POLITECHNIKA GDAŃSKA
WYDZIAŁ FIZYKI TECHNICZNEJ I MATEMATYKI STOSOWANEJ

Imię i nazwisko studenta: Michał Platta
Nr albumu: 155836
Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Fizyka Techniczna
Specjalność: Informatyka stosowana

PROJEKT DYPLOMOWY INŻYNIERSKI

Tytuł projektu w języku polskim: Algorytm optymalizacji globalnej bazujący na jednym z podstawowych praw fizyki i jego zastosowania

Tytuł projektu w języku angielskim: Global optimization algorithm based on the one of fundamental laws of physics and its applications

Potwierdzenie przyjęcia projektu	
Opiekun projektu *podpis*	Kierownik Katedry/Zakładu (pozostawić właściwe) *podpis*
dr hab. Leszek Kułak	

Data oddania projektu do dziekanatu:

POLITECHNIKA GDAŃSKA
WYDZIAŁ FIZYKI TECHNICZNEJ I MATEMATYKI STOSOWANEJ

OŚWIADCZENIE

Imię i nazwisko: Michał Platta
Data i miejsce urodzenia: 11.08.1994, Gdańsk
Nr albumu: 155836
Wydział: Wydział Fizyki Technicznej i Matematyki Stosowanej
Kierunek: fizyka techniczna
Poziom studiów: pierwszy
Forma studiów: stacjonarne

Ja, niżej podpisany(a), wyrażam zgodę/nie wyrażam zgody* na korzystanie z mojej pracy dyplomowej zatytułowanej: Algorytm optymalizacji globalnej bazujący na jednym z podstawowych praw fizyki i jego zastosowania
do celów naukowych lub dydaktycznych.[1]

Gdańsk, dnia ....
podpis studenta

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. z 2016 r., poz. 666 z późn. zm.) i konsekwencji dyscyplinarnych określonych w ustawie Prawo o szkolnictwie wyższym (Dz. U. z 2012 r., poz. 572 z późn. zm.),[2] a także odpowiedzialności cywilno-prawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

Niniejsza(y) praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.

Wszystkie informacje umieszczone w ww. pracy dyplomowej, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

Potwierdzam zgodność niniejszej wersji pracy dyplomowej z załączoną wersją elektroniczną.

Gdańsk, dnia ....
podpis studenta

Upoważniam Politechnikę Gdańską do umieszczenia ww. pracy dyplomowej w wersji elektronicznej w otwartym, cyfrowym repozytorium instytucjonalnym Politechniki Gdańskiej oraz poddawania jej procesom weryfikacji i ochrony przed przywłaszczaniem jej autorstwa.

Gdańsk, dnia ....
podpis studenta

*) niepotrzebne skreślić

---

[1] Zarządzenie Rektora Politechniki Gdańskiej nr 34/2009 z 9 listopada 2009 r., załącznik nr 8 do instrukcji archiwalnej PG.

[2] Ustawa z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym:
Art. 214 ustęp 4. W razie podejrzenia popełnienia przez studenta czynu podlegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzego utworu rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego.
Art. 214 ustęp 6. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 4, rektor wstrzymuje postępowanie o nadanie tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o popełnieniu przestępstwa.

STRESZCZENIE

Metaheurystyczne algorytmy optymalizacji są powszechnie wykorzystywane w wielu działach nauki i techniki. Z powodu korzyści jakie mogą przynieść, popyt na nie jak i sama ich ilość wciąż rośnie. Jednakże sprawia to iż źródeł inspiracji dla nowych algorytmów jest coraz mniej. W niniejszej pracy opisano aplikację komputerową służąca do przeprowadzania optymalizacji globalnej funkcji dwóch zmiennych rzeczywistych algorytmami Gravitational Search Algorithm (GSA) i Charged System Search (CSS). Zaproponowano również udoskonalenie algorytmu GSA. Efekt tej modyfikacji porównano z podstawowym algorytmem, a uzyskane wyniki potwierdzają lepszą zbieżność i dokładność zaproponowanego algorytmu.

Słowa kluczowe: algorytmy heurystyczne, algorytmy metaheurystyczne, CSS, GSA, Gravitational Search Algorithm, Charged System Search, optymalizacja globalna

Dziedzina nauki i techniki, zgodnie z wymogami OECD: Nauki o komputerach i informatyka; Nauka o komputerach, informatyka i bioinformatyka

ABSTRACT

Metaheuristic optimization algorithms are commonly used in many science and technology departments. The demand and number of different algorithms still increases because of many benefits they provide. However, it makes harder and harder to come up with ideas for new strategies. This work describes the computer application that can be used to perform global optimization of functions of two real variables by Gravitational Search Algorithm (GSA) and Charged System Search (CSS). Additionally, it contains a GSA improvement. The effect of modification was compared with the basic algorithm. Obtained results confirm better convergence and precision of proposed algorithm.

Keywords: heuristic algorithms, metaheuristic algorithms, CSS, GSA, Gravitational Search Algorithm, Charged System Search, global optimization

SPIS TREŚCI

WYKAZ WAŻNIEJSZYCH POJĘĆ

Algorytm	-	sposób prowadzący do rozwiązania problemu
Funkcja celu	-	funkcja dla której szukane jest optymalne rozwiązanie
Eksploracja	-	przeszukiwanie całego wyznaczonego obszaru funkcji w poszukiwaniu potencjalnych ekstremów globalnych
Eksploatacja	-	przeszukiwanie obszaru funkcji na którym istnieje podejrzenie znalezienia ekstremum globalnego
Ekstremum	-	minimum lub maksimum funkcji na danym przedziale
Dopasowanie / Przystosowanie	-	wartość agenta na funkcji celu w odniesieniu do poszukiwanego ekstremum
Metaheurystyka	-	ogólny algorytm do rozwiązywania problemów obliczeniowych
Heurystyka	-	metoda znajdowania rozwiązań, dla której nie ma pewności znalezienia rozwiązania optymalnego a nawet prawidłowego
Klasa	-	abstrakcyjny typ danych
Metoda	-	podprogram składowy / funkcja klasy
Pole	-	pojedyncza zmienna stanowiąca fragment klasy
JSON	-	lekki format wymiany danych komputerowych

WSTĘP

Algorytmy optymalizacji globalnej stosowane są w wielu różnych dziedzinach, od projektowania mostów i samolotów, po lakierowanie samochodów. Pomagają one w znalezieniu rozwiązań optymalnych, takich, aby przy jak najmniejszych kosztach i zużyciu materiałów osiągnąć jak najlepsze parametry produktów. Przy produkcji masowej nawet jeden gram zaoszczędzonego materiału może znacznie wpłynąć na zysk końcowy firmy i cenę jej produktów. Sprawia to iż algorytmy potrafiące znaleźć takie optimum są wciąż rozwijane, a także poszukuje się nowych źródeł inspiracji do ich tworzenia. Jednym z takich źródeł jest matematyka, przykładowym algorytmem bazującym na niej jest metoda Newtona lub metoda gradientu prostego. Jednakże takie rozwiązania mimo udowodnionej zbieżności do ekstremum posiadają wady, między innymi mają dużą złożoność obliczeniową. Innym źródłem pomysłów jest sama Natura, jak na przykład, zachowywanie się latających robaczków świętojańskich (algorytm świetlikowy) lub rozprzestrzenianie się grzybni w sposób do dzisiaj niezrozumiały. W rozwiązaniach bazujących na powyższych zjawiskach problemem jest wyznaczenie wzoru na taki algorytm. Lecz Natura ma do zaoferowania również fizykę, dla której większość zjawisk jest przecież znana i dobrze opisana wzorami. Znając wzory na przyciąganie grawitacyjne lub elektrostatyczne obiektów można je wykorzystać w algorytmie tak, by rozwiązania lepsze przyciągały mocniej od tych słabszych. Takimi algorytmami są przykładowo: przeszukiwanie systemem naładowanych cząstek lub algorytm grawitacyjnych poszukiwań.

W niniejszej pracy podjęto się zmodyfikowania modyfikacji algorytmu Gravitational Search Algorithm (w skrócie GSA). Ta modyfikacja jest dołączeniem do GSA metodologii algorytmu Charged System Search (w skrócie CSS). Tak powstały algorytm został nazwany Gravitational Search Algorithm Plus (w skrócie GSA+). Ponadto podjęto się stworzenia aplikacji komputerowej pozwalającej na optymalizację globalną funkcji dwóch zmiennych rzeczywistych przy pomocy wyżej wymienionych algorytmów oraz na wizualizację ich działania.

Prezentowana praca składa się z części teoretycznej i części praktycznej. Rozdział pierwszy przedstawia podstawy teoretyczne opracowanych algorytmów. Wzory na jakich one bazują, a także metodę ich działania. Kolejny rozdział zawiera opis technologii wykorzystanej do napisania algorytmów, jak i aplikacji komputerowej. Opisano również sposób instalacji i uruchomienia aplikacji oraz wszystkie jej funkcjonalności i istotne fragmenty kodu. Kolejne dwa rozdziały, trzeci i czwarty, prezentują wyniki testów nowego algorytmu w porównaniu do algorytmów, na których bazuje oraz wnioski wyciągnięte na ich podstawie. Część praktyczna zawiera programową realizację projektu.

CEL PRACY

Celem pracy jest stworzenie aplikacji komputerowej do optymalizacji globalnej przy pomocy algorytmów metaheurystycznych inspirowanych podstawowymi prawami fizyki oraz udoskonalenie jednego z nich. Wybranymi algorytmami są Gravitational Search Algorithm oraz Charged System Search. Aby dobrze wykonać projekt, niezbędne było zapoznanie się podstawami działania tych algorytmów oraz ich zastosowaniem.

1. PODSTAWY TEORETYCZNE

1.1. Charged System Search

Autorzy: A. Kaveh, S. Talatahari

Rok publikacji: 2010

Tytuł orginalny publikacji: *„A novel heuristic optimization method: charged system search”*

1.1.1. Podstawy fizyczne

Na podstawie Coulomba wiemy, że siła oddziaływań między dwoma punktowymi cząstkami F_{ij} wynosi:

$$F_{ij} = k_e \frac{q_i q_j}{r_{ij}^2}, \tag{1.1}$$

gdzie k_e jest stałą Coulomba; r_{ij} jest odległością między tymi dwiema cząstkami; q_i i q_j są ładunkami w nich zwartymi.

Rozważmy izolowaną kulę o promieniu *a*, jednorodnie naładowaną ładunkiem dodatnim q_i. Natężenie pola elektrycznego E_{ij} w punkcie poza kulą wynosi:

$$E_{ij} = k_e \frac{q_i}{r_{ij}^2}. \tag{1.2}$$

Natężenie pola elektrycznego wewnątrz kuli można wyznaczyć z prawa Gaussa:

$$E_{ij} = k_e \frac{q_i}{a^3} r_{ij}, \tag{1.3}$$

gdzie: *a* jest promieniem tej że kuli.

Korzystając ze wzorów (1.2) i (1.3) ogólny wzór na natężenie pola elektryczne E_{ij} przyjmie postać:

$$E_{ij} = \begin{cases} k_e \frac{q_i}{a^3} r_{ij} & gdy\ r_{ij} < a, \\ k_e \frac{q_i}{r_{ij}^2} & gdy\ r_{ij} > a. \end{cases} \tag{1.4}$$

Wartość i kierunek siły działającej na ładunek q_i w punkcie $\boldsymbol{r}_i$ wywołanej ładunkiem q_j w punkcie $\boldsymbol{r}_j$, można wyrazić w następującym wzorem:

$$\boldsymbol{F}_{ij} = E_{ij} q_j \frac{\boldsymbol{r}_i - \boldsymbol{r}_j}{\|\boldsymbol{r}_i - \boldsymbol{r}_j\|}. \tag{1.5}$$

Uwzględniając wzory (1.4) i (1.5) można wyprowadzić wzór na siłę wypadkowa oddziaływująca na daną cząstkę:

$$\boldsymbol{F}_j = k_e q_j \sum_{i, i \neq j} \left(\frac{q_i}{a^3} r_{ij} \cdot i_1 + \frac{q_i}{r_{ij}^2} \cdot i_2 \right) \frac{\boldsymbol{r}_i - \boldsymbol{r}_j}{\|\boldsymbol{r}_i - \boldsymbol{r}_j\|} \tag{1.6}$$

$$gdzie \begin{cases} i_1 = 1,\ i_2 = 0 \Leftrightarrow r_{ij} < a, \\ i_1 = 0,\ i_2 = 1 \Leftrightarrow r_{ij} \geq a. \end{cases}$$

Zakładamy, że ruch cząstek będą opisywały prawa Newtona, zatem cząstki będą poruszały się ruchem jednostajnie przyspieszonym. Przemieszczenie cząstki opisuje wzór:

$$\boldsymbol{r_{new}} = \frac{1}{2}\boldsymbol{a} \cdot \Delta t^2 + \boldsymbol{v_{old}} \cdot \Delta t + \boldsymbol{r_{old}}. \tag{1.7}$$

gdzie: $\boldsymbol{r_{new}}$ jest nową pozycją cząstki a $\boldsymbol{r_{old}}$ starą pozycją; Δt jest zmianą czasu w trakcie przemieszczenia; $\boldsymbol{v_{old}}$ to prędkość cząstki równa zmianie jej pozycji w poprzedniej chwili czasu; $\boldsymbol{a}$ jest przyspieszeniem.

Z drugiej zasady dynamiki Newtona wzór na siłę jest następujący:

$$\boldsymbol{F} = m \cdot \boldsymbol{a}. \tag{1.8}$$

gdzie *m* jest masą obiektu; ***F*** – wypadkową siłą działającą na ten obiekt.

Podstawiając do wzoru (1.7) przyspieszenie obliczone ze wzoru (1.8), otrzymujemy:

$$\boldsymbol{r_{new}} = \frac{1}{2}\frac{\boldsymbol{F}}{m} \cdot \Delta t^2 + \boldsymbol{v_{old}} \cdot \Delta t + \boldsymbol{r_{old}}. \tag{1.9}$$

1.1.2. Działanie algorytmu

Niech każde rozwiązanie $\boldsymbol{X_j}$ reprezentuje naładowaną cząstkę (agenta), każda z nich może wpływać na pole elektryczne innych cząstek. Każdy agent jest różnoimienny ładunkiem w stosunku do innych agentów. Wartość ładunku jest definiowana w oparciu o wartość funkcji celu. Wielkość ładunku dla i-tego agenta opisana jest wzorem:

$$q_i = \frac{\mathrm{f}(\mathbf{X_i}) - \mathrm{f_{worst}}}{\mathrm{f_{best}} - \mathrm{f_{worst}}}, \tag{1.10}$$

gdzie *f()* jest funkcją celu; f_{worst} jest najgorszą znalezioną wartością funkcji celu przez wszystkich agentów; f_{best} najlepszą znalezioną wartością funkcji celu przez wszystkich agentów.

Wzór na odległość agentów od siebie został zaproponowany metodą prób i błędów zastosowaną do funkcji testowych, tak by również uwzględniał położenie ogólnie najlepszego agenta, ma on postać:

$$r_{ij} = \frac{\left|\mathbf{X_i} - \mathbf{X_j}\right|}{\left|\frac{1}{2}(\mathbf{X_i} - \mathbf{X_j}) - \mathbf{X_{best}}\right| + \varepsilon}, \tag{1.11}$$

gdzie: $\mathbf{X_i}$ i $\mathbf{X_j}$ to pozycje i-tego i j-tego agenta; $\mathbf{X_{best}}$ jest położeniem najlepszego agenta; ε to mała liczba dodatnia pozwalająca uniknąć dzielenia przez zero.

Początkowa pozycja każdego agenta zostaje wylosowana z przedziału na którym ma zostać uruchomiony algorytm. Przyjmujemy równe zeru początkowe wartości prędkości i przyspieszenia.

Są trzy możliwe warunki związane z rodzajami siły przyciągania, ale w tej aplikacji został wykorzystany tylko jeden. Agent może wpływać na innych wyłącznie jeżeli wielkość jego ładunku jest lepsza od innych agentów. Siła oddziaływująca na j-tego agenta jest zatem równa:

$$\boldsymbol{F}_j = q_j \sum_{i,i\neq j} \left(\frac{q_i}{a^3} r_{ij} \cdot i_1 + \frac{q_i}{r_{ij}^2} \cdot i_2 \right) p_{ij} \left(\boldsymbol{X_i} - \boldsymbol{X_j} \right), \quad gdzie \begin{cases} j = 1,2,..,N, \\ i_1 = 1,\ i_2 = 0 \Leftrightarrow r_{ij} < a, \\ i_1 = 0,\ i_2 = 1 \Leftrightarrow r_{ij} \geq a. \end{cases} \tag{1.12}$$

gdzie: p_{ij} jest zmienną przyjmująca wartość zero, jeżeli agent i-ty jest słabszy od j-tego, w przeciwnym wypadku przyjmuje wartość jeden; q_j i q_i to ładunki wyznaczone wzorem (1.10); r_{ij} to odległość ze wzoru (1.11). Stała Coulomba zostaje pominięta.

Każdy agent jest traktowany jako jednorodnie naładowana kula o promieniu *a.* Promień ten jest zależny od wielkości przeszukiwanej przestrzeni, jego wartość została oszacowana metodą prób i błędów na podstawie funkcji testowych:

$$a = 0.1 \cdot \max\left(\{\left|x_{i,max} - x_{i,min}\right| \ i = 1,2,\dots,n\}\right), \tag{1.13}$$

gdzie: *n* jest ilością wymiarów; $x_{i,max}$ i $x_{i,min}$ to wartości maksymalne i minimalne w i-tym wymiarze. Kiedy większość agentów znajduje się w obrębie promienia kuli, algorytm wchodzi w tryb eksploatacji dający dokładniejsze wyniki. Im większy jest promień tym szybciej algorytm wejdzie w ten tryb, lecz istnieje również większe ryzyko wpadnięcia w ekstremum lokalne.

Modyfikując równanie (1.9) otrzymujemy wzór na nową pozycję j-tego agenta, znając nową pozycję, można obliczyć przemieszczenie a znając przemieszczenie nową prędkość.

$$\boldsymbol{X_{j,new}} = rand_{j1} \cdot k_a \cdot \frac{\boldsymbol{F_j}}{m_j} \cdot \Delta \mathrm{t}^2 + rand_{j2} \cdot k_v \cdot \boldsymbol{V_{j,old}} \cdot \Delta \mathrm{t} + \boldsymbol{X_{j,old}}, \tag{1.14}$$

$$\boldsymbol{V_{j,new}} = \frac{\boldsymbol{X_{j,new}} - \boldsymbol{X_{j,old}}}{\Delta t}, \tag{1.15}$$

gdzie: $\mathbf{X}_{\mathrm{j,new}}$ i $\mathbf{X}_{\mathrm{j,old}}$ to nowa i stara pozycja j-tego agenta; k_a jest parametrem związanym z siłami przyciągania; k_v jest parametrem związanym ze poprzednią prędkością agenta; $rand_{j1}$ i $rand_{j2}$ to liczby pseudolosowe z przedziału (0,1); Δt to jeden krok iteracji który zawsze jest równy 1; m_j jest masą cząstki; $\boldsymbol{V_{j,new}}$ jest nową prędkością agenta. Jako iż algorytm jest bezwymiarowy, masę przyjmuje się za równą ładunkowi danej cząstki.

By z czasem działania algorytmu zwiększyć eksploatację i zmniejszyć eksplorację, parametr k_a powinien się zwiększać, natomiast i k_v zmniejszać. Parametry te, spełniające powyższe wymagania, można zdefiniować następująco:

$$k_{\boldsymbol{v}} = 0.5\left(1 - \frac{\mathrm{iter}}{\mathrm{iter}_{\max}}\right), \ k_{\boldsymbol{a}} = 0.5\left(1 + \frac{\mathrm{iter}}{\mathrm{iter}_{\max}}\right), \tag{1.16}$$

gdzie: *iter* jest aktualną iteracją, a $\mathrm{iter}_{\max}$ maksymalną ilością iteracji.

Po połączeniu wzorów (1.12), (1.14), (1.15) i (1.16), wzór na nową pozycję i prędkość j-tego agenta przyjmuje postać:

$$\boldsymbol{X_{j,new}} = 0.5 rand_{j1} \cdot \left(1 + \frac{\text{iter}}{\text{iter}_{\max}}\right) \cdot \sum_{i,i \neq j} \left(\frac{q_i}{a^3} r_{ij} \cdot i_1 + \frac{q_i}{r_{ij}^2} \cdot i_2\right) p_{ij} \left(\boldsymbol{X_i} - \boldsymbol{X_j}\right) + 0.5 rand_{j2} \cdot \left(1 - \frac{\text{iter}}{\text{iter}_{\max}}\right) \cdot \boldsymbol{V_{j,old}} + \boldsymbol{X_{j,old}}, \tag{1.17}$$

$$\boldsymbol{V_{j,new}} = \boldsymbol{X_{j,new}} - \boldsymbol{X_{j,old}}. \tag{1.18}$$

gdzie: *a* jest promieniem obliczonym ze wzoru (1.13); $\boldsymbol{V_{j,old}}$ jest poprzednią prędkością j-tego agenta. Wzory (1.17) i (1.18) są wzorami końcowymi, stosowanymi w danym algorytmie. Zaimplementowany algorytm przystosowany jest wyłącznie do funkcji jednej i dwóch zmiennych.

1.2. Gravitational Search Algorithm

Autorzy: Esmat Rashedi, Hossein Nezamabadi-pour, Saeid Saryazdi
Rok publikacji: 2009
Tytuł oryginalny publikacji: „*GSA: A Gravitational Search Algorithm"*

Algorytm bazuje na podstawowym prawie powszechnego ciążenia Newtona. Gdzie każdy obiekt we wszechświecie przyciąga każdy inny obiekt z siłą, która jest równa:

$$F = G \frac{M_1 M_1}{R^2} \tag{1.19}$$

gdzie: *G* jest stałą grawitacyjną; M_1 i M_2 to masy oddziaływujących ze sobą obiektów (agentów); *R* jest odległością między tymi obiektami. Każdemu agentowi w algorytmie przyporządkowujemy masę zależną od wartości funkcji celu oraz odpowiednie położenie w dziedzinie tej funkcji. Wprowadza się jednak pewne zmiany do wzoru (1.19). Przede wszystkim stała *G* zostaje uzależniona od wieku Wszechświata, odpowiadającego aktualnej iteracji algorytmu. Przedstawia się ją wzorem:

$$G(t) = G_0 e^{-\alpha \frac{t}{T}}, \tag{1.20}$$

gdzie: G_0 jest stałą grawitacyjną w pierwszym momencie czasu; α stała wpływająca na prędkość zmniejszania się stałej *G*, zazwyczaj przyjmowana jako równa 20; *t* – aktualna iteracja; *T* – maksymalna ilość iteracji. Wyróżnia się również trzy rodzaje mas:

- aktywna masa grawitacyjna M_a, jest miarą siły pola grawitacyjnego obiektu,
- bierna masa grawitacyjna M_p, miara siły interakcji obiektu z polem grawitacyjnym,
- masa bezwładna M_i, miara siły przeciwstawianie się obiektu zmianom swojego położenia.

Korzystając ze wzoru (1.20), uzyskujemy równanie na siłę oddziaływującą na agenta i-tego przez agenta j-tego:

$$\boldsymbol{F}_{ij}(t) = G(t) \frac{M_{pi}(t) M_{aj}(t)}{R_{ij} + \varepsilon} (\boldsymbol{X_j}(t) - \boldsymbol{X_i}(t)), \tag{1.21}$$

gdzie: ε to mała stała dodatnia liczba zapobiegająca dzieleniu przez zero; $\boldsymbol{X_i}(t)$ i $\boldsymbol{X_j}(t)$ to położenie agenta i-tego i j-tego w czasie iteracji *t,* pozwala to na zachowanie wektora na którym oddziałuję dana siła. Podczas tworzenia algorytmu, twórcy zauważyli iż lepsze wyniki daje wzór bez podnoszenia odległości do kwadratu.

Uwzględniając wzór (1.21) i sortując agentów od najgorszego do najlepszego można policzyć siłę wypadkową działającą na i-tego agenta:

$$\boldsymbol{F}_i(t) = \sum_{j \in Kbest, j \neq i}^{N} rand_j \boldsymbol{F}_{ij}(t), \tag{1.22}$$

gdzie: *N* to ilość agentów; K_{best} jest zbiorem najlepszych agentów; $rand_j$ to zmienna losowa z przedziału (0,1).

Znając siły oddziaływujące na agenta, wzór (1.22), można wyznaczyć jego przyspieszenia (z drugiej zasady dynamiki Newtona):

$$\boldsymbol{a}_i(t) = \frac{\boldsymbol{F}_i(t)}{M_{ii}(t)}, \tag{1.23}$$

i dodanie go do poprzedniej jej wartości:

$$\boldsymbol{v}_i(t+1) = rand_i \cdot \boldsymbol{v}_i(t) + \boldsymbol{a}_i(t), \tag{1.24}$$

gdzie: $rand_i$ to zmienna losowa z przedziału (0,1).

Wyliczona prędkość (1.24) umożliwia wyliczenie nowej pozycji agenta wzorem (1.25).

$$\boldsymbol{X}_i(t+1) = \boldsymbol{X}_i(t) + \boldsymbol{v}_i(t+1). \tag{1.25}$$

Masę każdemu agentowi przyporządkowuje się w oparciu o wartość funkcji celu na poddstawie następujących wzorów:

$$m_i(t) = \frac{f(\boldsymbol{X}_i(t)) - f_{worst}(t)}{f_{best}(t) - f_{worst}(t)}, \tag{1.26}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}, \tag{1.27}$$

gdzie: m_i jest tymczasową masą bezwładności; $f(\boldsymbol{X}_i(t))$ to wartość funkcji celu w punkcie położenia agenta i-tego w iteracji *t;* $f_{best}(t)$ i $f_{worst}(t)$ są najlepszą i najgorszą wartością funkcji celu znalezioną przez algorytm w iteracji *t*.

W wersji podstawowej w algorytmie przyjmuje się, że wszystkie trzy rodzaje masy są równe masie ze wzoru (1.27), $M_{ai} = M_{pi} = M_{ii} = M_i$ dla $i = 1,2,\ldots,N$.

W zaimplementowany algorytmie przyjmuje się wartości $G_0 = 100,\ \alpha = 20$. Algorytm został przystosowany wyłącznie do funkcji jednej i dwóch zmiennych rzeczywistych. Liczebność zbioru najlepszych agentów K_{best} zmniejsza się liniowo w czasie.

1.3. Gravitational Search Algorithm Plus

Algorytm ten został stworzony z połączenia podstawowego algorytmu Gravitational Search Algoritchm (GSA) oraz Charged System Search (CSS). Tak jak algorytm GSA wykorzystuje on prawo powszechnego ciążenia, jednak stała grawitacyjna w pierwszym momencie czasu jest zależna od wielkości wszechświata.

$$G(0) = \frac{\max\left(\left|x_{i,max} - x_{i,min}\right|\right)}{2} \tag{1.28}$$

gdzie: $x_{i,max}$ i $x_{i,min}$ to wartości maksymalne i minimalne w i-tym wymiarze. Wzór (1.28) został dobrany metodą prób i błędów na podstawie obserwacji działania algorytmu GSA.

Uwzględniony został również promień agentów, wyliczany według równania (1.13). Tak jak w algorytmie CSS agenci będący w odległości mniejszej od niego podlegają innemu równaniu. Siła oddziaływująca na agenta i-tego przez agenta j-tego wygląda następująco:

$$\begin{gathered} \boldsymbol{F}_{ij}(t) = \left(\frac{M_{pj}(t)}{a^3} R_{ij} \cdot i_1 + \frac{M_{pj}(t) M_{ai}(t)}{R_{ij}} \cdot i_2\right), \\ gdzie \begin{cases} i_1 = 1,\ i_2 = 0 \Leftrightarrow R_{ij} < a, \\ i_1 = 0,\ i_2 = 1 \Leftrightarrow R_{ij} \geq a, \end{cases} \end{gathered} \tag{1.29}$$

gdzie: *a* jest promieniem agentów; M_{pj} to bierna masa grawitacyjna j-tego agenta; M_{ai} jest to aktywna masą agenta i-tego; R_{ij} to odległość między agentami; i1 i i2 są parametrami przełączającymi między równaniami.

Algorytm pomija zbiór najlepszych agentów, pozwalając na oddziaływanie agenta o słabszym przystosowaniu na agenta o lepszym przystosowaniu. Jednakże siła takiego oddziaływania jest sto tysięcy razy słabsza. Wartość ta została wyznaczona metodą prób i błędów. Zatem:

$$\begin{gathered} \boldsymbol{F}_i(t) = \sum_{j,j \neq i}^{N} rand_j \boldsymbol{F}_{ij}(t) p_{ij}, \\ gdzie \begin{cases} p_{ij} = 0.00001 \Leftrightarrow f(\boldsymbol{X}_j) < f(\boldsymbol{X}_i), \\ p_{ij} = 1 \Leftrightarrow f\left(\boldsymbol{X}_j\right) > f(\boldsymbol{X}_i), \end{cases} \end{gathered} \tag{1.30}$$

gdzie: $\boldsymbol{F}_{ij}(t)$ jest siłą oddziaływania między dwoma obiektami w iteracji *t* ze wzoru (1.29); $rand_j$ to zmienna losowa z przedziału (0,1); *N* jest ilością agentów.

Masa bezwładna jest wyznaczana według wzorów (1.26) oraz (1.27), ponadto zakładamy że wszystkie trzy masy są sobie równe, $M_{ai} = M_{pi} = M_{ii} = M_i$ dla $i = 1,2,\ldots,N$.

Przyspieszenie obiektu jest obliczane na podstawie wzoru (1.23), gdzie siłą oddziaływującą na agenta jest siłą ze wzoru (1.30). Początkowe położenie agentów jest losowe, ich prędkość jest równa zeru. Nowe wartości są wyliczane na podstawie równań (1.24) oraz (1.25).

2. APLIKACJA KOMPUTEROWA

2.1. Technologie wykorzystane do realizacji projektu

Główne algorytmy aplikacji zostały napisane w języku programowania JavaScript z uwagi na jego prostotę oraz jego wciąż rosnącą popularność i dostępność licznych bibliotek zewnętrznych wprowadzających przydatne funkcjonalności. Interfejs użytkownika został wykonany w języku HTML 5.0 i CSS 3.0.

Konwertowanie funkcji z ciągu znaków jest możliwe dzięki wykorzystaniu biblioteki Math.js, natomiast jej graficzne przedstawienie zostało wykonane za pomocą HTML5 Canvas, który jest elementem języka HTML. Do integracja interfejsu z algorytmami oraz jego dynamicznych zmian została wykorzystana biblioteka jQuery.

2.2. Instalacja i uruchomienie aplikacji

Wymagania instalacyjne: dowolny system operacyjny Windows lub Linux oraz około 20MB wolnego miejsca na dysku. By zainstalować aplikację wystarczy przenieść wszystkie pliki i katalogi do wybranego katalogu, jeżeli jest to instalacja na serwerze, to wszystkie pliki należy przenieść do udostępnianego katalogu.

Wymagania systemowe do uruchomienia aplikacji są zależne od sposobu jej instalacji, związane jest to z użyciem w aplikacji dedykowanych wątków roboczych. Wątki te podlegają polityce tego samego pochodzenia (same-origin policy) w modelu bezpieczeństwa aplikacji internetowych, która jest różnie przestrzegana przez twórców przeglądarek internetowych.

2.2.1. Uruchomienie aplikacji zainstalowanej na serwerze

Wymagania do uruchomienia aplikacji: dowolny system operacyjny z zainstalowaną graficzną przeglądarką internetową Chrome (od wersji 60), Firefox (od wersji 57) lub Opera (od wersji 49).

By uruchomić aplikację, w pasku adresu jednej z powyżej wymienionych przeglądarek, należy wpisać adres serwera, na którym została zainstalowana aplikacja.

2.2.2. Uruchamianie aplikacji niezainstalowanej na serwerze

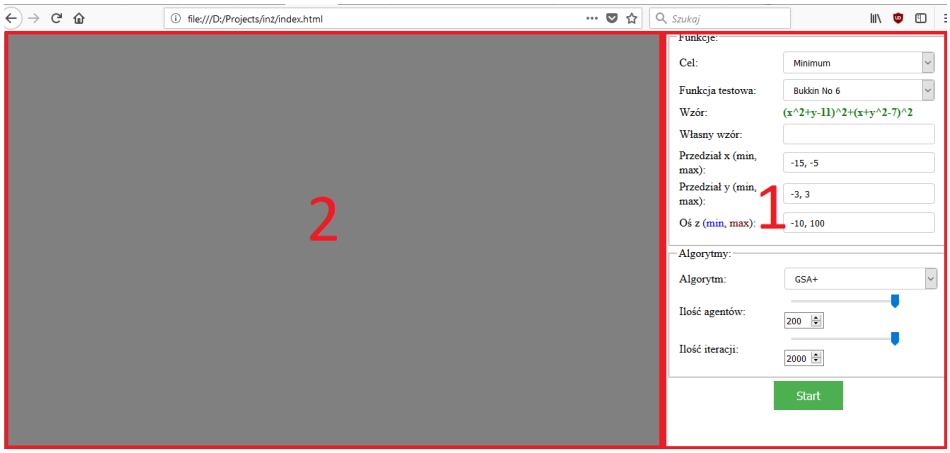
Wymagania do uruchomienia aplikacji: dowolny system operacyjny z zainstalowaną graficzną przeglądarką internetową Firefox (od wersji 57).

By uruchomić aplikację, w pasku adresu przeglądarki Firefox należy wpisać ścieżkę do pliku *index.html* w katalogu, w którym znajduje się aplikacja.

Inny sposób to przejść do katalogu, w którym aplikacja została zainstalowana i klikając prawym przyciskiem myszy na pliku *index.html* wybrać opcję „Otwórz za pomocą”, a następnie wybrać przeglądarkę Firefox.

2.3. Obsługa aplikacji

Po uruchomieniu aplikacji w przeglądarce, pojawi się interfejs jak na Rys. 2.1. W jego prawej części znajdować się będzie panel sterujący aplikacją [1], natomiast w lewej części pole [2], w którym zostanie przedstawiona graficznie funkcja celu i wizualizacja algorytmu.

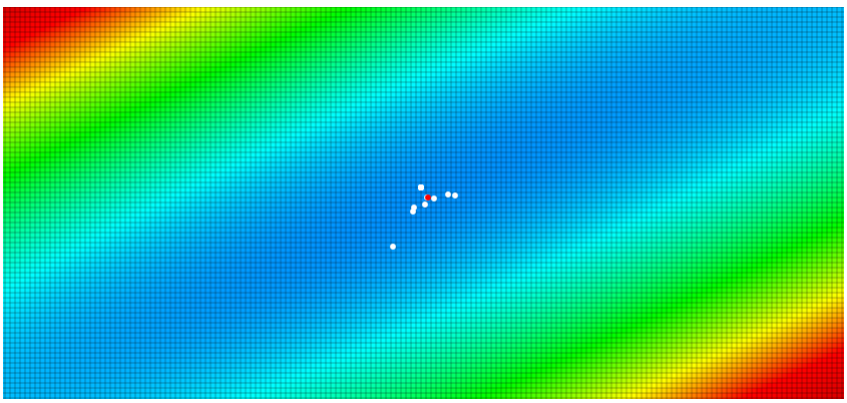


Rys. 2.1. Interfejs aplikacji

W panelu sterującym w dziale „Funkcje” należy wybrać „Cel” poszukiwań (minimum lub maksimum) oraz funkcje celu, jedną z funkcji testowych lub wpisać własną funkcję w polu „Własny wzór”. Wpisanie własnej funkcji znacznie zwiększa czas obliczeń algorytmów, jest to spowodowane wolnym działaniem biblioteki Math.js. W polu „Wzór” zostanie wyświetlony wzór funkcji testowej. W polach „Przedział x” oraz „Przedział y” wymagane jest podanie przedziału na jakim ma dana funkcja zostać przeszukiwana. Pole „Oś z” jest odpowiedzialne za interpolację kolorów w graficznym przedstawieniu funkcji. Wartości minimalnej odpowiada kolor niebieski (RGB(0, 0, 255)), a maksymalnej kolor czerwony (RGB(100, 0, 0)).

W dziale „Algorytmy” należy wybrać „Algorytm”, który ma zostać uruchomiony oraz związane z nim parametry „Ilość agentów” i „Ilość iteracji”.

Po wybraniu odpowiednich parametrów i algorytmu należy przycisnąć przycisk „Start”, po jego wciśnięciu w lewej części aplikacji zostanie wygenerowana wizualizacja działania algorytmu na graficznym przedstawieniu funkcji celu jak na Rys. 2.2.

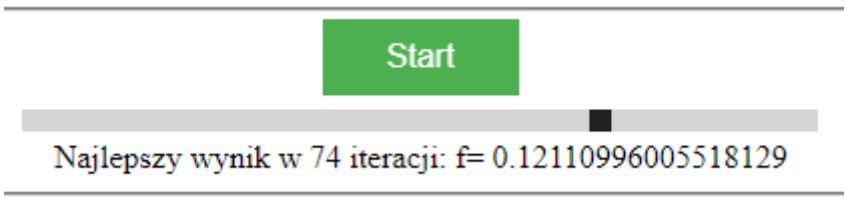


Rys. 2.2. Wizualizacja działania algorytmu

Natomiast pod przyciskiem pojawi się dział „Wyniki” z aktualnie najlepszym rozwiązaniem. Czas wygenerowania graficznego przedstawienia funkcji celu jest zależny od rozdzielczości ekranu, wielkości okna przeglądarki, w którym jest uruchomiona aplikacja oraz parametrów komputera, na którym aplikacja została uruchomiona.

W trakcie działania algorytmu wizualizacja prezentuje aktualne położenie agentów (białe punktu) w co dziesiątej iteracji oraz najlepsze aktualnie znalezione rozwiązanie (czerwony punkt). Przycisk „Start” zostaje zastąpiony przyciskiem „Stop” jego wciśnięcie powoduje bezwzględne zatrzymanie algorytmu bez uzyskania wyniku końcowego.

Po zakończeniu obliczeń w dziale „Wyniki” pojawi się czas działania algorytmu podany w milisekundach. Podany czas związany jest wyłącznie z procesem obliczania bez uwzględnienia czasu wyświetlania i czasu komunikacji między wątkami. Przycisk „Stop” zostaje na powrót zastąpiony przyciskiem „Start”, pod którym pojawi się suwak jak na Rys. 2.3. Jego przesuwanie pozwala na wyświetlenie zmian stanu algorytmu w czasie.



Rys. 2.3. Suwak

Ponowne przyciśnięcie przycisku „Start” spowoduje po raz kolejny uruchomienie algorytmu.

Wygląd interfejsu może się różnić w zależności od użytej przeglądarki internetowej. Niektóre elementy mogą zmienić się lub pojawić za wcześnie np. przycisk „Stop” lub suwak, jest to spowodowane asynchronicznym działaniem biblioteki jQuery.

2.4. Rozwiązania programowe oraz istotne fragmenty kodu

Wszystkie zaimplementowane algorytmy zostały przystosowane do szukania wyłącznie maksimów funkcji, jednakże jeżeli zostanie wybrana opcja „Minimum” algorytmy będą odwracały wyniki funkcji. Takie rozwiązanie ułatwia obliczanie przystosowania agentów bez większego nakładu pracy oraz dodatkowych funkcji.

Jako iż język JavaScript jest językiem skryptowym oraz specyfika projektu utrudnia użycie klas, większość projektu jest proceduralna. Obiekty zostały użyte do przechowywania parametrów agentów oraz wysłania ich między wątkami.

Graficzne przedstawienie funkcji zostało wykonane, po przez interpolację kolorów, z modelu przestrzeni barw HSV.

Każdy algorytm wykorzystuje własną klasę cząstek np. *ParticleCSS* dziedziczącą po klasie Agent zaprezentowanej na Rys. 2.4.

```
1  function Agent(x, y, value, valueToShow) {
2      this.x = x;
3      this.y = y;
4      this.value = value;
5      this.valueToShow = valueToShow;
6      this.velocity = {
7          x : 0.0,
8          y : 0.0
9      };
10 }
```

Rys. 2.4. Klasa Agent

Klasa Agent przetrzymuje podstawowe wartości cząstek. Takie jak położenie, prędkość i wartość funkcji celu. Pole *valueToShow* przechowuje faktyczną wartość funkcji celu natomiast *value* wartość przystosowania, czyli odwróconą jeśli szukane jest minimum.

Do komunikacji między wątkami wykorzystywana jest metoda *postMessage(data)*. Dane wysyłane są w formacie JSON.

2.4.1. Charged System Search

Pierwszym etapem po uruchomieniu algorytmu jest obliczenie promienia (Rys. 2.5) cząstek stosując wzór (1.13) zaprogramowany w sposób przedstawiony na Rys. 2.6.

```
const radius = calculateRadius(data.interval);
```

Rys. 2.5. Wywołanie funkcji liczącej promień

```
function calculateRadius(interval) {
    let radiusX = 0.1 * (interval.max.x - interval.min.x);
    let radiusY = 0.1 * (interval.max.y - interval.min.y);

    return Math.max(radiusX, radiusY);
}
```

Rys. 2.6. Implementacja programowa wzoru (1.13)

Następnie losowane są pozycje agentów i tworzone ich obiekty klasy *ParticleCSS*. Wszyscy agenci są umieszczani w tablicy *agents*. Kolejnym etapem jest wyliczanie ich ładunków w zależności od ich przystosowania. Zaprezentowane zostało to na Rys. 2.7 oraz Rys. 2.8 i jest to zaprogramowany wzór (1.10).

```
87          for (let j = 0; j < data.numberOfAgents; j++) {
88              agents[j].calculateCharge(best.value, worst.value);
89          }
```

Rys. 2.7. Wywołanie metody *calculateCharge()* na obiekcie agenta klasy *ParticleCSS*

```
7    this.calculateCharge = function(bestValue, worstValue) {
8        // check if function is constant
9        if (bestValue != worstValue)
10           this.q = ((this.value - worstValue) / (bestValue - worstValue));
11       else this.q = 0.5;
12   }
```

Rys. 2.8. Implementacja metody *calculateCharge()*

Gdzie *best.value i worst.value* są przystosowaniem najlepszego i najgorszego agenta. Natomiast funkcja *calculateCharge()* jest metodą klasy *ParticleCSS,* a *q* jest jej polem reprezentującym ładunek agenta. Jeśli przystosowanie najgorszego i najlepszego agenta jest równe, oznacza to iż funkcja w tych punktach jest stała. Dla tego ładunek wszystkich agentów ustawiany jest 0.5, pozwala to na ciągłe poruszanie się agentów w poszukiwaniu ekstremów.

Implementacja wzoru (1.12) została przedstawiona na Rys. 2.9. Wiersze 107 i 108 odpowiadają za obliczanie siły oddziaływującej na cząstkę natomiast 106 odpowiada za zmienną p_{ij}. Agent słabszy nie oddziałuje na lepszego. Natomiast wiersze 111 i 112 odpowiadają za zachowanie kierunku oddziaływania siły.

```
98           // calculate sum force acting on j-th paritcle
99           for (var k = 0; k < data.numberOfAgents; k++) {
100              if (k === j) { continue };
101
102              var distance = calculateDistance(agents[j], agents[k], best);
103              var scalarForce = 0.0;
104
105              // calculate scalar force
106              if (agents[k].value > agents[j].value) {
107                  if (distance >= radius) { scalarForce = (agents[k].q / Math.pow(distance, 2)); }
108                  else { scalarForce = (agents[k].q / Math.pow(radius, 3)) * distance; }
109              } else scalarForce = 0;
110
111              force.x += (scalarForce * (agents[k].x - agents[j].x));
112              force.y += (scalarForce * (agents[k].y - agents[j].y));
113          }
```

Rys. 2.9. Wyliczanie siły działającej na cząstkę w algorytmie CSS

Funkcja *calculateDistance()* odpowiada za wyliczenie odległości między agentami, według wzoru (1.11), przedstawiono ją na Rys. 2.10.

```
172  function calculateDistance(agent1, agent2, best) {
173      let epsilon = 0.000000001;          // small positive number to avoid singularities
174      let numerator = Math.sqrt(Math.pow((agent1.x - agent2.x), 2)
175          + Math.pow((agent1.y - agent2.y), 2));
176      let denominator = Math.sqrt(Math.pow(((agent1.x + agent2.x) * 1/2) - best.x, 2)
177          + Math.pow(((agent1.y + agent2.y) * 1/2) - best.y, 2));
178
179      return numerator / (denominator + epsilon);
180  }
```

Rys. 2.10. Funkcja obliczająca dystans między cząstkami

Nowa pozycja cząstki wyliczana ze wzoru (1.17) została zaimplementowana w sposób przedstawiony na Rys. 2.11.

```
118             // exploration (kv) and exploitation (ka) controls
119             let ka = (1 / 2) * (1 + (i / data.iterations));
120             let kv = (1 / 2) * (1 - (i / data.iterations));
121
122             // calculate new particle postion
123             let tmpX = agents[j].x + (rand1 * ka * force.x) + (rand2 * kv * agents[j].velocity.x);
124             let tmpY = agents[j].y + (rand1 * ka * force.y) + (rand2 * kv * agents[j].velocity.y);
125
```

Rys. 2.11. Obliczanie nowej pozycji cząstki

Zmienne *ka* i *kv* są implementacją wzoru (1.16).

Jeśli nowa pozycja agenta wyjdzie poza wyznaczony przedział, zostanie on przeniesiony do najbliższej krawędzi.

2.4.2. Gravitational Search Algorithm

Po uruchomieniu algorytmu losowane są pozycje agentów. Wyliczana jest wartość funkcji w tych pozycjach i przystosowanie agentów. Agenci są obiektami klasy *ParticleGSA* umieszczonymi w tablicy *agents*.

Po rozpoczęciu każdej kolejnej iteracji wyliczana jest stała grawitacyjna w danej chwili czasu, ze wzoru (1.20). Przedstawiono to na Rys. 2.12.

```
var gInTime = G * Math.exp(-alfa * ((i + 1) / data.iterations));
```

Rys. 2.12. Implementacja wzoru (1.20) na stałą grawitacyjną w danej chwili czasu

Gdzie *G* jest stałą przyjmującą wartość 100; *i* to aktualna iteracja natomiast *data.iterations* jest maksymalną liczbą iteracji podanych przez użytkownika.

Przed wyliczeniem wypadkowej siły, wyznaczana jest wielkość grupy K_{best}. Następnie agenci są sortowani od tych z najlepszym przystosowaniem do tych z najgorszym. Zaprezentowano to na Rys. 2.13.

```
88              quantityOfBest -= step;
89              if (quantityOfBest < 1) quantityOfBest = 1;
90
91              // sort agents from best to worst fitness
92              agents.sort(function(agent1, agent2) {
93                  if (agent1.value > agent2.value) return -1;
94                  else if (agent1.value < agent2.value) return 1;
95                  else return 0;
96              });
```

Rys. 2.13. Wyznaczanie wielkości grupy K_{best} (wiersze 88 i 89) oraz sortowanie agentów (od 92 - 96)

Stała *step* odpowiada za liniowe zmniejszanie się w czasie wielkości grupy K_{best}, jej implementacje przedstawiono na Rys. 2.14.

```
37          const step = data.numberOfAgents / data.iterations;
38
```

Rys. 2.14. Wyliczanie kroku zmiany wielkości grupy K_{best}

Gdzie *data.numberOfAgents* jest ilością agentów użytą w algorytmie a *data.iterations* to maksymalna liczba iteracji.

Następnie wyliczana zostaje masa agentów, zaprogramowana jak na Rys. 2.15 oraz Rys. 2.16, według wzorów (1.26) i (1.27).

```
for (var j = 0; j < data.numberOfAgents; ++j) {
    agents[j].calculateMass(agents[0].value, agents[data.numberOfAgents - 1].value);
    sumOfMj += agents[j].mj;
}

for (var j = 0; j < data.numberOfAgents; ++j) {
    agents[j].calculateInertiaMass(sumOfMj);

    // calculate new position for worst agents
    if (agents[j].M === 0) {
        agents[j].x = Math.random() * (data.interval.max.x - data.interval.min.x) + data.interval.min.x;
        agents[j].y = Math.random() * (data.interval.max.y - data.interval.min.y) + data.interval.min.y;
    }
}
```

Rys. 2.15. Obliczanie masy bezwładnej

Dla najgorszych agentów masa bezwładna wynosi 0, w takim wypadku zostaje wylosowana ich nowa pozycja. Funkcje *calculateMass()* i *calculateInertiaMass()* są metodami klasy *ParticleGSA.*

```
12      this.calculateInertiaMass = function(sumOfMj) {
13          this.M = this.mj / sumOfMj;
14      }
15
16      this.calculateMass = function(bestValue, worstValue) {
17          // check if function is constant
18          if (bestValue != worstValue)
19              this.mj = ((this.value - worstValue) / (bestValue - worstValue));
20          else this.mj = 0.5;
21      }
```

Rys. 2.16. Implementacja metod *calculateInertiaMass()* i *calculateMass()*

Wzór (1.22) został zrealizowany w sposób zaprezentowany na Rys. 2.17. Wiersze od 129 – 131 odpowiadają za wyliczenie sumy sił oddziaływujących na cząstkę.

```
122                 // calculate sum force acting on j-th paritcle
123                 for (var k = 0; k < quantityOfBest; ++k) {
124                     if (k === j) { continue };
125
126                     let distance = calculateDistance(agents[j], agents[k]);
127                     let scalarForce = 0.0;
128
129                     scalarForce = gInTime * ((agents[j].M * agents[k].M) / distance);
130                     force.x += (scalarForce * (agents[k].x - agents[j].x) * rand);
131                     force.y += (scalarForce * (agents[k].y - agents[j].y) * rand);
132                 }
```

Rys. 2.17. Wyliczanie siły działającej na cząstkę w algorytmie GSA

Gdzie *calculateDistance()* jest funkcją obliczającą zwykłą odległość między wektorami.

Nowe przyspieszenie, prędkości i położenie agentów zostały zaprogramowane w sposób przedstawiony na Rys. 2.18. Gdzie funkcje *calculateAcceleration()* oraz *calculateVelocity()* są metodami klasy *ParticleGSA*, zaprezentowano je na Rys. 2.19.

```
134             agents[j].calculateAcceleration(force);
135             agents[j].calculateVelocity();
136
137             // calculate new particle postion
138             let tmpX = agents[j].x + agents[j].velocity.x;
139             let tmpY = agents[j].y + agents[j].velocity.y;
```

Rys. 2.18. Obliczanie nowej prędkości, przyspieszenia i położenia agenta

```
23  this.calculateAcceleration = function(force) {
24      let e = 0.00000001;
25
26      this.acceleration.x = force.x / (this.M + e);
27      this.acceleration.y = force.y / (this.M + e);
28  }
29
30  this.calculateVelocity = function() {
31      let rand = Math.random();
32
33      this.velocity.x = (rand * this.velocity.x) + this.acceleration.x;
34      this.velocity.y = (rand * this.velocity.y) + this.acceleration.y;
35  }
36  }
```

Rys. 2.19. Metody odpowiedzialne za wyliczenie przyspieszenia i prędkości

Również w tym algorytmie, jeśli nowa pozycja jest poza wyznaczonym przedziałem, agent zostaje przeniesiony do najbliższej krawędzi.

2.4.3. Gravitational Search Algorithm Plus

Tak jak GSA, algorytm wykorzystuje tablicę obiektów klasy *ParticleGSA* do przechowywania informacji o agentach.

Pierwszym etapem jest obliczenie wartości stałej grawitacyjnej w pierwszej chwili czasu (Rys. 2.20), według wzoru (1.28), zaprezentowano to na Rys. 2.21.

```
const G = calculateGravitationalConstant(data.interval);
```

Rys. 2.20. Wywołanie funkcji liczącej stałą grawitacyjną w pierwszej chwili czasu

```
function calculateGravitationalConstant(interval) {
    let rangeX = (interval.max.x - interval.min.x);
    let rangeY = (interval.max.y - interval.min.y);

    return Math.max(rangeX, rangeY) / 2;
}
```

Rys. 2.21. Zaprogramowany wzór (1.28)

Następnie losowane są pierwsze pozycje agentów w podanym przedziale i obliczany. W przeciwieństwie do algorytmu GSA, pomijany jest etap sortowania tablicy agentów i od razu jest wyliczana masa bezwładna cząstek tak jak na . Różnicą jest wywołanie funkcji *calculateMass* z parametrami *best.value* oraz *worst.value* odpowiadającymi za ogólnie najlepsze i najgorsze znalezione rozwiązanie.

Implementacja wzoru (1.30) została przedstawiona na Rys. 2.22. Wiersze 120 i 121 odpowiadają za wyliczanie siły działającej na agenta j-tego przez k-tego według wzoru (1.29). Natomiast 125 i 126 pozwalają zachować kierunek oddziaływania tej siły.

```
112     // calculate sum force acting on j-th paritcle
113     for (var k = 0; k < data.numberOfAgents; ++k) {
114         if (k === j) { continue };
115 
116         let distance = calculateDistance(agents[j], agents[k]);
117         let scalarForce = 0.0;
118         let pij = 1;
119 
120         if (distance >= radius) scalarForce = ((agents[j].M * agents[k].M) / distance);
121         else scalarForce = ((agents[k].M) / Math.pow(radius, 3)) * distance;
122 
123         if (agents[k].value < agents[j].value) pij = 0.00001;
124 
125         force.x += (pij * gInTime * scalarForce * (agents[k].x - agents[j].x) * rand);
126         force.y += (pij * gInTime * scalarForce * (agents[k].y - agents[j].y) * rand);
127     }
```

Rys. 2.22. Wyliczanie wypadkowej siły oddziaływującej na j-tego agenta, implementacja wzoru (1.30)

Funkcja *calculateDistance()* jest identyczna z tą zastosowaną w GSA, tak jak obliczanie nowych pozycji i prędkości agentów przedstawione na Rys. 2.18 i Rys. 2.19 oraz stałej grawitacyjnej w chwili czasu. Natomiast promień pod zmienną *radius* jest wyznaczany tak jak w algorytmie CSS w Rys. 2.6.

3. WYNIKI

Algorytmy zostały ze sobą porównane pod względem średniego czasu pracy oraz średniego najlepszego dopasowania. Funkcje testowe użyte do tych porównań zostały wymienione i opisane w Tabela 3.1. Są to funkcje dla których znane są ich ekstrema globalne. Wykorzystywane są one do testowania algorytmów optymalizujących.

Tabela 3.1. Funkcje testowe

Nazwa funkcji:	**Wzór:**	**Ekstrema globalne:**	**Przedział poszukiwań:**
Rastrigina	$F_1(x,y) = 20 + (x^2 - 10\cos(2\pi x) + (y^2 - 10\cos(2\pi y)$	$F_1(0,0) = 0$	-5.12 ≤ *x,y* ≤ 5.12
Matyasa	$F_2(x,y) = 0.26(x^2 + y^2) - 0.48xy$	$F_2(0,0) = 0$	-10 ≤ *x,y* ≤ 10
Bukkina nr. 6	$F_3(x,y) = 100\sqrt{\lvert y - 0.01x^2\rvert} + 0.01\lvert x + 10\rvert$	$F_3(-10,1) = 0$	-15 ≤ *x* ≤ -5 -3 ≤ *y* ≤ 3
Rosenbrocka	$F_4(x,y) = (1-x)^2 + 100(y - x^2)^2$	$F_4(1,1) = 0$	-4 ≤ x,*y* ≤ 4

3.1. W odniesieniu do zmiennej ilości agentów

Algorytmy został uruchomiony dwadzieścia razy dla każdej zmiany ilości agentów. Owa zmiana wynosiła zawsze 10. Testy zostały rozpoczęte zaczynając od 10 kończąc na 200 agentach na każdej z funkcji testowych. Liczba iteracji była stała, równa 2000.

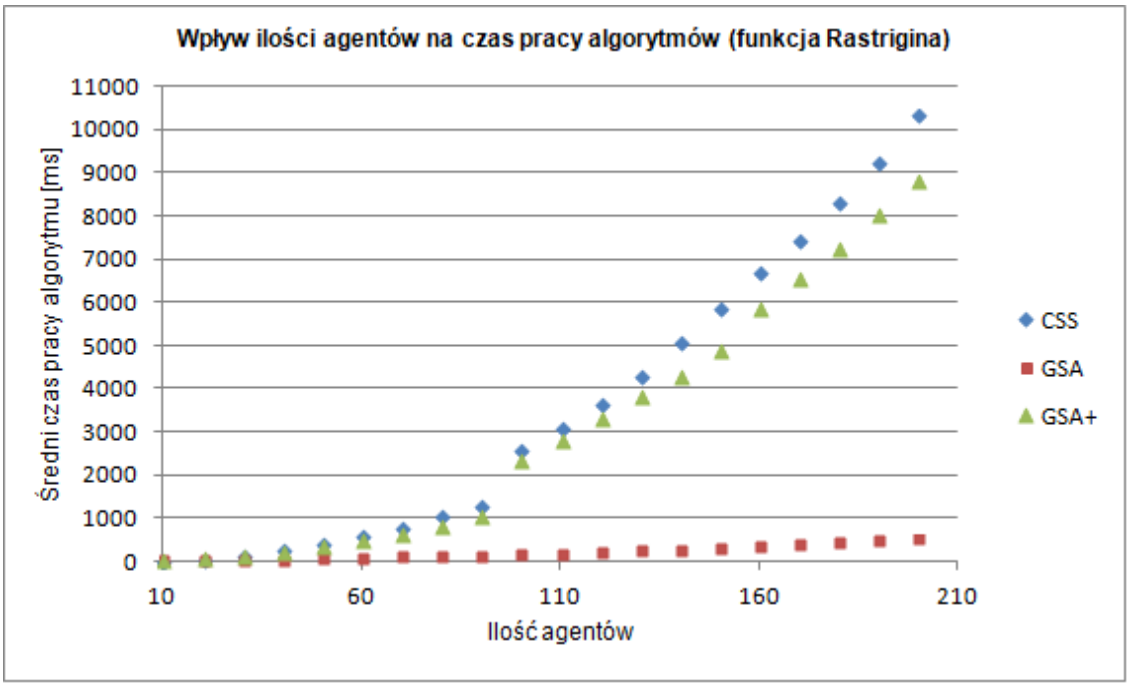
Seria pomiarów dla funkcji F_1 (Rastrigina):

Tabela 3.2. Wyniki zmiany ilości agentów dla funkcji F_1 (Rastrigina), globalnym minimum dla *f(0,0) = 0*

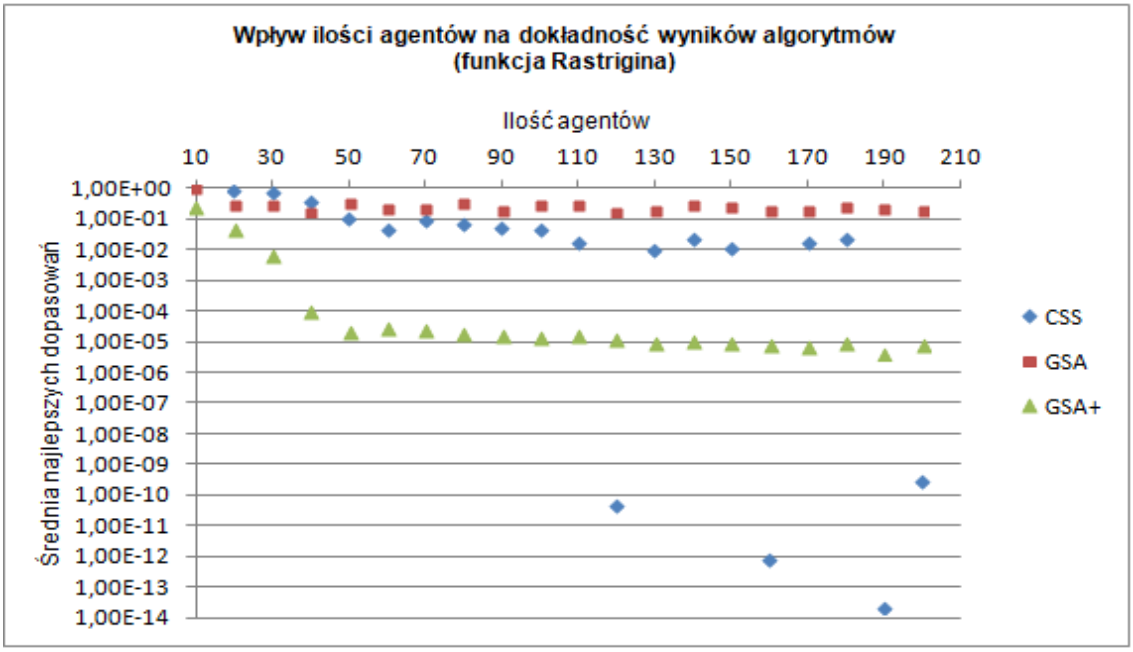
	CSS		**GSA**		**GSA+**	
Ilość agentów:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:
10	3,11E+00	31,19	9,76E-01	19,0292	2,32E-01	42,59
20	8,21E-01	80,01	2,90E-01	25,7345	4,72E-02	88,70
30	7,63E-01	160,24	3,07E-01	41,5818	6,34E-03	161,03
40	3,60E-01	279,05	1,81E-01	56,4024	9,97E-05	247,18
50	1,03E-01	417,55	3,19E-01	68,5285	2,07E-05	364,15
60	4,69E-02	611,15	2,30E-01	83,2767	2,96E-05	516,13
70	9,05E-02	819,85	2,32E-01	108,0844	2,51E-05	668,70
80	6,72E-02	1059,22	3,35E-01	125,9247	1,94E-05	860,87
90	5,42E-02	1332,84	1,88E-01	147,4145	1,66E-05	1067,49
100	4,48E-02	2584,24	2,78E-01	168,2828	1,42E-05	2350,83
110	1,87E-02	3103,02	3,02E-01	192,9013	1,58E-05	2812,32
120	4,91E-11	3676,92	1,80E-01	222,8505	1,14E-05	3342,63
130	9,91E-03	4304,27	1,91E-01	254,5624	9,20E-06	3836,83
140	2,32E-02	5111,77	2,84E-01	288,6182	1,00E-05	4331,98

150	1,20E-02	5902,06	2,59E-01	325,0177	9,19E-06	4923,14
160	8,37E-13	6720,32	2,03E-01	353,2805	7,93E-06	5870,87
170	1,86E-02	7475,21	2,08E-01	399,2484	6,86E-06	6559,45
180	2,29E-02	8351,67	2,47E-01	431,5548	8,56E-06	7252,26
190	2,26E-14	9246,72	2,10E-01	481,547	4,12E-06	8052,36
200	2,90E-10	10352,82	1,85E-01	521,0211	8,13E-06	8819,13

Wykresy na podstaw**ie** Tabela 3.2:



Wykres 3.1. Wpływ ilości agentów na czas pracy algorytmów dla funkcji Rastrigina
Źródło: Własne, Tabela 3.2



Wykres 3.2. Wpływ ilości agentów na dokładność wyników dla funkcji Rastrigina
Źródło: Własne, Tabela 3.2

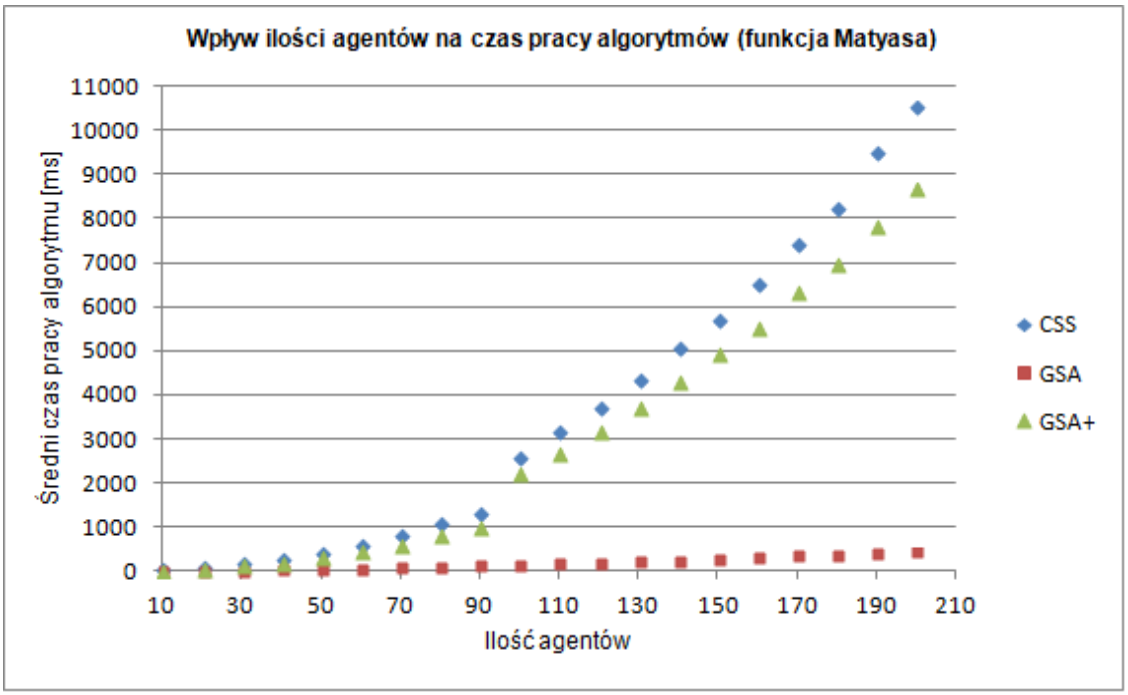
Seria pomiarów dla funkcji $\boldsymbol{F_2}$ (Matyasa):

Tabela 3.3. Wyniki zmiany ilości agentów dla funkcji F_2 (Matyasa), globalne minimum dla *f(0,0) = 0*

	CSS		**GSA**		**GSA+**	
Ilość agentów::	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas trwania [ms]:
10	3,34E-02	32,70	1,04E-02	22,536	1,18E-05	44,47
20	7,09E-03	79,26	6,68E-04	28,3774	3,47E-07	87,24
30	5,49E-05	162,80	5,94E-04	43,2474	2,85E-07	151,56
40	4,97E-05	283,79	5,72E-04	52,8889	5,51E-08	231,05
50	1,56E-09	416,08	4,76E-04	67,1531	9,23E-09	349,47
60	1,73E-15	585,39	7,67E-04	85,0912	1,76E-09	477,29
70	6,11E-14	804,51	6,12E-04	105,763	1,16E-09	637,74
80	5,74E-13	1095,84	4,47E-04	121,4114	7,23E-10	828,51
90	1,12E-13	1335,31	7,94E-04	147,3552	1,04E-09	1030,21
100	9,70E-14	2591,76	6,22E-04	165,2934	2,56E-09	2261,57
110	1,94E-12	3162,55	4,67E-04	187,3243	1,34E-09	2683,97
120	3,52E-13	3731,16	4,69E-04	210,6254	8,15E-10	3213,68
130	4,16E-14	4330,94	4,21E-04	242,8363	1,13E-09	3727,76
140	3,82E-14	5060,67	4,00E-04	270,7415	3,71E-10	4348,90
150	8,30E-15	5710,16	4,55E-04	309,5412	4,57E-10	4950,89
160	2,50E-13	6514,71	4,46E-04	336,2519	5,68E-10	5536,17
170	3,75E-14	7413,82	4,31E-04	366,1217	4,47E-10	6346,07
180	1,18E-13	8248,70	3,77E-04	377,6798	3,69E-10	7018,35
190	7,94E-18	9510,15	4,39E-04	422,2827	1,81E-10	7860,51
200	6,37E-15	10521,40	3,67E-04	453,2898	1,61E-10	8692,42

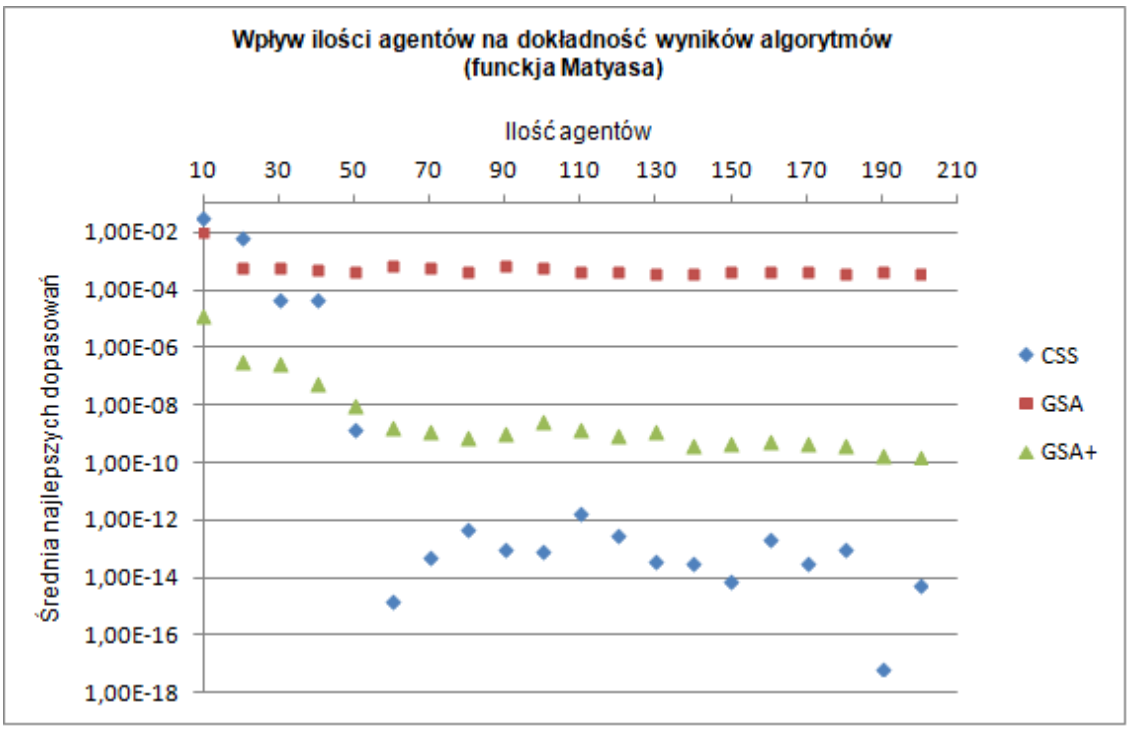
Wykresy na podstawie Seria pomiarów dla funkcji F_2 (Matyasa):

Tabela 3.3:



Wykres 3.3. Wpływ ilości agentów na czas pracy dla funkcji Matyasa
Źródło: Własne, Seria pomiarów dla funkcji F_2 (Matyasa):

Tabela 3.3



Wykres 3.4. Wpływ ilości agentów na dokładność wyników dla funkcji Matyasa
Źródło: Własne, Seria pomiarów dla funkcji F_2 (Matyasa):

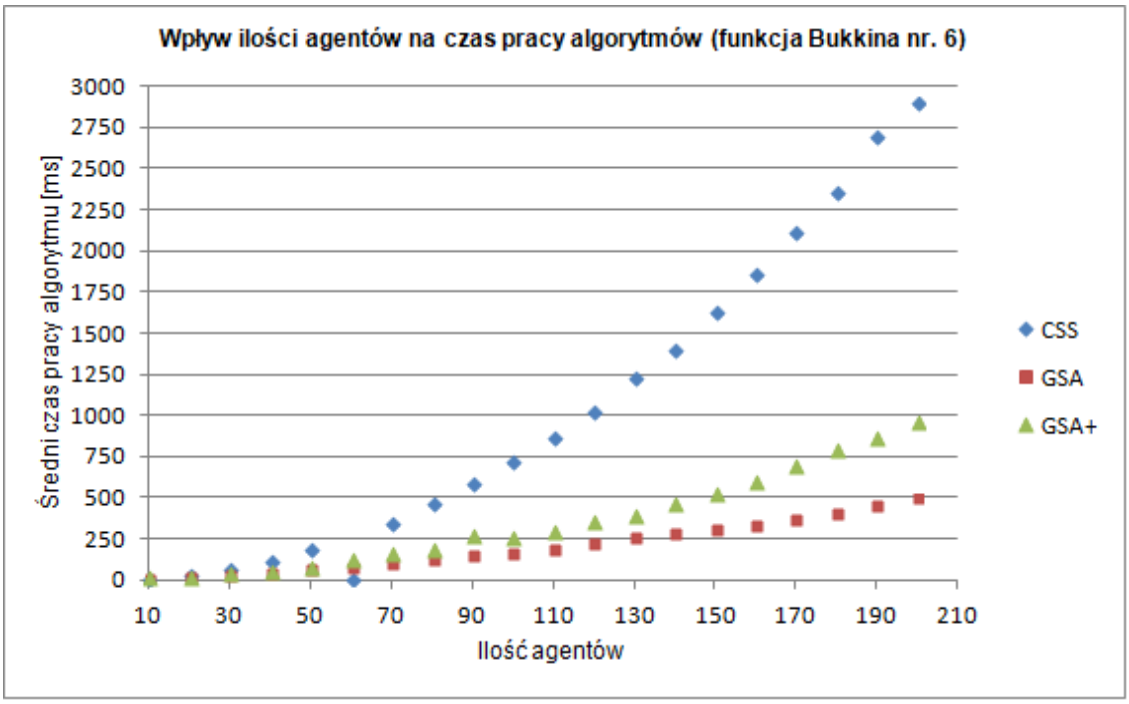
Tabela 3.3

Seria pomiarów dla funkcji F_3 (Bukkina nr.6):

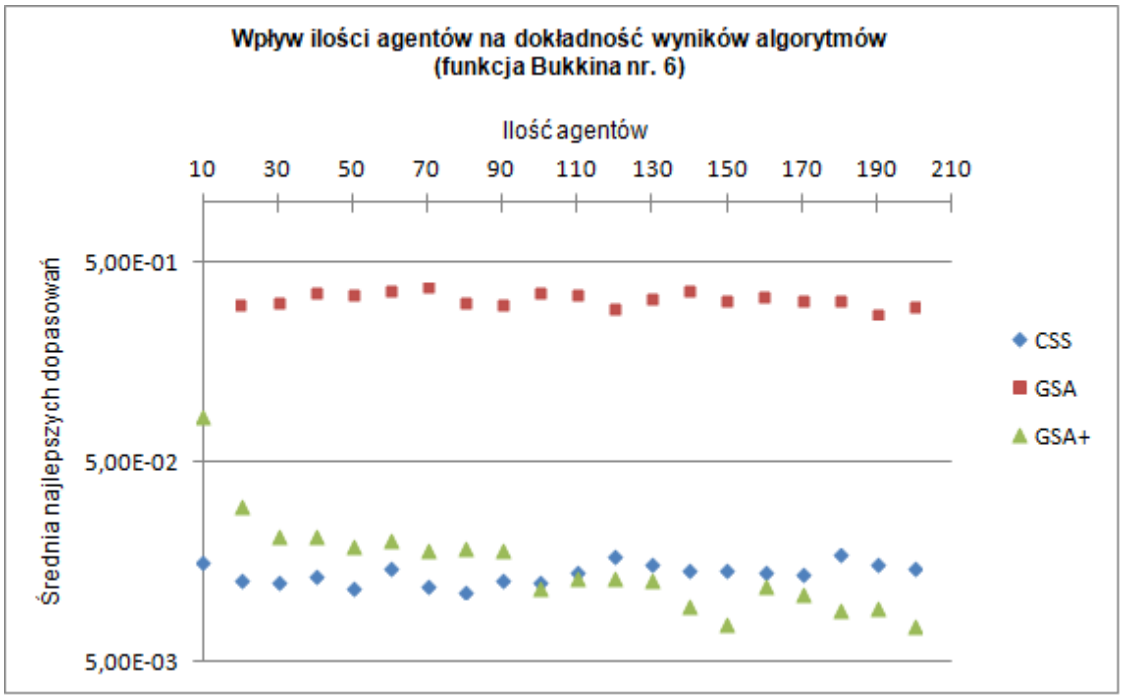
Tabela 3.4. Wyniki zmiany ilości agentów dla funkcji F_3 (Bukkina nr. 6), globalne minimum dla *f(-10,1) = 0*

	CSS		**GSA**		**GSA+**	
Ilość agentów:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:
10	1,58E-02	15,33	3,66E+00	17,94	8,47E-02	15,37
20	1,28E-02	31,26	3,06E-01	24,22	3,01E-02	21,34
30	1,25E-02	77,43	3,13E-01	38,32	2,14E-02	41,40
40	1,36E-02	123,75	3,53E-01	50,60	2,13E-02	59,74
50	1,16E-02	187,69	3,44E-01	68,45	1,91E-02	84,92
60	1,47E-02	11,78	3,59E-01	87,16	2,04E-02	123,15
70	1,20E-02	356,09	3,75E-01	113,15	1,83E-02	159,23
80	1,13E-02	468,81	3,18E-01	129,93	1,88E-02	185,76
90	1,27E-02	594,52	3,12E-01	158,66	1,81E-02	270,11
100	1,26E-02	723,34	3,56E-01	175,54	1,18E-02	266,66
110	1,41E-02	865,71	3,49E-01	198,35	1,31E-02	292,92
120	1,69E-02	1031,40	2,92E-01	231,17	1,33E-02	360,20
130	1,53E-02	1234,06	3,33E-01	269,58	1,29E-02	399,10
140	1,43E-02	1410,57	3,60E-01	287,25	9,63E-03	468,62
150	1,44E-02	1636,54	3,27E-01	316,24	7,86E-03	529,96
160	1,41E-02	1860,10	3,38E-01	335,74	1,20E-02	606,26
170	1,37E-02	2113,94	3,23E-01	379,92	1,10E-02	692,46
180	1,71E-02	2358,07	3,21E-01	418,63	9,16E-03	791,09
190	1,53E-02	2697,21	2,75E-01	462,38	9,41E-03	871,95
200	1,46E-02	2900,86	3,00E-01	504,16	7,60E-03	957,98

Wykresy na podstawie Tabela 3.4:



Wykres 3.5. Wpływ ilości agentów na czas pracy dla funkcji Bukkina nr. 6
Źródło: Własne, Tabela 3.4



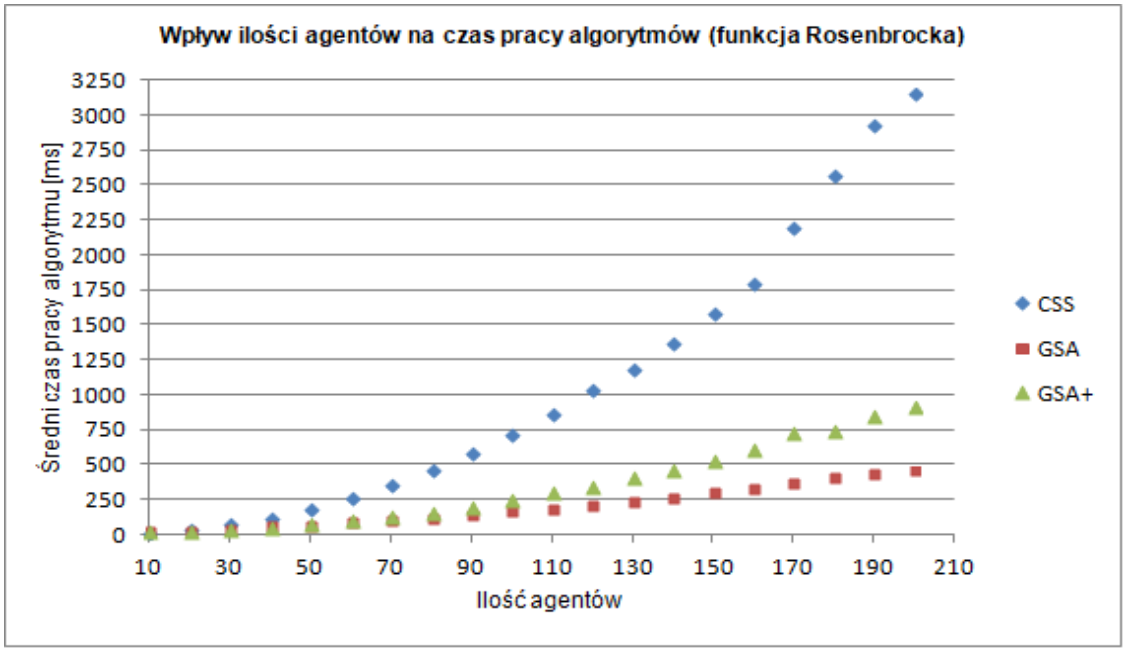
Wykres 3.6. Wpływ ilości agentów na dokładność wyników dla funkcji Bukkina nr. 6
Źródło: Własne, Tabela 3.4

Seria pomiarów dla funkcji F_4 (Rosenbrocka):

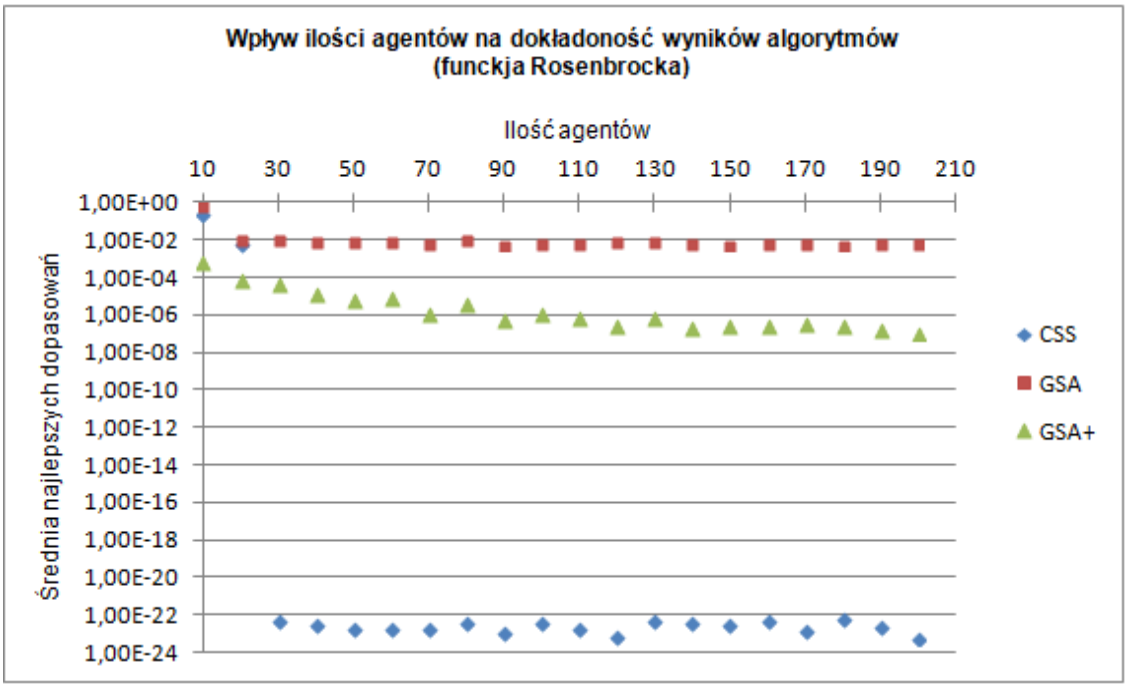
Tabela 3.5. Wyniki zmiany ilości agentów dla funkcji F_4 (Rosenbrocka), globalne minimum dla *f(1,1) = 0*

	CSS		**GSA**		**CSS**	
Ilość agentów:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:
10	2,03E-01	20,70	6,54E-01	22,40	6,91E-04	22,52
20	6,00E-03	39,83	1,28E-02	31,47	9,22E-05	24,41
30	4,82E-23	78,69	1,16E-02	46,25	5,62E-05	38,60
40	3,13E-23	127,11	8,65E-03	60,18	1,74E-05	58,28
50	1,98E-23	195,81	8,13E-03	72,61	6,66E-06	74,51
60	1,56E-23	269,70	9,54E-03	90,15	8,54E-06	102,43
70	1,81E-23	359,81	6,38E-03	107,16	1,51E-06	128,87
80	3,85E-23	469,39	1,08E-02	124,41	4,32E-06	165,63
90	1,04E-23	588,01	5,86E-03	145,72	7,19E-07	200,25
100	3,94E-23	719,77	6,29E-03	168,22	1,45E-06	258,14
110	1,64E-23	868,71	7,22E-03	192,13	8,72E-07	302,18
120	6,20E-24	1044,67	9,71E-03	209,94	3,01E-07	350,43
130	4,77E-23	1185,13	8,14E-03	236,42	8,48E-07	415,09
140	3,51E-23	1383,30	7,53E-03	265,76	2,69E-07	471,04
150	3,05E-23	1591,72	6,23E-03	301,81	2,87E-07	528,93
160	4,90E-23	1802,27	6,80E-03	336,03	3,21E-07	618,96
170	1,49E-23	2210,67	7,24E-03	377,11	3,76E-07	728,50
180	6,41E-23	2582,68	5,87E-03	409,81	2,93E-07	752,96
190	2,03E-23	2934,85	7,75E-03	434,41	2,10E-07	848,33
200	5,64E-24	3157,74	7,10E-03	459,66	1,16E-07	914,07

Wykresy na podstawie Tabela 3.5:



Wykres 3.7. Wpływ ilości agentów na czas pracy dla funkcji Rosenbrocka
Źródło: Własne, Tabela 3.5



Wykres 3.8. Wpływ ilości agentów na dokładność wyników dla funkcji Rosenbrocka
Źródło: Własne, Tabela 3.5

3.2. W odniesieniu do zmiennej liczby iteracji

Algorytmy zostały uruchomione dwadzieścia razy dla każdej zmiany ilości iteracji. Owa zmiana wynosiła zawsze 50. Testy zostały rozpoczęte zaczynając od 50 kończąc na 2000 iteracji na każdej z funkcji testowych. Dodatkowo również została wykonana seria pomiarów dla 10 iteracji. Liczba agentów była stała, równa 200.

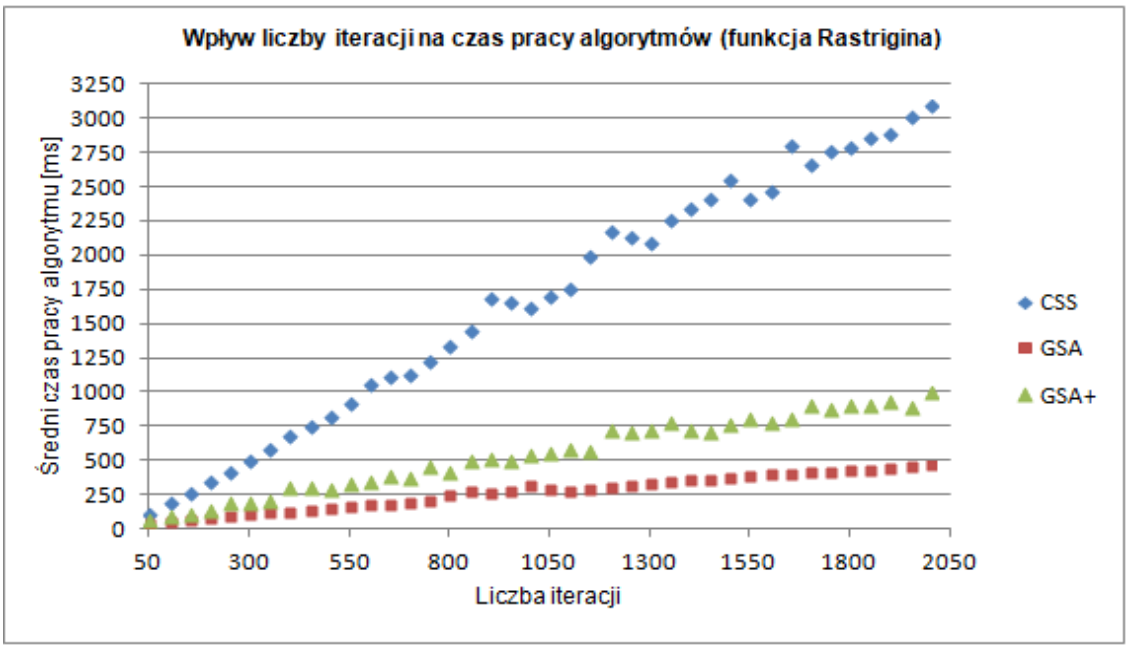
Seria pomiarów dla funkcji F_1 (Rastrigina):

Tabela 3.6. Wyniki zmiany liczby iteracji dla funkcji F_1 (Rastrigina), globalnym minimum dla *f(0,0) = 0*

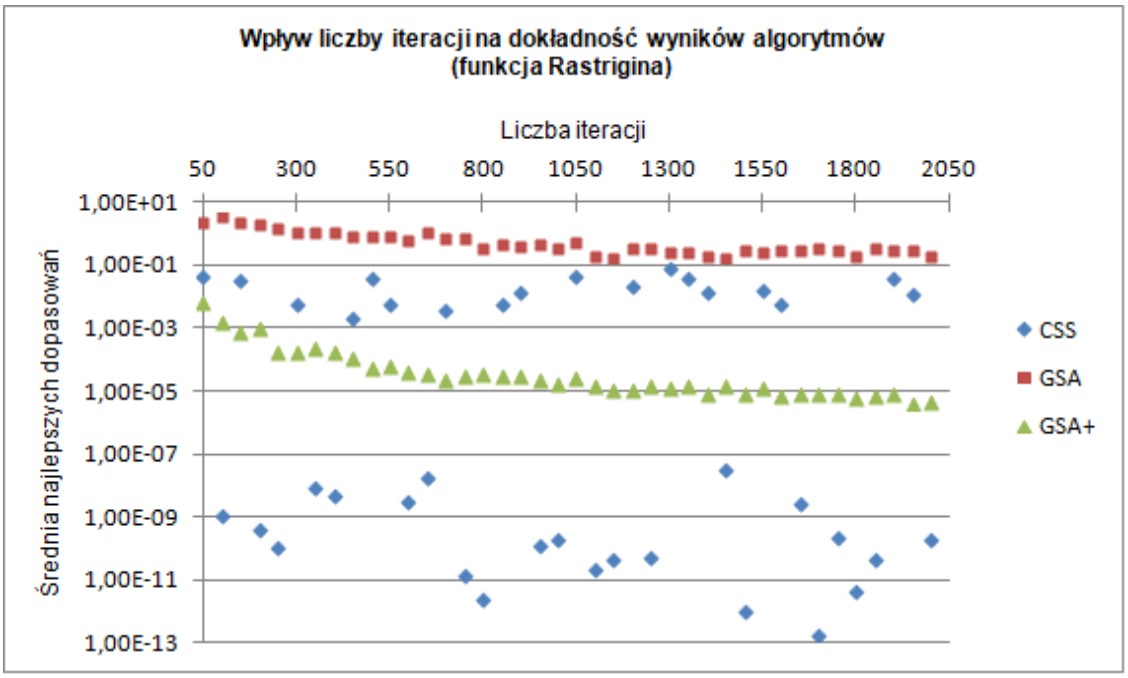
	CSS		**GSA**		**CSS**	
Liczba iteracji:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:
10	8,86E-02	38,05	6,37E-01	32,79	2,39E-01	40,20
50	4,70E-02	106,78	2,35E+00	43,24	6,24E-03	72,10
100	1,28E-09	185,43	3,24E+00	55,99	1,67E-03	96,40
150	3,54E-02	264,36	2,31E+00	73,22	7,65E-04	117,36
200	4,52E-10	346,61	1,83E+00	82,29	9,54E-04	135,42
250	1,29E-10	420,03	1,51E+00	95,66	1,87E-04	195,17
300	6,14E-03	494,67	1,10E+00	109,12	1,76E-04	202,79
350	9,52E-09	577,25	1,07E+00	123,05	2,19E-04	216,67
400	5,34E-09	684,82	1,04E+00	127,98	1,63E-04	304,89
450	2,28E-03	755,26	8,07E-01	137,41	1,06E-04	306,37
500	4,44E-02	820,54	8,15E-01	151,13	5,70E-05	297,82
550	6,54E-03	914,23	8,03E-01	162,74	6,41E-05	342,13
600	3,39E-09	1053,47	6,23E-01	178,40	4,16E-05	349,36
650	2,10E-08	1118,50	1,05E+00	183,90	3,39E-05	386,92

700	4,35E-03	1126,35	7,38E-01	195,74	2,31E-05	378,66
750	1,62E-11	1218,69	7,19E-01	213,67	2,91E-05	465,29
800	2,63E-12	1338,23	3,27E-01	249,29	3,50E-05	421,35
850	6,40E-03	1447,40	4,72E-01	283,35	2,98E-05	508,55
900	1,64E-02	1687,08	3,69E-01	265,96	2,84E-05	513,69
950	1,45E-10	1657,10	4,26E-01	279,68	2,17E-05	508,26
1000	2,18E-10	1612,39	3,37E-01	316,91	1,81E-05	542,30
1050	4,73E-02	1703,94	5,19E-01	293,68	2,58E-05	564,00
1100	2,53E-11	1757,08	1,86E-01	279,08	1,56E-05	587,02
1150	5,36E-11	1995,46	1,65E-01	293,57	1,15E-05	576,77
1200	2,53E-02	2170,88	3,60E-01	306,67	1,13E-05	725,67
1250	5,74E-11	2125,29	3,31E-01	318,63	1,43E-05	710,66
1300	8,39E-02	2085,98	2,44E-01	331,95	1,19E-05	728,66
1350	4,62E-02	2249,07	2,55E-01	347,78	1,40E-05	777,53
1400	1,55E-02	2334,48	1,88E-01	354,32	8,30E-06	726,09
1450	3,82E-08	2405,33	1,75E-01	362,15	1,45E-05	718,21
1500	1,17E-12	2549,85	2,76E-01	370,39	8,41E-06	767,75
1550	1,90E-02	2407,11	2,58E-01	383,44	1,18E-05	815,52
1600	6,24E-03	2470,81	3,04E-01	398,59	6,91E-06	789,21
1650	3,08E-09	2792,24	2,94E-01	406,64	8,00E-06	805,13
1700	2,09E-13	2661,47	3,26E-01	420,16	7,66E-06	906,33
1750	2,65E-10	2754,43	2,80E-01	415,27	8,41E-06	882,40
1800	5,02E-12	2790,40	2,01E-01	425,45	6,04E-06	912,79
1850	4,85E-11	2860,75	3,59E-01	433,86	7,28E-06	909,11
1900	4,13E-02	2885,97	2,83E-01	446,13	8,74E-06	936,27
1950	1,37E-02	3003,26	3,09E-01	452,81	4,17E-06	900,82
2000	2,07E-10	3084,86	1,94E-01	465,45	4,86E-06	999,62

Wykresy na podstawie Tabela 3.6:



Wykres 3.9. Wpływ liczby iteracji na czas pracy dla funkcji Rastrigina
Źródło: Własne, Tabela 3.6



Wykres 3.10. Wpływ liczby iteracji na dokładność wyników dla funkcji Rastrigina
Źródło: Własne, Tabela 3.6

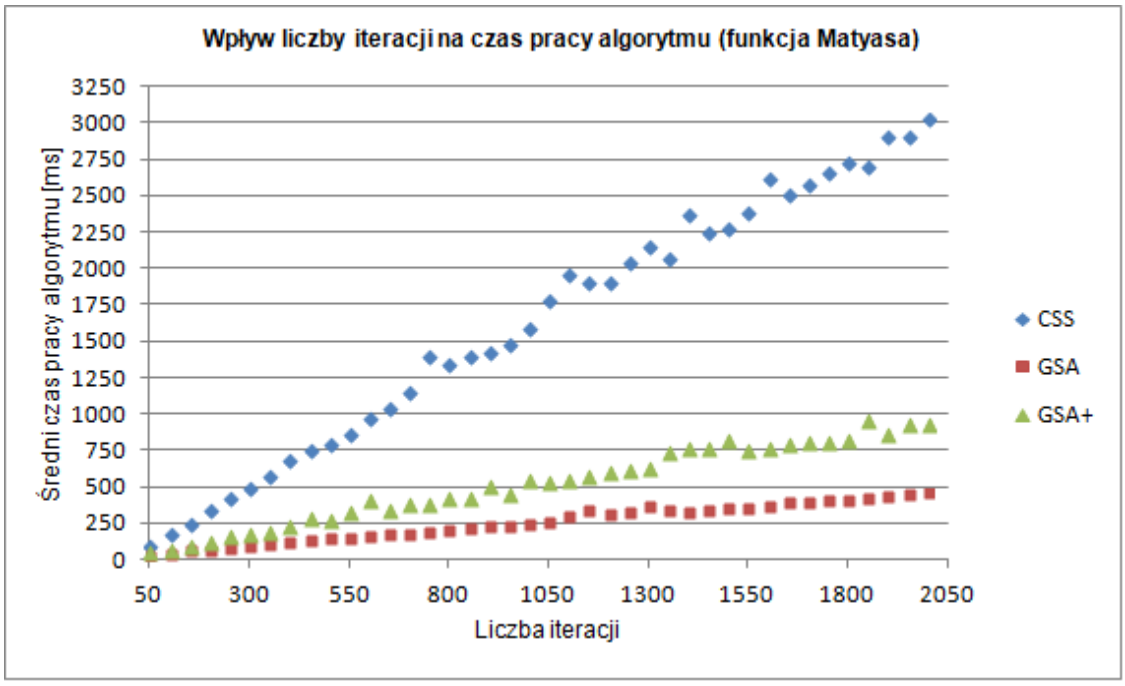
Seria pomiarów dla funkcji F_2 (Matyasa):

Tabela 3.7. Wyniki zmiany liczby iteracji dla funkcji F_2 (Matyasa), globalne minimum dla *f(0,0) = 0*

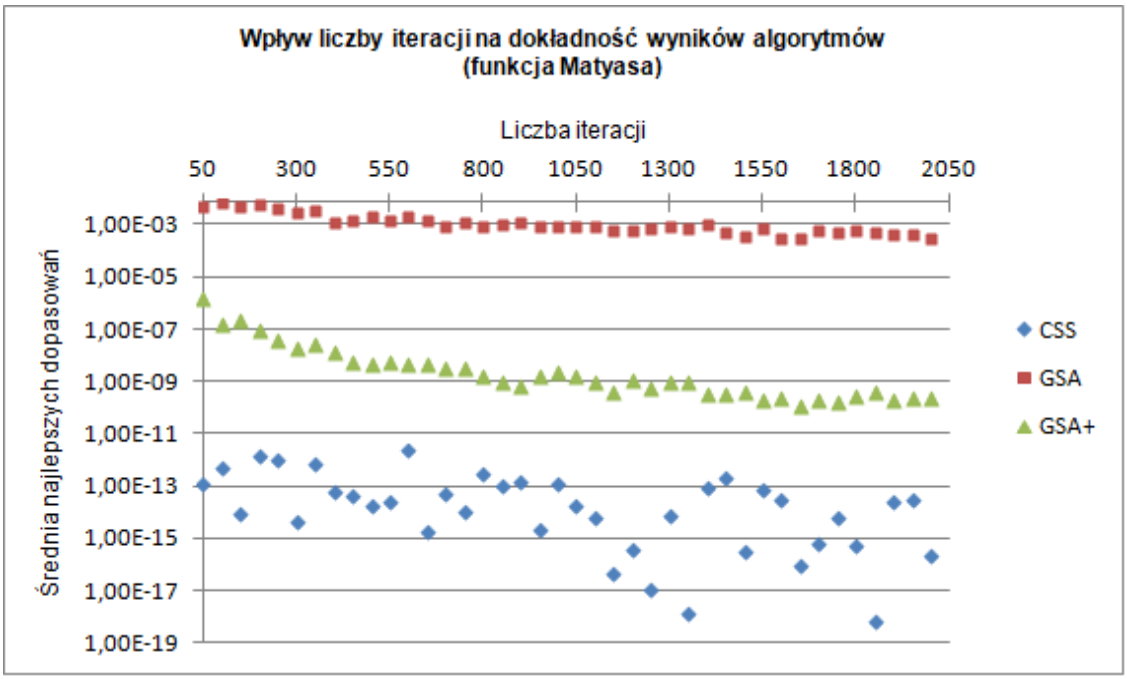
	CSS		**GSA**		**CSS**	
Liczba iteracji:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:
10	5,40E-07	30,16	7,12E-04	35,05	1,15E-04	37,10
50	1,38E-13	105,23	4,72E-03	36,61	1,54E-06	53,30
100	5,61E-13	180,57	6,18E-03	45,91	1,74E-07	75,32
150	9,51E-15	255,96	4,93E-03	66,27	2,33E-07	100,82
200	1,80E-12	343,63	6,01E-03	73,89	9,90E-08	132,51
250	1,10E-12	426,81	3,82E-03	86,24	4,58E-08	161,61
300	5,31E-15	499,54	2,62E-03	98,92	2,01E-08	177,21
350	8,81E-13	574,78	3,09E-03	108,02	2,99E-08	198,62
400	6,55E-14	690,77	1,18E-03	123,84	1,42E-08	236,56
450	5,05E-14	753,90	1,32E-03	138,65	5,70E-09	293,88
500	2,27E-14	801,37	2,09E-03	148,24	5,19E-09	280,58
550	2,82E-14	862,90	1,35E-03	155,93	5,62E-09	333,11
600	2,60E-12	973,80	1,89E-03	166,15	5,20E-09	410,94
650	2,13E-15	1051,58	1,32E-03	185,82	5,24E-09	345,86
700	6,51E-14	1154,63	8,69E-04	187,17	3,76E-09	393,50
750	1,29E-14	1398,42	1,08E-03	194,15	3,81E-09	389,17
800	3,20E-13	1352,08	8,91E-04	209,05	1,90E-09	430,59
850	1,24E-13	1406,97	9,14E-04	220,91	1,04E-09	423,26
900	1,71E-13	1431,69	1,16E-03	233,59	7,33E-10	504,77
950	2,69E-15	1490,65	8,77E-04	239,47	1,69E-09	462,30
1000	1,35E-13	1600,87	8,19E-04	251,17	2,70E-09	546,19

1050	2,10E-14	1787,90	7,61E-04	268,23	1,82E-09	544,21
1100	7,08E-15	1970,88	7,53E-04	310,99	9,98E-10	547,77
1150	5,17E-17	1913,59	5,57E-04	339,09	4,41E-10	581,99
1200	4,49E-16	1914,08	6,20E-04	319,66	1,26E-09	602,53
1250	1,43E-17	2042,79	7,37E-04	334,40	6,19E-10	623,96
1300	9,33E-15	2154,19	8,12E-04	378,74	1,05E-09	628,19
1350	1,65E-18	2071,00	6,92E-04	351,04	1,10E-09	744,51
1400	1,10E-13	2375,28	1,02E-03	332,37	3,69E-10	765,97
1450	2,59E-13	2252,28	4,47E-04	343,15	3,85E-10	768,93
1500	3,89E-16	2284,78	3,51E-04	353,43	4,32E-10	821,07
1550	8,33E-14	2390,25	7,25E-04	365,38	2,05E-10	763,99
1600	3,81E-14	2627,71	2,63E-04	378,62	2,79E-10	764,40
1650	1,15E-16	2513,54	2,84E-04	394,49	1,24E-10	804,58
1700	8,09E-16	2582,32	5,65E-04	398,61	2,14E-10	806,08
1750	7,57E-15	2669,15	4,55E-04	408,99	1,83E-10	814,04
1800	6,36E-16	2734,75	5,75E-04	414,11	3,19E-10	824,87
1850	7,76E-19	2714,16	5,10E-04	428,08	4,17E-10	956,29
1900	2,96E-14	2910,58	3,99E-04	439,26	2,25E-10	873,66
1950	3,58E-14	2913,46	3,84E-04	449,50	2,68E-10	930,68
2000	2,58E-16	3030,50	2,72E-04	462,57	2,56E-10	929,29

Wykresy na podstawie Tabela 3.7:



Wykres 3.11. Wpływ liczby iteracji na czas pracy dla funkcji Matyasa
Źródło: Własne, Tabela 3.7



Wykres 3.12. Wpływ liczby iteracji na dokładność wyników dla funkcji Matyasa
Źródło: Własne, Tabela 3.7

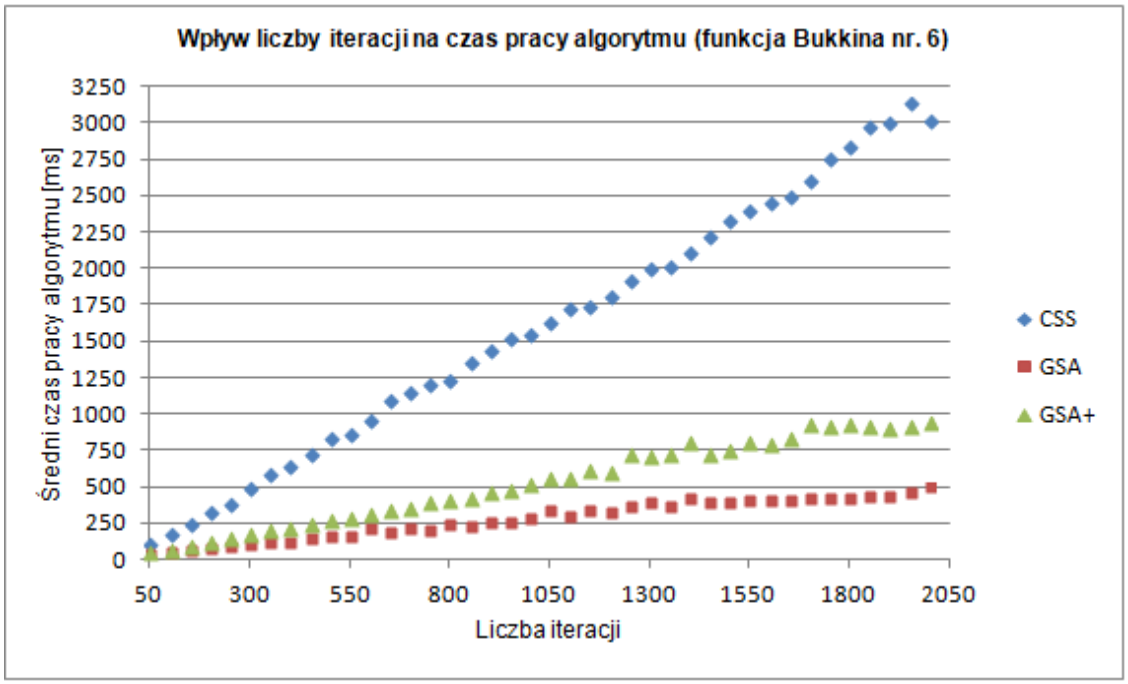
Seria pomiarów dla funkcji F_3 (Bukkina nr.6):

Tabela 3.8. Wyniki zmiany liczby iteracji dla funkcji F_3 (Bukkina nr. 6), globalne minimum dla *f(-10,1) = 0*

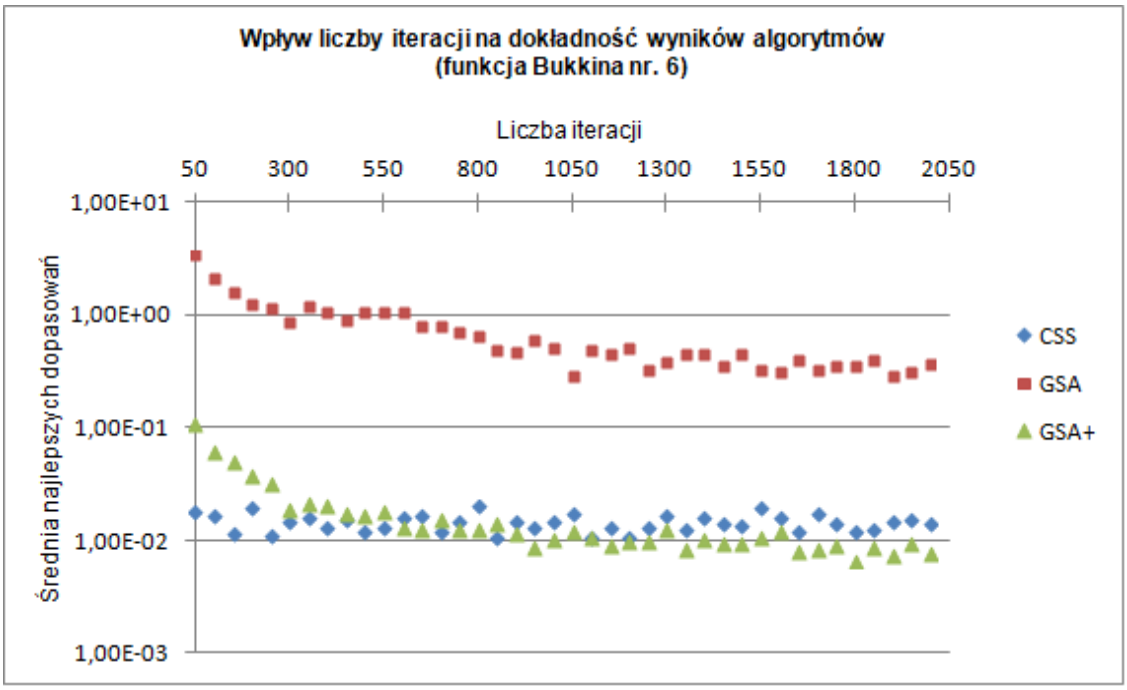
	CSS		**GSA**		**CSS**	
Liczba iteracji:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:
10	6,44E-01	32,25	3,63E+00	27,19	1,47E+00	28,49
50	1,88E-02	96,03	3,43E+00	46,47	1,06E-01	53,46
100	1,70E-02	169,72	2,11E+00	61,20	6,08E-02	71,82
150	1,21E-02	239,51	1,63E+00	76,47	4,97E-02	97,85
200	1,98E-02	316,60	1,27E+00	85,86	3,74E-02	124,71
250	1,14E-02	380,11	1,16E+00	95,20	3,14E-02	150,22
300	1,54E-02	482,76	8,63E-01	114,46	1,87E-02	174,91
350	1,62E-02	580,06	1,19E+00	125,82	2,11E-02	204,76
400	1,35E-02	638,96	1,07E+00	132,33	2,07E-02	222,36
450	1,56E-02	716,13	9,22E-01	150,68	1,72E-02	254,24
500	1,25E-02	826,04	1,05E+00	163,75	1,65E-02	269,01
550	1,32E-02	850,96	1,06E+00	171,12	1,79E-02	294,59
600	1,67E-02	949,53	1,09E+00	223,50	1,30E-02	317,72
650	1,71E-02	1097,07	8,10E-01	190,34	1,23E-02	345,24
700	1,25E-02	1141,56	7,97E-01	220,10	1,55E-02	362,85
750	1,54E-02	1204,53	7,05E-01	205,03	1,24E-02	403,20
800	2,06E-02	1224,41	6,50E-01	248,94	1,22E-02	411,10
850	1,10E-02	1348,99	4,96E-01	231,74	1,39E-02	431,50
900	1,50E-02	1433,53	4,68E-01	265,58	1,13E-02	461,06
950	1,36E-02	1512,82	5,98E-01	269,82	8,65E-03	479,54
1000	1,53E-02	1549,58	5,09E-01	297,01	1,01E-02	522,38

1050	1,77E-02	1627,08	2,95E-01	341,96	1,20E-02	563,33
1100	1,08E-02	1723,13	4,94E-01	308,25	1,07E-02	560,71
1150	1,35E-02	1736,73	4,61E-01	341,41	8,96E-03	624,65
1200	1,08E-02	1801,40	5,15E-01	326,84	9,61E-03	602,33
1250	1,34E-02	1911,56	3,32E-01	378,02	9,69E-03	730,96
1300	1,69E-02	1992,13	3,82E-01	401,96	1,25E-02	708,67
1350	1,27E-02	2006,63	4,49E-01	368,76	8,34E-03	722,10
1400	1,63E-02	2101,64	4,51E-01	432,89	1,01E-02	812,24
1450	1,45E-02	2216,43	3,51E-01	398,10	9,43E-03	729,95
1500	1,41E-02	2319,61	4,65E-01	400,85	9,24E-03	757,90
1550	1,99E-02	2396,60	3,32E-01	408,71	1,05E-02	805,90
1600	1,62E-02	2446,63	3,13E-01	416,06	1,18E-02	791,44
1650	1,26E-02	2485,45	4,07E-01	415,40	7,88E-03	842,22
1700	1,76E-02	2595,57	3,26E-01	427,62	8,17E-03	933,59
1750	1,44E-02	2745,91	3,52E-01	432,03	9,03E-03	919,30
1800	1,22E-02	2837,72	3,58E-01	429,16	6,51E-03	931,64
1850	1,30E-02	2972,99	4,00E-01	438,27	8,51E-03	926,20
1900	1,53E-02	2991,16	2,87E-01	444,09	7,46E-03	906,14
1950	1,56E-02	3127,96	3,23E-01	465,90	9,38E-03	919,64
2000	1,43E-02	3005,35	3,78E-01	504,04	7,75E-03	940,56

Wykresy na podstawie Tabela 3.8:



Wykres 3.13. Wpływ liczby iteracji na czas pracy dla funkcji Bukkina nr. 6
Źródło: Własne, Tabela 3.8



Wykres 3.14. Wpływ liczby iteracji na dokładność wyników dla funkcji Bukkina nr. 6
Źródło: Własne, Tabela 3.8

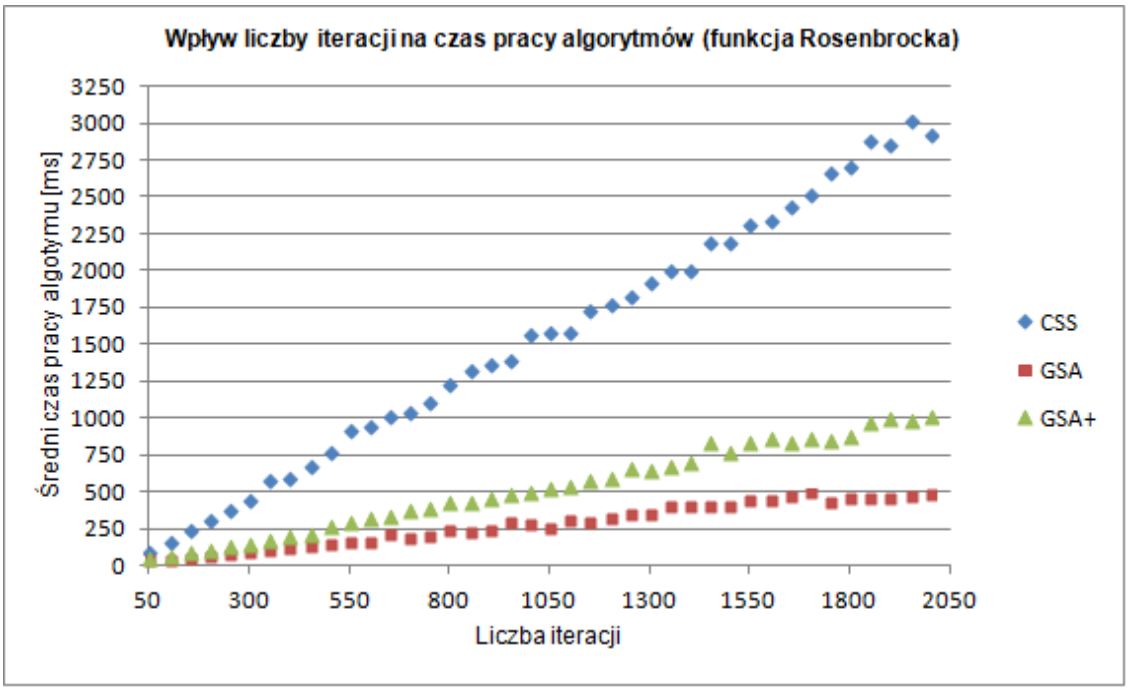
Seria pomiarów dla funkcji F_4 (Rosenbrocka):

Tabela 3.9. Wyniki zmiany liczby iteracji dla funkcji F_4 (Rosenbrocka), globalne minimum dla *f(1,1) = 0*

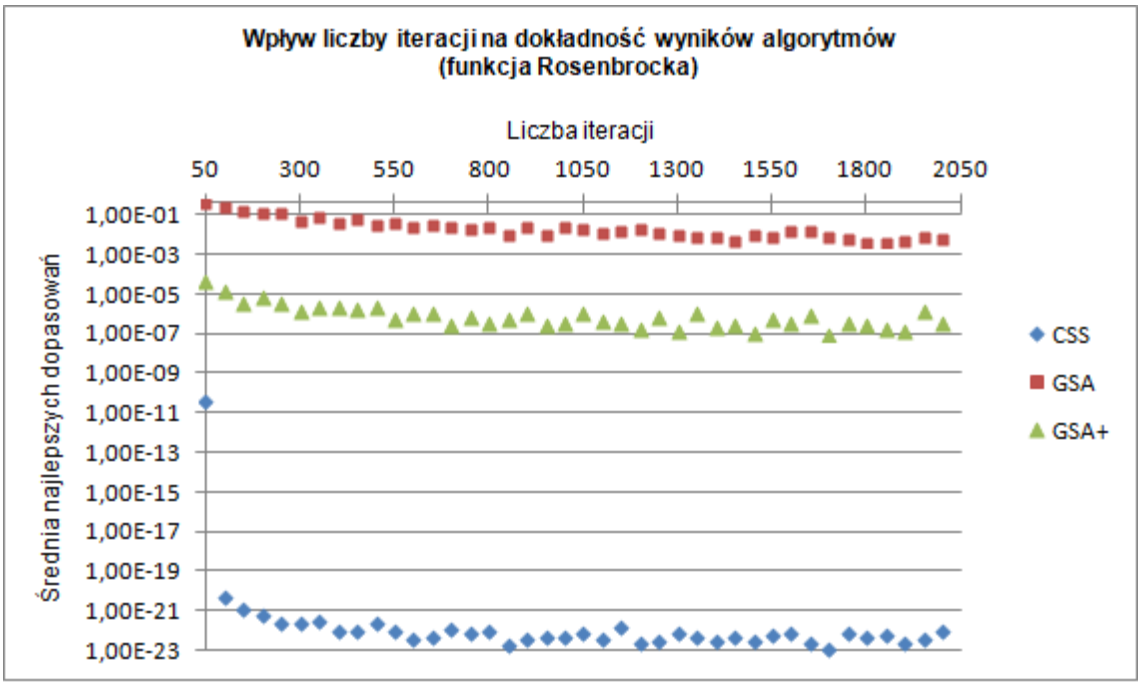
	CSS		**GSA**		**CSS**	
Liczba iteracji:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:	Średnie najlepsze dopasowanie:	Średni czas pracy [ms]:
10	6,99E-02	32,31	1,30E-01	25,70	3,38E-02	28,40
50	3,53E-11	97,49	3,43E-01	37,38	3,91E-05	55,88
100	4,78E-21	164,13	2,43E-01	47,89	1,43E-05	69,92
150	1,24E-21	246,72	1,58E-01	58,98	3,20E-06	94,65
200	5,59E-22	316,37	1,11E-01	67,83	6,89E-06	114,78
250	2,39E-22	384,02	1,08E-01	82,87	3,64E-06	135,72
300	2,19E-22	452,55	4,37E-02	92,45	1,33E-06	158,67
350	3,03E-22	581,39	7,24E-02	104,40	2,07E-06	181,23
400	9,16E-23	599,88	4,21E-02	124,85	1,93E-06	211,69
450	8,87E-23	680,24	5,61E-02	141,53	1,61E-06	226,45
500	2,56E-22	770,09	3,18E-02	153,63	2,38E-06	274,85
550	8,83E-23	928,08	3,67E-02	167,77	5,92E-07	297,63
600	3,68E-23	944,85	2,62E-02	168,06	1,09E-06	333,14
650	4,42E-23	1012,81	2,66E-02	212,72	1,07E-06	342,82
700	1,35E-22	1044,42	2,17E-02	194,42	2,91E-07	384,36
750	7,74E-23	1112,95	2,10E-02	203,56	7,29E-07	392,76
800	9,07E-23	1238,20	2,38E-02	243,80	3,05E-07	437,76
850	1,74E-23	1330,89	8,34E-03	225,89	5,72E-07	443,69
900	3,78E-23	1371,12	2,54E-02	248,84	1,02E-06	463,19
950	5,07E-23	1390,61	1,03E-02	301,81	2,54E-07	486,10
1000	4,48E-23	1572,95	2,16E-02	282,55	3,21E-07	505,72

1050	7,13E-23	1590,75	2,01E-02	262,96	1,00E-06	528,82
1100	3,91E-23	1582,58	1,11E-02	314,16	3,96E-07	550,63
1150	1,38E-22	1730,38	1,36E-02	295,69	3,01E-07	582,14
1200	2,28E-23	1778,98	2,01E-02	324,41	1,78E-07	602,64
1250	2,99E-23	1827,62	1,09E-02	350,89	7,29E-07	672,64
1300	7,75E-23	1926,79	8,81E-03	347,15	1,39E-07	653,32
1350	4,31E-23	2001,03	7,78E-03	410,16	1,06E-06	686,61
1400	2,80E-23	2003,49	7,81E-03	403,56	2,14E-07	708,78
1450	5,33E-23	2201,29	4,54E-03	406,37	2,60E-07	840,54
1500	2,72E-23	2201,74	9,50E-03	409,52	9,82E-08	780,86
1550	5,61E-23	2323,26	7,54E-03	447,27	4,74E-07	837,65
1600	7,74E-23	2348,92	1,44E-02	449,76	3,06E-07	873,91
1650	2,44E-23	2433,36	1,46E-02	473,59	9,32E-07	846,26
1700	1,08E-23	2524,19	7,10E-03	495,64	8,49E-08	864,44
1750	6,93E-23	2671,96	6,23E-03	439,44	3,33E-07	862,32
1800	4,75E-23	2706,03	4,00E-03	459,47	2,39E-07	886,53
1850	5,40E-23	2888,90	4,01E-03	463,91	1,50E-07	976,26
1900	2,68E-23	2865,02	4,71E-03	467,60	1,23E-07	1007,58
1950	3,96E-23	3015,62	6,88E-03	474,56	1,29E-06	994,08
2000	9,08E-23	2930,24	6,07E-03	485,43	3,58E-07	1013,12

Wykresy na podstawie Tabela 3.9:



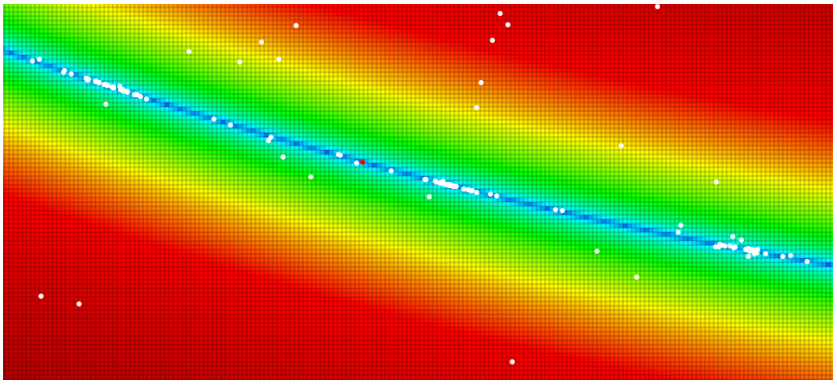
Wykres 3.15. Wpływ liczby iteracji na czas pracy dla funkcji Rosenbrocka
Źródło: Własne. Tabela 3.9



Wykres 3.16. Wpływ liczby iteracji na dokładność wyników dla funkcji Rosenbrocka
Źródło: Własne, Tabela 3.9

3.3. Obserwacje

Agenci algorytmu GSA+ w trakcie działania skupiają się w części ekstremów lokalnych jednocześnie jak na Rys. 3.1. Funkcją testową dobrze prezentującą dane zjawisko jest funkcja F_3 (Bukkina nr.6).



Rys. 3.1. Grupy agentów wokół ekstremów lokalnych

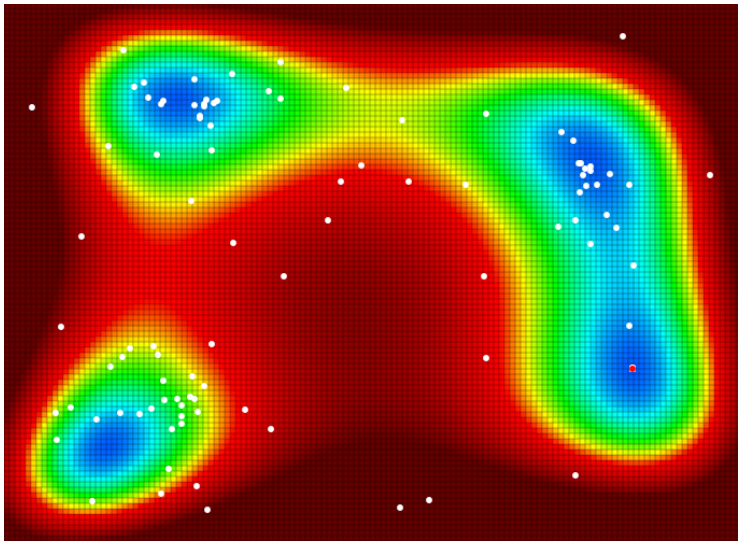
W funkcjach posiadających więcej niż jedno ekstremum globalne agenci skupiają się w obrębie każdego z nich, jak na Rys. 3.2, aż do etapu eksploatacji. W którym to grupa agentów posiadająca najlepsze przystosowanie ściąga pozostałych w swoją stronę, jak na Rys. 3.3. Funkcją, na której można zaobserwować to zjawisko jest nie wymienioną wcześniej funkcją Himmelblaua.

Jej wzór to:

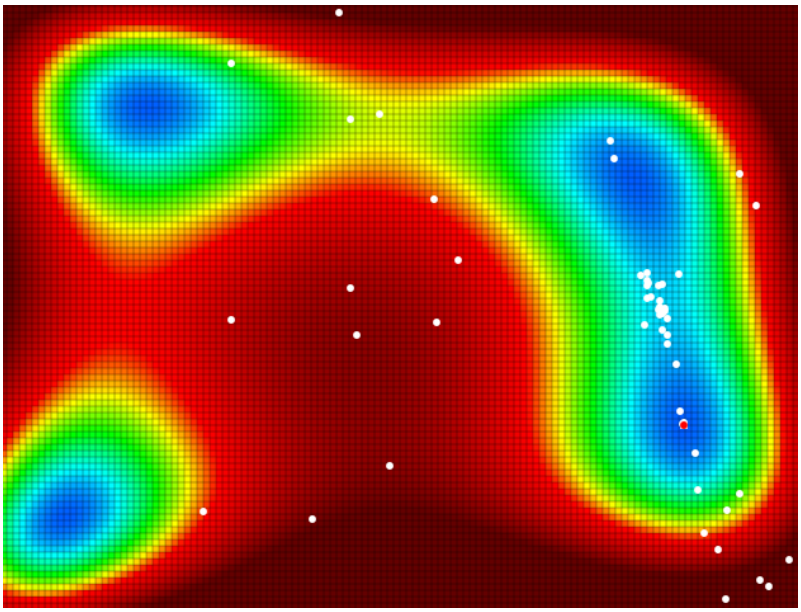
$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \tag{3.1}$$

Podsiada ona cztery następujące minima globalne:

$$\begin{cases} f(3,2) & = 0 \\ f(-2.805118, 3.131312) & = 0 \\ f(-3.779310, -3.283186) & = 0 \\ f(3.584428, -1.848126) & = 0 \end{cases}$$



Rys. 3.2. Etap eksploracji



Rys. 3.3. Etap eksploatacji

4. WNIOSKI

Z przeprowadzonych testów wynika iż algorytm CSS posiada bardzo dokładny etap eksploatacji, jednak często popada w ekstrema lokalne (dobrze widoczne na funkcji Rastrigina F_1). Natomiast obserwacje algorytmu GSA pozwalają stwierdzić, że przechodzi on w etap eksploatacji zbyt wcześnie, jednak po pewnym czasie, jego agenci zostają rozrzuceni w okolicy większości ekstremów lokalnych i globalnych. Pozwala to, na ustalenie przedziału, w którym znajdują się potencjalne ekstrema.

Na podstawie wykresów od Wykres 3.1 do Wykres 3.16 można orzec iż algorytm Gravitational Search Algorith Plus, osiągnął lepsze rezultaty od podstawowego algorytmu GSA. Poprawiony został znacznie etap eksploatacji, który pozwala osiągnąć dokładniejsze wyniki. Mimo to osiągana dokładność jest mniejsza od algorytmu CSS, lecz zmniejszone zostało również prawdopodobieństwo popadania w ekstrema lokalne jak widać na Wykres 3.10 i Rys. 3.1. Jest to związane z zachowaniem etapu eksploracji z algorytmu GSA.

Rezultaty te zostały osiągnięte niewielkim wzrostem czasu obliczeniowego algorytmu w stosunku do algorytmu GSA.

PODSUMOWANIE

Celem niniejszego projektu było stworzenie aplikacji komputerowej pozwalającej na optymalizację globalną algorytmami metaheurystycznymi bazującymi na podstawowych prawach fizyki, a także udoskonalenie jednego z nich. Zostało to zrealizowane dzięki dokładnej analizie zasady działań tych algorytmów i próbie znalezienia w nich najistotniejszych fragmentów. W projekcie zawarto instrukcję obsługi aplikacji oraz teoretyczne przedstawienie nowego algorytmu. Zamieszczono również wyniki pomiarów dla wszystkich zaimplementowanych algorytmów i ich porównanie ze sobą, potwierdzających osiągnięcie celu.

Aplikacja została napisana w formie strony internetowej, wykorzystano do tego języki programowania: HTML5 i JavaScript. Taka forma aplikacji pozwala na jej większą niezależność od systemów operacyjnych oraz większą wygodę w jej użytkowaniu. Aplikacja pozwala również na wizualizację działań algorytmów na funkcji celu. Graficzne przedstawienie funkcji, wykonane zostało za pomocą interpolacji kolorów, w modelu przestrzeni barw HSV.

Udoskonalonym algorytmem został algorytm GSA, po przez uwzględnienie promienia jego agentów według wzoru z algorytmu CSS. Został również zmieniony schemat oddziaływań między cząstkami, tak aby słabsza cząstka mogła również oddziaływać na cząstkę lepszą od siebie.

Cel pracy został osiągnięty przy wykorzystaniu wiedzy kierunkowej zdobytej w trakcie studiów pierwszego stopnia.

Podjęcie dalszych prac nad projektem pozwoliło by na zwiększenie dokładności algorytmów oraz zmniejszenie czasu ich działania, na przykład przez zrównoleglenie obliczeń wykorzystując do tego karty graficzne. Pozwoliło by to na zwiększenie maksymalnej liczby iteracji i ilości agentów. Dalsze prace umożliwiły by również na znalezienie i wykorzystanie szybszych bibliotek przetwarzających ciągi znaków na funkcje matematyczne, lub zaprogramowanie własnych. Pozwoliło by także na przystosowanie algorytmów do funkcji o nieokreślonej liczbie zmiennych oraz na przeprowadzeniu większej serii testów.

Tak wykończony projekt mógł by zostać użyty do celów naukowych i komercyjnych, przy poszukiwaniu ekstremum globalnego bardzo złożonych funkcji celu w fizyce kwantowej lub trudnych zagadnieniach technicznych. Pozwolił by to również na wykorzystanie go do uczenia sieci neuronowych, które są przyszłością informatyki.

WYKAZ LITERATURY

1. M. Dorigo, et al. Ant Colony Optimization And Swarm Intelligence (Springer, 2006).

2. Kaveh A, Talatahari S. A novel heuristic optimization method: charged system search, Acta Mechanica, 2010, DOI: 10.1007/ s00707-009-0270-4

3. Rashedi, E.; Nezamabadi-pour, H.; Saryazdi, S. (2009). "GSA: a gravitational search algorithm". Information Science 179 (13): 2232–2248

4. Nobahari, H.; Nikusokhan, M.. "Non-dominated Sorting Gravitational Search Algorithm". International Conference on Swarm Intelligence.

5. Y.L. Lin, W.D. Chang, J.G. Hsieh, A particle swarm optimization approach to nonlinear rational filter modeling, Expert Systems with Applications 34 (2008) 1194–1199.

6. Kaveh, A., Talatahari, S.: An improved ant colony optimization for constrained engineering design problems. Engineering Computations 27(1) (2010, in press)

7. Kaveh, A., Talatahari, S.: Particle swarm optimizer, ant colony strategy and harmony search scheme hybridized for optimization of truss structures. Comput. Struct. 87(5–6), 267–283 (2009)

8. J.S. Zieliński Inteligentne systemy w zarządzaniu. Teoria i praktyka. PWN (Warszawa, 2000)

WYKAZ STRON INTERNETOWYCH

9. Strona internetowa MDN web docs https://developer.mozilla.org (data dostępu 21.12.2017)

10. Strona internetowa w3schools.com https://www.w3schools.com (data dostępu 21.12.2017)

11. Strona internetowa alanzucconi.com https://www.alanzucconi.com
(data dostępu 21.12.2017)

WYKAZ RYSUNKÓW

WYKAZ TABEL

WYKAZ WYKRESÓW

ZAŁĄCZNIKI

1. Płyta CD-ROM z kodem źródłowym oraz cyfrową wersją dokumentu