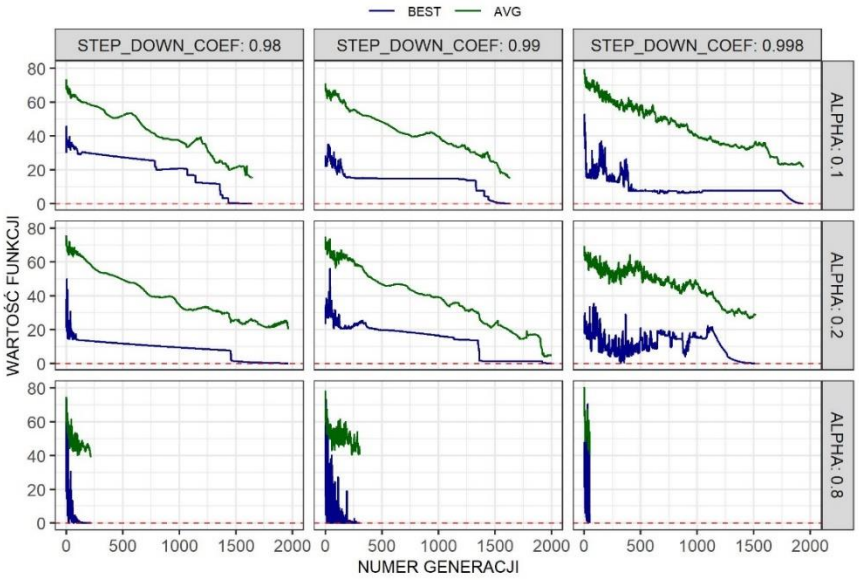
Tabela 6. Szczegółowe wyniki pracy algorytmu dla funkcji testowej Himmelblau

POP_SIZE	ALPHA	STEP_DOWN_COEF	MIN_RESULT	MAX_RESULT	AVG_RESULT	STD_RESULT	AVG_LAST_GEN	MIN_WORK_TIME	MAX_WORK_TIME	AVG_WORK_TIME
30	0,1	0,980	0,6227	49,6761	11,3241	13,5672	1605,9000	00:00:07	00:00:18	00:00:16
		0,990	0,0015	1,1703	0,4505	0,4092	3228,3000	00:00:24	00:00:34	00:00:31
		0,998	0,0000	0,1501	0,0206	0,0456	4919,1000	00:00:16	00:01:25	00:00:44
	0,2	0,980	0,0144	5,4644	1,6918	1,7148	1578,6000	00:00:09	00:00:19	00:00:16
		0,990	0,0003	0,9889	0,2868	0,4283	3586,1000	00:00:07	00:01:09	00:00:33
		0,998	0,0000	0,0612	0,0140	0,0218	4539,7000	00:00:03	00:01:25	00:00:42
	0,8	0,980	0,0001	0,0431	0,0082	0,0129	3290,5000	00:00:16	00:01:23	00:00:30
		0,990	0,0000	0,1664	0,0238	0,0488	2792,0000	00:00:01	00:00:33	00:00:25
		0,998	0,0000	0,1514	0,0383	0,0435	6426,9000	00:00:00	00:01:12	00:00:47
60	0,1	0,980	0,0004	0,5611	0,1196	0,1818	1355,9000	00:00:14	00:02:11	00:01:05
		0,990	0,0000	0,1383	0,0324	0,0520	1790,8000	00:00:13	00:02:39	00:01:20
		0,998	0,0000	0,0052	0,0005	0,0016	1047,3000	00:00:10	00:02:20	00:01:07
	0,2	0,980	0,0000	0,0175	0,0040	0,0059	892,1000	00:00:03	00:01:56	00:00:45
		0,990	0,0000	0,3416	0,0346	0,1023	1180,9000	00:00:12	00:02:55	00:01:15
		0,998	0,0000	0,0027	0,0005	0,0008	1626,6000	00:00:06	00:02:51	00:01:39
	0,8	0,980	0,0000	0,0094	0,0024	0,0035	956,4000	00:00:02	00:02:05	00:00:50
		0,990	0,0000	0,0091	0,0024	0,0030	1445,6000	00:00:01	00:02:19	00:01:09
		0,998	0,0008	0,1887	0,0418	0,0534	2250,0000	00:01:35	00:02:10	00:01:54
90	0,1	0,980	0,0000	0,7038	0,1074	0,2068	840,2000	00:00:28	00:01:35	00:01:17
		0,990	0,0000	0,0013	0,0003	0,0005	535,2000	00:00:14	00:01:32	00:00:49
		0,998	0,0000	0,0102	0,0011	0,0030	533,9000	00:00:04	00:01:34	00:00:49
	0,2	0,980	0,0000	0,0065	0,0011	0,0020	615,9000	00:00:07	00:01:32	00:00:56
		0,990	0,0000	0,0007	0,0001	0,0002	413,4000	00:00:07	00:01:31	00:00:38
		0,998	0,0000	0,0002	0,0000	0,0001	252,8000	00:00:03	00:01:28	00:00:22
	0,8	0,980	0,0000	0,0057	0,0008	0,0017	365,0000	00:00:04	00:01:25	00:00:31
		0,990	0,0000	0,0635	0,0090	0,0183	884,2000	00:00:03	00:01:25	00:01:12
		0,998	0,0001	0,0843	0,0260	0,0305	1000,0000	00:01:21	00:01:36	00:01:29

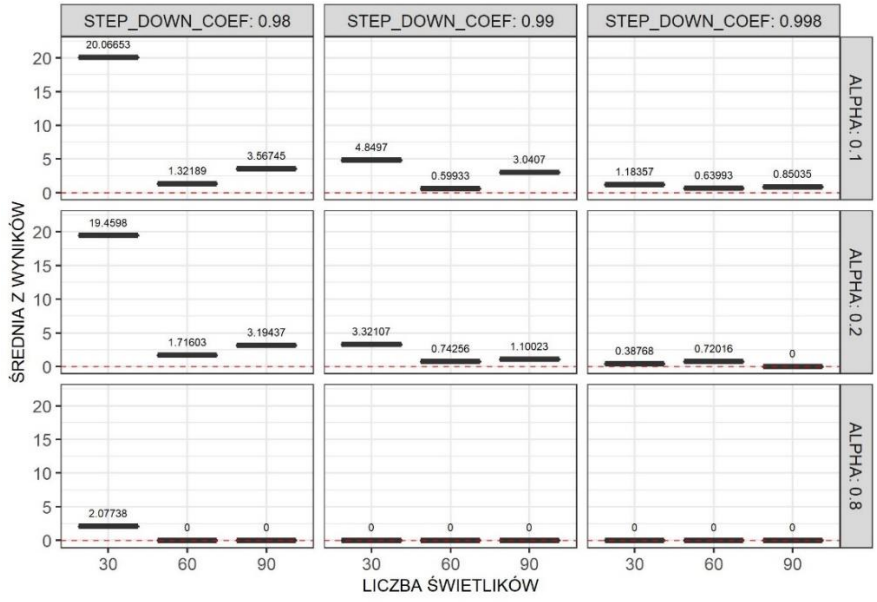
Analizując wyniki z tabeli 6, można zauważyć, że najmniej iteracji, średnio 253, było wykonywanych przy zastosowaniu konfiguracji $(m;\ \alpha;\ \delta) = (90;\ 0{,}2\ ;\ 0{,}998)$, natomiast najwięcej iteracji, średnio 6427, dla konfiguracji $(m;\ \alpha;\ \delta)$ $(30;\ 0{,}8;\ 0{,}998)$. Średni czas pracy algorytmu po wszystkich konfiguracjach wynosił 53 sekundy. Najkrótsze uruchomienie spośród wszystkich trwało 1 sekundę, a najdłuższe ok. 3 minuty. Najgorszy wynik równy 49,6761 oraz największe odchylenie standardowe równe 13,5672 wystąpiły dla konfiguracji $(m;\ \alpha;\ \delta) = (30;\ 0{,}1;\ 0{,}98)$.

4.2. Funkcja testowa Rastrigin

Funkcja testowa Rastrigin została poddana badaniu w wariancie dla 4 zmiennych. Przykładowe wykresy zbieżności algorytmu pokazane są na rysunku 5. Przy zastosowaniu współczynnika kroku losowego $\alpha = 0{,}8$, zaobserwowano duże skoki wartości wyników w kolejnych iteracjach.



Rysunek 5. Przykładowe przebiegi algorytmu (przy $m = 60$) dla funkcji testowej Rastrigin



Rysunek 6. Średnie wyniki dla funkcji testowej Rastrigin

Na rysunku 6 widać, że średnie wyniki dla $m = 30$ oraz δ = 0,98 bez względu na zastosowaną wartość α znacząco odbiegają od poszukiwanego ekstremum. Jednocześnie widoczna jest 100% skuteczność w przypadku zastosowania współczynnika α równego 0,8 z wyjątkiem wspomnianej konfiguracji z 30 świetlikami.

Potwierdzają to wyniki z tabeli 7. 100% skuteczności wykazuje również konfiguracja $(m;\ \alpha;\ \delta) = (90;\ 0{,}2;\ 0{,}998)$. Łącznie 9 konfiguracji osiągnęło skuteczność równą 100%, a 13 co najmniej 50%.

Tabela 7. Skuteczność znajdowania ekstremów dla funkcji testowej Rastrigin (w %)

POP_SIZE	ALPHA	STEP_DOWN_COEF		
		0,98	**0,99**	**0,998**
30	**0,1**	0	0	10
	0,2	0	0	70
	0,8	10	100	100
60	**0,1**	10	40	60
	0,2	10	50	60
	0,8	100	100	100
90	**0,1**	0	0	30
	0,2	0	20	100
	0,8	100	100	100

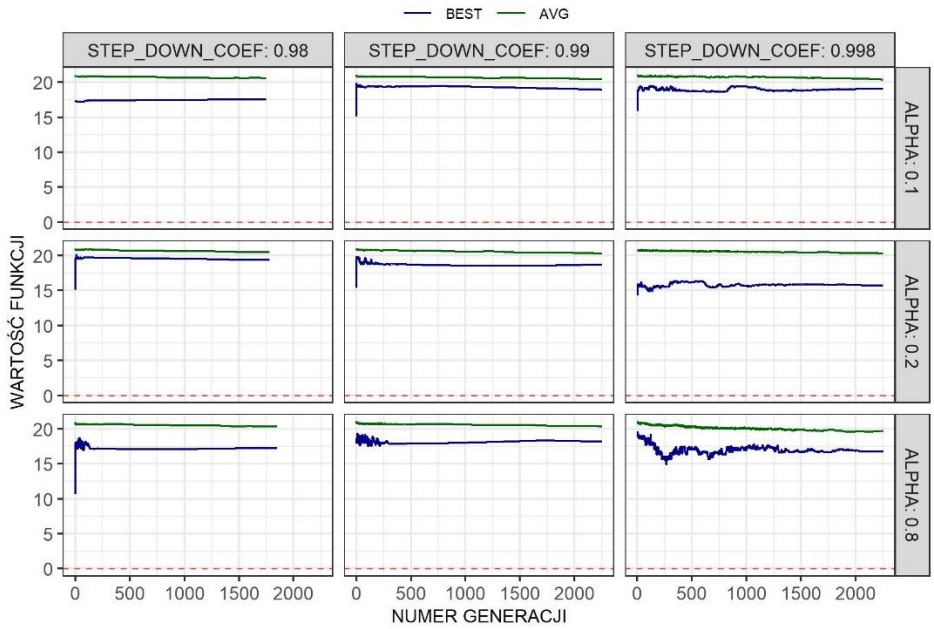
Analizując wyniki zamieszczone w tabeli 8 można zauważyć, że najmniej iteracji – średnio 40 – było wykonywanych przy zastosowaniu konfiguracji $(m;\ \alpha;\ \delta) = (90;\ 0{,}8;\ 0{,}998)$, natomiast najwięcej iteracji, średnio 8514, dla konfiguracji $(m;\ \alpha;\ \delta) = (30;\ 0{,}1;\ 0{,}998)$. Najkrótszy czas pracy algorytmu wynosił 4 sekundy, a najdłuższy ok. 12 minut. Średni czas pracy dla wszystkich wariantów wynosił 4 min. Najgorszy wynik (27,1231) z największym odchyleniem standardowym (3,7664) wystąpił dla konfiguracji $(m;\ \alpha;\ \delta) = (30;\ 0{,}1;\ 0{,}98)$.

Tabela 8. Szczegółowe wyniki pracy algorytmu dla funkcji testowej Rastrigin

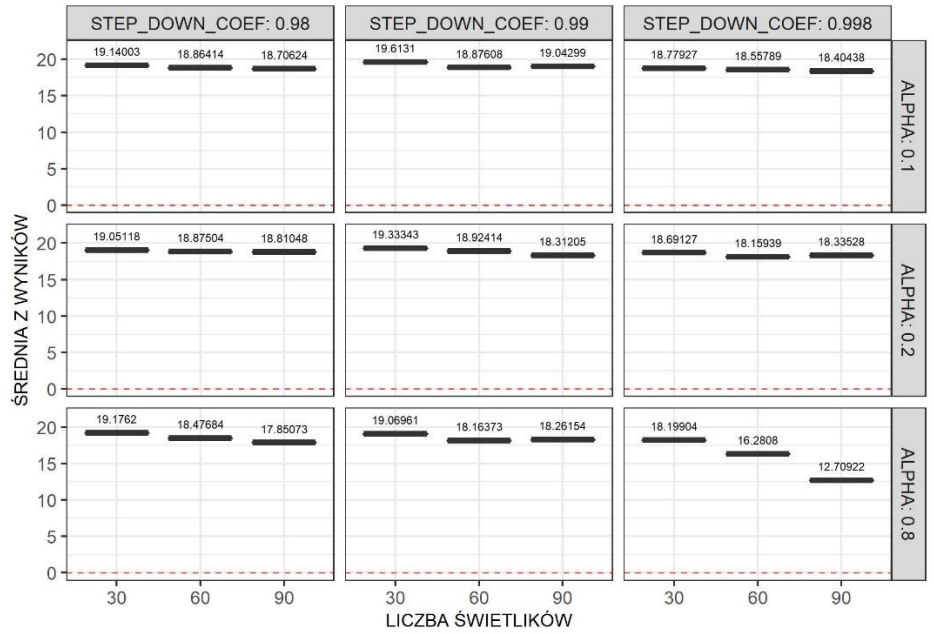
POP_SIZE	ALPHA	STEP_DOWN_COEF	MIN_RESULT	MAX_RESULT	AVG_RESULT	STD_RESULT	AVG_LAST_GEN	MIN_WORK_TIME	MAX_WORK_TIME	AVG_WORK_TIME
30	0,1	0,980	12,5081	27,1231	20,0665	3,7664	1993,3000	00:01:17	00:02:07	0:01:27
		0,990	0,3545	9,4085	4,8497	2,8530	3771,9000	00:02:28	00:02:53	0:02:42
		0,998	0,0000	2,5978	1,1836	0,6357	8514,3000	00:03:01	00:10:23	0:07:01
	0,2	0,980	13,3750	23,3056	19,4598	3,2460	2001,7000	00:01:17	00:01:56	00:01:28
		0,990	0,1093	6,9838	3,3211	2,4967	3797,8000	00:02:25	00:02:54	00:02:41
		0,998	0,0000	1,4534	0,3877	0,6033	6666,1000	00:02:56	00:10:15	00:05:49
	0,8	0,980	0,0000	4,1150	2,0774	1,8967	1768,0000	00:00:17	00:01:33	00:01:18
		0,990	0,0000	0,0000	0,0000	0,0000	582,0000	00:00:18	00:00:34	00:00:26
		0,998	0,0000	0,0000	0,0000	0,0000	655,7000	00:00:04	00:01:10	00:00:38
60	0,1	0,980	0,0000	3,5277	1,3219	1,2299	1906,6000	00:06:02	00:12:07	00:08:11
		0,990	0,0000	1,9036	0,5993	0,6090	2075,9000	00:05:03	00:09:33	00:07:00
		0,998	0,0000	2,0851	0,6399	0,8346	1917,3000	00:04:18	00:10:54	00:06:38
	0,2	0,980	0,0000	5,9032	1,7160	1,9672	1969,1000	00:05:13	00:07:44	00:05:51
		0,990	0,0000	3,4328	0,7426	1,1258	2063,8000	00:04:51	00:11:49	00:07:49
		0,998	0,0000	2,3685	0,7202	0,9269	1656,2000	00:03:00	00:07:33	00:05:20
	0,8	0,980	0,0000	0,0000	0,0000	0,0000	203,8000	00:00:07	00:01:22	00:00:51
		0,990	0,0000	0,0000	0,0000	0,0000	320,4000	00:00:55	00:02:00	00:01:19
		0,998	0,0000	0,0000	0,0000	0,0000	91,8000	00:00:06	00:00:34	00:00:18
90	0,1	0,980	1,1143	6,9574	3,5675	1,7848	1000,0000	00:06:43	00:06:47	00:06:44
		0,990	0,0126	7,6933	3,0407	2,4217	1000,0000	00:06:43	00:06:51	00:06:48
		0,998	0,0000	4,1994	0,8504	1,3173	982,3000	00:06:16	00:07:05	00:06:52
	0,2	0,980	0,0004	9,7663	3,1944	2,7476	1000,0000	00:06:42	00:06:48	00:06:44
		0,990	0,0000	5,4017	1,1002	1,6121	975,9000	00:05:36	00:06:53	00:06:39
		0,998	0,0000	0,0000	0,0000	0,0000	662,1000	00:04:01	00:05:59	00:04:58
	0,8	0,980	0,0000	0,0000	0,0000	0,0000	164,0000	00:01:00	00:01:19	00:01:08
		0,990	0,0000	0,0000	0,0000	0,0000	147,6000	00:00:09	00:01:55	00:01:02
		0,998	0,0000	0,0000	0,0000	0,0000	40,0000	00:00:05	00:00:36	00:00:18

4.3. Funkcja testowa Ackley

Funkcja testowa Ackley została poddana badaniu w wariancie dla 6 zmiennych. Na rysunku 7 widać płaskie przebiegi algorytmu. Rysunek 8 pokazuje, że najlepsze średnie wyniki udawało się uzyskać przy konfiguracji $(m;\ \alpha;\ \delta) = (90; 0{,}8; 0{,}998)$. Jednak algorytm był zdecydowanie daleki od znalezienia optymalnej wartości funkcji. Wydaje się, że zwiększając dalej wartość α oraz δ, algorytm byłby w stanie znaleźć poszukiwane ekstremum. We wszystkich badanych konfiguracjach ustawień parametrów algorytm ani razu nie zdołał znaleźć minimum globalnego.



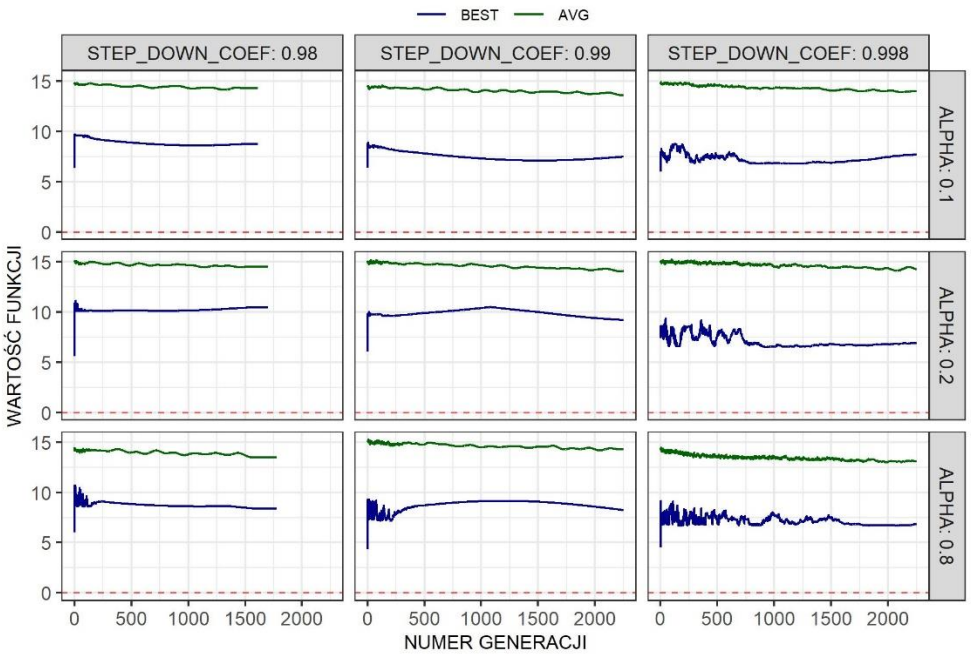
Rysunek 7. Przykładowe przebiegi algorytmu (przy $m = 60$) dla funkcji testowej Ackley



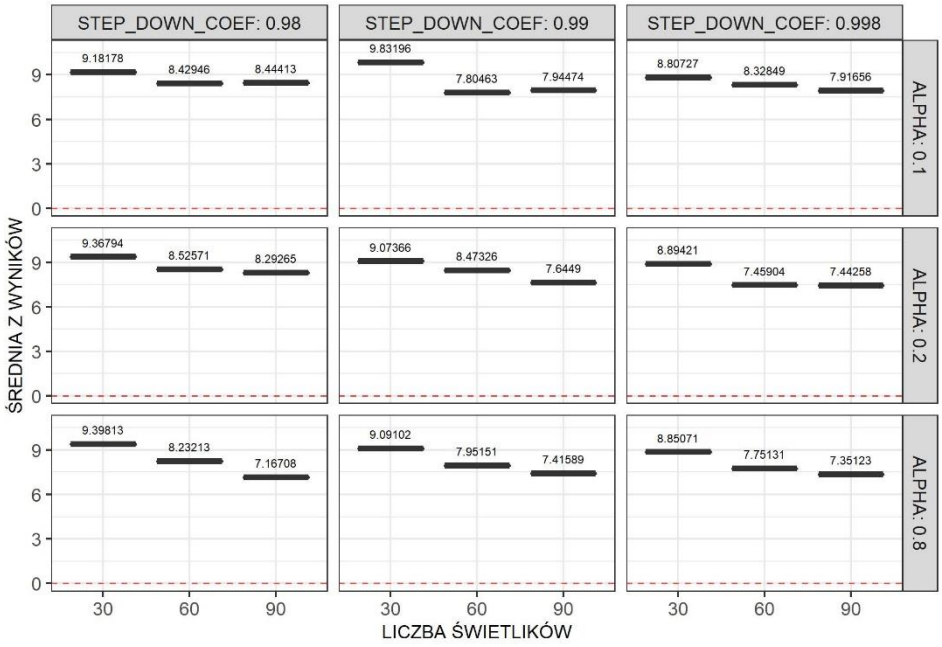
Rysunek 8. Średnie wyniki dla funkcji testowej Ackley

4.4. Funkcja testowa Salomon

Analizie został poddany wariant funkcji Salomon dla 6 zmiennych. Zastosowany tu algorytm nigdy nie znalazł ekstremum funkcji. Znalezione wyniki oscylowały w zakresie wartości funkcji równej 7-9, a poszukiwane minimum to wartość 0. Na podstawie przeprowadzonych badań trudno określić wpływ innych parametrów algorytmu na skuteczność jego działania. Zmiana tych parametrów nie poprawiała skuteczności znalezienia minimum. Przykładowe przebiegi i średnie wyniki z 10 przebiegów przedstawione są, odpowiednio, na rysunkach 9 i 10.



Rysunek 9. Przykładowe przebiegi algorytmu (przy $m = 60$) dla funkcji testowej Salomon

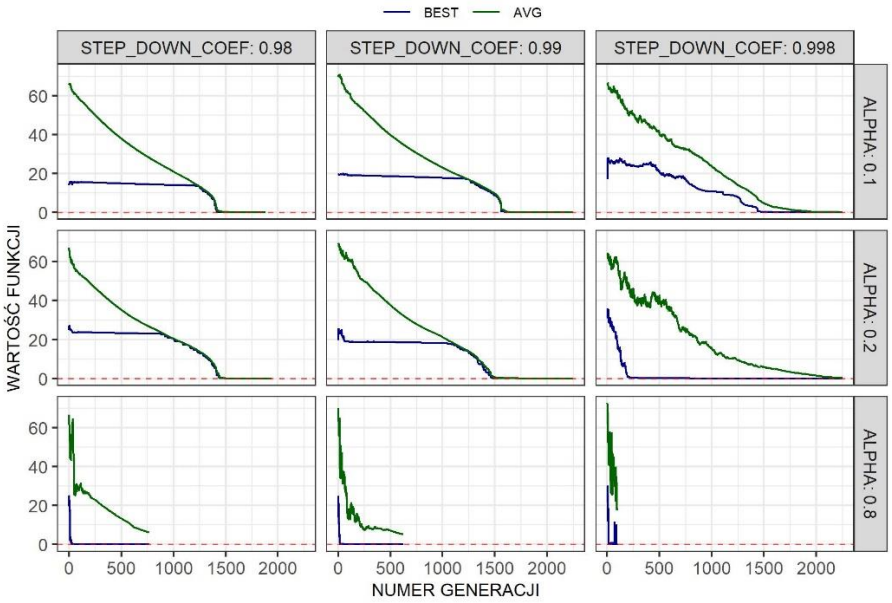


Rysunek 10. Średnie wyniki dla funkcji testowej Salomon

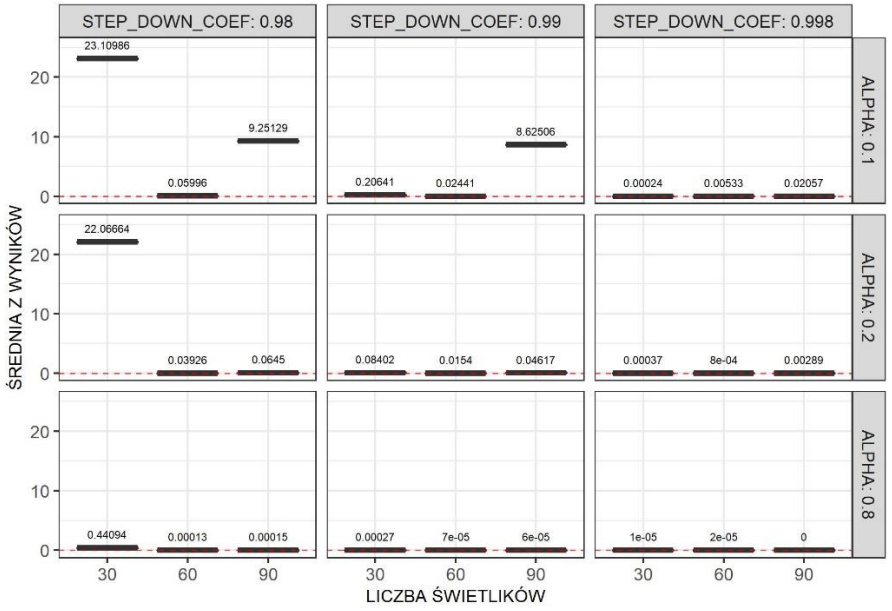
4.5. Funkcja testowa Sphere

Analizie została poddana funkcja testowa Sphere dla 8 zmiennych. Jest to funkcja posiadająca tylko jedno ekstremum. Na rysunkach 11 i 12 widać, że wysokie wartości współczynnika α sprzyjały szybszej zbieżności algorytmu w okolice ekstremum funkcji oraz wyższej skuteczności algorytmu.

Tabela 9 pokazuje, że dla 5 konfiguracji została osiągnięta skuteczność co najmniej 50%. Większa liczba świetlików nieznacznie poprawiała skuteczność, jednak zasadniczo do sukcesu przyczyniały się wysokie wartości parametrów α oraz δ. Warto dodać, że przy najwyższych badanych wartościach α oraz δ wzrost skuteczności algorytmu był skokowy. Skuteczność 100% wystąpiła dla konfiguracji $(m;\ \alpha;\ \delta)$= (90; 0,8; 0,998). Przy $\alpha = 0{,}8$ i $\delta = 0{,}998$ skuteczność utrzymuje się na wysokim poziomie 90% również dla mniejszej liczby świetlików (60 i 30). Potwierdza to rysunek 12, na którym widać, że średnie wyniki z 10 przebiegów algorytmu są bliskie minimum globalnego przy $\alpha = 0{,}8$ i $\delta = 0{,}998$ niezależnie od rozmiaru populacji.



Rysunek 11. Przykładowe przebiegi algorytmu (przy $m = 60$) dla funkcji testowej Sphere



Rysunek 12. Średnie wyniki dla funkcji testowej Sphere

Tabela 9. Skuteczność znajdowania ekstremów dla funkcji testowej Sphere (w procentach)

POP_SIZE	ALPHA	STEP_DOWN_COEF		
		0,98	**0,99**	**0,998**
30	**0,1**	0	0	10
	0,2	0	0	0
	0,8	10	0	90
60	**0,1**	0	0	0
	0,2	0	0	0
	0,8	20	60	90
90	**0,1**	0	0	0
	0,2	0	0	0
	0,8	30	60	100

Tabela 10. Szczegółowe wyniki pracy algorytmu dla funkcji testowej Sphere

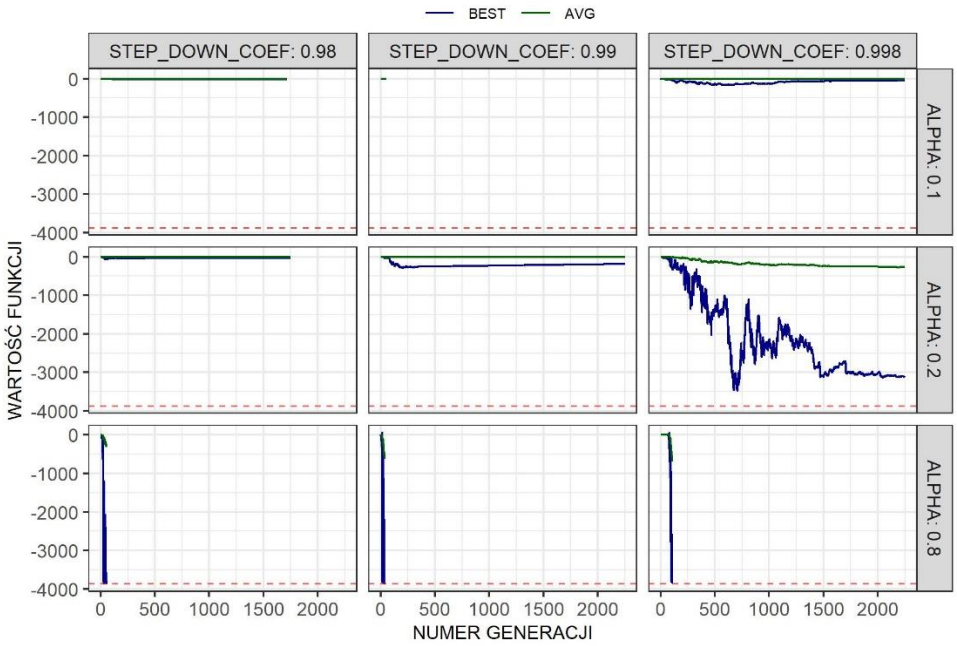
POP_SIZE	ALPHA	STEP_DOWN_COEF	MIN_RESULT	MAX_RESULT	AVG_RESULT	STD_RESULT	AVG_LAST_GEN	MIN_WORK_TIME	MAX_WORK_TIME	AVG_WORK_TIME
30	0,1	0,980	15,5585	30,1525	23,1099	4,4539	2347,7000	00:00:37	00:01:14	00:00:47
		0,990	0,0445	0,4455	0,2064	0,1291	3702,0000	00:01:02	00:01:21	00:01:07
		0,998	0,0000	0,0004	0,0002	0,0001	5883,9000	00:01:09	00:01:21	00:01:15
	0,2	0,980	14,5599	31,4366	22,0666	4,7767	2210,4000	00:00:37	00:01:08	00:00:44
		0,990	0,0238	0,1768	0,0840	0,0443	3825,7000	00:00:56	00:01:04	00:01:01
		0,998	0,0001	0,0011	0,0004	0,0003	5689,4000	00:00:57	00:01:15	00:01:07
	0,8	0,980	0,0000	3,9419	0,4409	1,1684	1797,4000	00:00:03	00:00:43	00:00:37
		0,990	0,0001	0,0005	0,0003	0,0001	1647,7000	00:00:18	00:00:39	00:00:26
		0,998	0,0000	0,0001	0,0000	0,0000	2110,9000	00:00:01	00:00:08	00:00:06
60	0,1	0,980	0,0316	0,0993	0,0600	0,0209	1895,7000	00:01:59	00:03:50	00:02:20
		0,990	0,0050	0,0444	0,0244	0,0113	2250,0000	00:02:07	00:04:04	00:02:30
		0,998	0,0003	0,0196	0,0053	0,0057	2250,0000	00:04:20	00:05:23	00:04:47
	0,2	0,980	0,0115	0,0830	0,0393	0,0250	1936,9000	00:01:53	00:03:36	00:02:15
		0,990	0,0064	0,0329	0,0154	0,0084	2250,0000	00:02:02	00:03:50	00:02:22
		0,998	0,0002	0,0022	0,0008	0,0007	2250,0000	00:03:25	00:04:41	00:04:05
	0,8	0,980	0,0000	0,0004	0,0001	0,0001	897,5000	00:00:05	00:01:16	00:00:54
		0,990	0,0000	0,0003	0,0001	0,0001	729,1000	00:00:05	00:00:44	00:00:21
		0,998	0,0000	0,0002	0,0000	0,0001	283,8000	00:00:05	00:00:26	00:00:11
90	0,1	0,980	5,8351	12,6867	9,2513	2,0740	1000,0000	00:02:50	00:03:02	00:02:57
		0,990	2,6218	12,8258	8,6251	3,1628	1000,0000	00:02:53	00:03:10	00:03:01
		0,998	0,0079	0,0739	0,0206	0,0187	1000,0000	00:03:15	00:03:23	00:03:19
	0,2	0,980	0,0194	0,1612	0,0645	0,0384	1000,0000	00:02:51	00:03:00	00:02:55
		0,990	0,0065	0,1603	0,0462	0,0426	1000,0000	00:02:57	00:03:10	00:03:04
		0,998	0,0013	0,0043	0,0029	0,0009	1000,0000	00:02:10	00:02:53	00:02:33
	0,8	0,980	0,0000	0,0006	0,0002	0,0002	465,9000	00:00:03	00:01:13	00:00:44
		0,990	0,0000	0,0003	0,0001	0,0001	486,4000	00:00:06	00:00:20	00:00:13
		0,998	0,0000	0,0000	0,0000	0,0000	43,0000	00:00:04	00:00:13	00:00:08

Tabela 10 zawiera szczegółowe wyniki działania algorytmu dla funkcji Sphere. Analizując wartość odchylenia standardowego można stwierdzić, że największą zmiennością odznaczały się wyniki dla $m = 30$ i $\delta = 0{,}98$. Jeżeli chodzi o czas pracy algorytmu, to dla $(m;\ \alpha;\ \delta) = (90;\ 0{,}8\ ;\ 0{,}998)$ ekstremum funkcji było znalezione

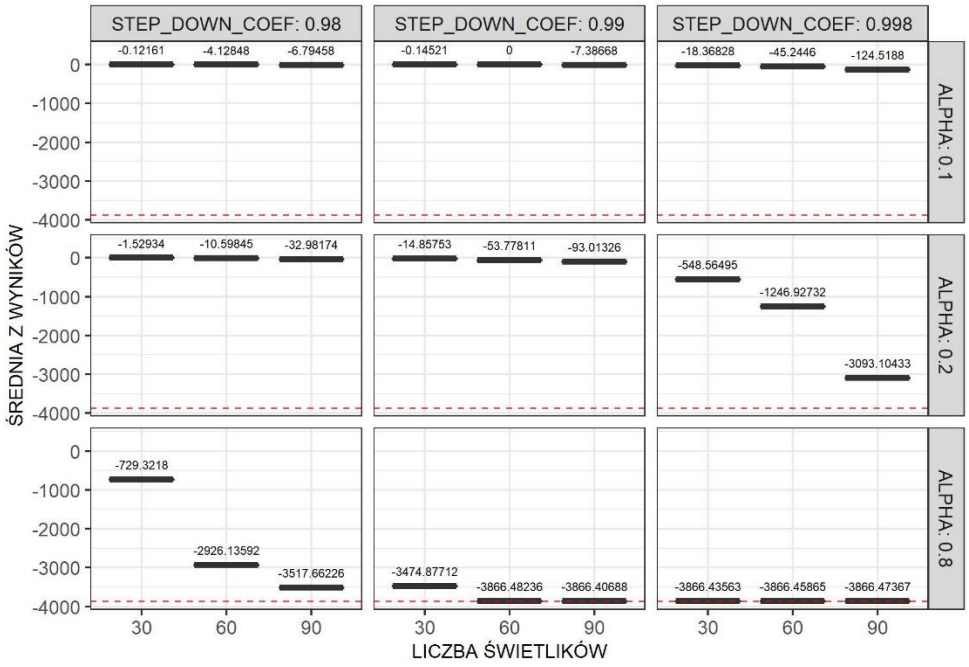
nawet w ciągu 4 sekund. Warto nadmienić, że wysoka wartość $\alpha = 0{,}8$ znacznie poprawia wydajność algorytmu, gdyż zmniejsza liczbę iteracji ponad dwukrotnie. Dla $m = 60$ i $m = 90$ jednoczesny wzrost parametru δ także pozytywnie wpływa na liczbę iteracji.

4.6. Funkcja testowa Alpine N.2

Działanie algorytmu świetlikowego zostało zbadane dla funkcji testowej Alpine N.2 w wariancie z 8 zmiennymi. Na rysunku 13 można zobaczyć, że dla wartości $\alpha = 0{,}1$ oraz $0{,}2$ wykresy przebiegu są najczęściej płaskie, podczas gdy dla $\alpha = 0{,}8$ wyniki wyraźnie zbiegają w kierunku minimum globalnego. Zachodzi to zwłaszcza dla populacji o większym rozmiarze, co widać na rysunku 14.



Rysunek 13. Przykładowe przebiegi algorytmu (przy $m = 60$) dla funkcji testowej Alpine N.2



Rysunek 14. Średnie wyniki dla funkcji testowej Alpine N.2

Tabela 11. Skuteczność znajdowania ekstremów dla funkcji testowej Alpine N.2 (w %)

POP_SIZE	ALPHA	STEP_DOWN_COEF		
		0,98	0,99	0,998
30	0,1	0	0	0
	0,2	0	0	0
	0,8	0	70	100
60	0,1	0	0	0
	0,2	0	0	0
	0,8	60	100	100
90	0,1	0	0	0
	0,2	0	0	80
	0,8	90	100	100

Analiza przedstawionych w tabeli 11 wartości, pokazuje, że dla α = 0,1 oraz 0,2 skuteczność wynosiła 0% (z wyjątkiem konfiguracji $(m;\ \alpha;\ \delta) = (90;\ 0{,}2;\ 0{,}998)$ dla której skuteczność wyniosła 80%). Konfiguracje z α = 0,8 oraz δ = 0,998 zawsze osiągały 100% skuteczności. Podsumowując, 9 konfiguracji miało skuteczność nie mniejszą niż 50%, z czego 5 równą 100%. Większa liczba świetlików poprawiała nieco skuteczność w konfiguracjach z α = 0,8 ; δ = 0,998.

W tabeli 12 widać wyraźne wahania wartości odchylenia standardowego w zakresie od 0 do 1546,5522. Największy średni czas pracy algorytmu zarejestrowany dla konfiguracji $(m;\ \alpha;\ \delta) = (30;\ 0{,}2;\ 0{,}998)$ i wyniósł ok. 7 minut. Najkrótszy czas ma wartość mniejszą niż jedna sekunda i udało się go osiągnąć dla konfiguracji $(m;\ \alpha;\ \delta) = (60;\ 0{,}1;\ 0{,}99)$. Analizując średnią liczbę iteracji, zaobserwowano, że wartość α = 0,8 przyczynia się do wcześniejszego zakończenia pracy algorytmu. Średnia liczba iteracji dla wszystkich konfiguracji wyniosła 923. Najniższa średnia liczba iteracji równa 23 wystąpiła dla konfiguracji $(m;\ \alpha;\ \delta) = (90;\ 0{,}8;\ 0{,}99)$, a największa, równa 7210, dla $(m;\ \alpha;\ \delta) = (30;\ 0{,}2;\ 0{,}98)$.

Tabela 12. Szczegółowe wyniki pracy algorytmu dla funkcji testowej Alpine N.2

POP_SIZE	ALPHA	STEP_DOWN_COEF	MIN_RESULT	MAX_RESULT	AVG_RESULT	STD_RESULT	AVG_LAST_GEN	MIN_WORK_TIME	MAX_WORK_TIME	AVG_WORK_TIME
30	0,1	0,980	-0,8376	0,0000	-0,1216	0,2568	511,3000	00:00:00	00:00:45	00:00:10
		0,990	-0,5505	0,0000	-0,1452	0,2245	835,8000	00:00:00	00:01:01	00:00:14
		0,998	-79,6071	0,0000	-18,3683	29,1929	2737,2000	00:00:00	00:05:34	00:01:29
	0,2	0,980	-11,2615	0,0000	-1,5293	3,3651	577,2000	00:00:00	00:00:59	00:00:12
		0,990	-55,2545	0,0000	-14,8575	18,5209	1757,1000	00:00:00	00:02:15	00:01:01
		0,998	-1409,2112	0,0000	-548,5650	471,9697	7210,0000	00:00:00	00:13:29	00:06:48
	0,8	0,980	-1966,8737	0,0000	-729,3218	611,5715	1586,8000	00:00:00	00:01:48	00:01:30
		0,990	-3866,6861	0,0000	-3474,8771	1158,3328	913,3000	00:00:00	00:03:26	00:00:52
		0,998	-3866,6848	-3866,1777	-3866,4356	0,1728	68,7000	00:00:03	00:00:11	00:00:05
60	0,1	0,980	-13,4500	0,0000	-4,1285	5,5094	888,2000	00:00:00	00:05:23	00:02:09
		0,990	0,0000	0,0000	0,0000	0,0000	51,3000	00:00:00	00:00:00	0:00:00
		0,998	-163,9006	0,0000	-45,2446	69,3125	710,0000	00:00:00	00:14:45	00:04:10
	0,2	0,980	-60,8702	0,0000	-10,5985	19,2483	729,9000	00:00:00	00:07:08	00:02:33
		0,990	-286,5572	0,0000	-53,7781	90,8577	935,8000	00:00:00	00:13:04	00:04:12
		0,998	-3470,0924	0,0000	-1246,9273	1147,6284	1399,6000	00:00:00	00:13:12	00:06:13
	0,8	0,980	-3866,6879	0,0000	-2926,1359	1378,8800	569,6000	00:00:00	00:11:03	00:03:21
		0,990	-3866,6879	-3866,0770	-3866,4824	0,2188	55,3000	00:00:09	00:00:23	00:00:15
		0,998	-3866,6860	-3866,0169	-3866,4587	0,2091	40,2000	00:00:04	00:00:11	00:00:07
90	0,1	0,980	-23,1124	0,0000	-6,7946	7,6359	624,6000	00:00:00	00:05:05	00:02:42
		0,990	-26,0983	0,0000	-7,3867	11,2978	361,1000	00:00:00	00:07:01	00:02:00
		0,998	-515,0918	0,0000	-124,5188	183,1225	443,5000	00:00:00	00:09:35	00:03:40
	0,2	0,980	-109,5088	0,0000	-32,9817	34,3097	723,9000	00:00:00	00:07:33	00:04:16
		0,990	-225,6457	0,0000	-93,0133	76,1845	715,0000	00:00:00	00:08:30	00:05:35
		0,998	-3866,6876	0,0000	-3093,1043	1546,5522	309,2000	00:00:00	00:04:44	00:02:45
	0,8	0,980	-3866,6706	-378,3327	-3517,6623	1046,4432	123,2000	00:00:07	00:08:02	00:00:59
		0,990	-3866,6870	-3866,0365	-3866,4069	0,2255	23,3000	00:00:07	00:00:15	00:00:10
		0,998	-3866,6870	-3866,1155	-3866,4737	0,2219	29,4000	00:00:06	00:00:12	00:00:09

5. Podsumowanie i wnioski

W artykule przedstawiono algorytm świetlikowy oraz sprawdzono jego skuteczność dla wybranych funkcji testowych (Himmelblau, Rastrigin, Ackley, Salomon, Sphere, Alpine N. 2) dla dwudziestu siedmiu różnych konfiguracji parametrów. Zmianie zostawały poddawane trzy parametry: rozmiar populacji (m), współczynnik kroku losowego (α), współczynnik redukcji wartości α (δ).

Przeprowadzone badanie prowadzi do poniższych wniosków:

- Skuteczność algorytmu świetlikowego w znajdowaniu ekstremów globalnych badanych funkcji jest różna w zależności od funkcji. Kluczowym działaniem dla osiągnięcia wysokiej skuteczności był odpowiedni dobór parametrów algorytmu.
- Szybkość oraz możliwość znalezienia ekstremum funkcji były determinowane w głównej mierze przez parametry α i δ. Zwiększenie ich wartości zwykle poprawiało skuteczność znajdowania ekstremum. Mimo to, zastosowane konfiguracje parametrów nie zawsze były wystarczające do osiągnięcia ekstremum. Dzięki analizie takich przypadków można wyznaczyć kierunek do dalszych badań.
- Żadna z badanych konfiguracji nie charakteryzowała się 100% skutecznością dla wszystkich funkcji testowych. W większości przypadków najlepsze wyniki dawała konfiguracja $(m;\ \alpha;\ \delta) = (90;\ 0{,}8;\ 0{,}998)$.
- Zwiększanie rozmiaru populacji zwykle zwiększało skuteczność znajdowania ekstremum przez algorytm. Jednak ze względu na czas pracy algorytmu zwiększanie populacji nie zawsze jest możliwe.
- Minimum zostało znalezione w około 21% przeprowadzonych badań. Wynik ten jest spowodowany w głównej mierze słabym działaniem algorytmu przy $\alpha = 0{,}1$ oraz $\delta = 0{,}98$. Również wpłynęły na to nałożone limity na liczbę wykonywanych iteracji.
- Zaimplementowany algorytm radził sobie najlepiej w znajdowaniu ekstremum funkcji Rastrigin. Na 270 uruchomień ekstremum zostało znalezione 127 razy, a ponadto 100% skuteczność została uzyskana w 9 konfiguracjach na 27. Ekstrema globalne udało się osiągnąć również dla funkcji Himmelblau, Sphere i Alpine N. 2 – w przypadku dwóch ostatnich niektóre konfiguracje

parametrów algorytmu dawały 100% skuteczność znajdowania ekstremum. Najgorsze wyniki zanotowano dla funkcji Salomon i Ackley, gdzie minimum nie zostało znalezione ani razu. W przypadku funkcji Salomon, słabe wyniki można uzasadnić dużym zakresem wartości zmiennych, przez co algorytm mógł wpadać w pułapkę lokalnego minimum. Natomiast dla funkcji Ackley powodem mogła być charakterystyka funkcji (tzn. liczba i położenie ekstremów oraz ich otoczenie).

– Skuteczność algorytmu zależy w dużej mierze od charakterystyki funkcji oraz liczby zmiennych. Pierwszy wymieniony przypadek dobrze obrazuje funkcja Himmelblau, w przypadku której algorytm lepiej radził sobie przy małej wartości współczynnika α, gdyż otoczenie ekstremów ma wartość zbliżoną do optymalnej. Przykładem dla drugiego wymienionego przypadku może być funkcja Sphere, która była badana dla 8 zmiennych. Minimum dla tej z pozoru mało skomplikowanej do optymalizacji funkcji zostało znalezione tylko 47 razy na 270.

Literatura

[1] Dorigo, M., Maniezzo, V., Colorni, A. (1996). Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics* B(1), s. 29-41.

[2] Dorigo, M., Gambardella, L. M. (1997). Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* 1(1), 53-66.

[3] Kennedy, J., Eberhart, R. (1995). Particle swarm optimization. *Proceedings of the 911 1995 IEEE International Conference on Neural Networks* 4, s. 1942-1948.

[4] Passino, K.M. (2002). Biomimicry of Bacterial Foraging for distributed optimization and control. *IEEE Control Systems Magazine*, s. 52-68, June, 2002.

[5] Passino, K. M. (2010). Bacterial Foraging Optimization. *International Journal of Swarm Intelligence Research* 1(1).

[6] Yang, X-S. (2008). Firefly algorithm. *Nature-Inspired Metaheuristic Algorithms* 20, s. 79-90.

[7] Yang X-S. (2009) Firefly Algorithms for Multimodal Optimization. [w]: Watanabe O., Zeugmann T. (ed) *Stochastic Algorithms: Foundations and Applications. SAGA 2009. Lecture Notes in Computer Science* 5792. Springer, Berlin.

[8] Yang, X-S. (2010). Firefly Algorithm, Stochastic Test Functions and Design Optimization. *Int. J. Bio-Inspired Computation* 2 (2), s. 78-84.

[9] Yang, X-S., He, X. (2013). Firefly Algorithm: Recent Advances and Applications, *International Journal of Swarm Intelligence* 1, s. 36-50.

[10] Dey, N. (2020). *Applications of Firefly Algorithm and its Variants; Case Studies and New Developments*, Springer.

[11] Subotic, M., Misic, I., Tuba, M. (2012). An Object-Oriented Implementation of Firefly Algorithm for Continous Unconstrained Optimization Problems. *Proceedings of the 6th WSEAS international conference on Computer Engineering and Applications*.

[12] Wang, C., Chu, X. (2019). An Improved Firefly Algorithm with Specific Probability and Its Engineering Application. *IEEE Access* 7, s. 57424-57439.

[13] *BenchmarkFcns Toolbox* [on-line], http://benchmarkfcns.xyz [18-04-2021].

Using firefly algorithm for global optimization of mathematical test functions

Abstract

The paper presents one of the newest swarm intelligence methods, namely firefly algorithm proposed by Xin-She Yang in 2008. The analysis of the performance of the algorithm is carried out and the influence of the algorithm parameters settings on the quality of the solutions is examined using nonlinear single and multi-modal mathematical test functions.

Keywords: *Swarm intelligence, Firefly algorithm, Optimization*