AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

Praca dyplomowa magisterska

Skuteczność algorytmu roju cząstek w wybranych problemach optymalizacji ciągłej

Effectiveness of particle swarm algorithm in selected continuous optimization problems

Autor: *Mateusz Gruszka*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr hab. inż. Marek Kisiel-Dorohinicki*

Kraków, 2015

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję promotorowi oraz opiekunowi za wsparcie oraz pomoc na każdym etapie powstawania pracy.

Spis treści

Spis rysunków

1. Wprowadzenie

Tematem niniejszej pracy jest sprawdzenie skuteczności algorytmu roju cząstek w wybranych problemach optymalizacji ciągłej. Skuteczność rozwiązania została porównana z istniejącym rozwiązaniem zaimplementowanym na platformie pyAgE.

Optymalizacja jest wyznaczeniem spośród dopuszczalnych rozwiązań danego problemu, rozwiązania najlepszego ze względu na przyjęte kryterium (wskaźnik) jakości (np. koszt, zysk, niezawodność). Optymalizacja ciągła jest takim rodzajem optymalizacji, w którym wszystkie zmienne funkcji celu są ciągłe, czyli należą do nieprzerwanego zbioru liczb. Często problemy optymalizacyjne są problemami, w których przestrzeń przeszukiwań jest za duża lub zbyt skomplikowana, żeby można było ją eksplorować jednocześnie wydajnie i dokładnie.

Przy rozwiązywaniu tak złożonych problemów wykorzystywane są heurystyki. Heurystyką nazywamy taką metodę znajdowania rozwiązań, która nie daje gwarancji znalezienia optymalnego rozwiązania. Brak gwarancji znalezienia rozwiązania wynika z faktu, że heurystyki przeszukują przestrzeń rozwiązań i wraz z postępem algorytmu modyfikują jego składowe parametry dostosowywując się do osiąganych wyników pośrednich.

Algorytm roju cząstek, jak również algorytm wykorzystywany w niniejszej pracy w celu porównia skuteczności są heurystykami. Doskonalą one rozwiązanie danego problemu poprzez iteracyjne modyfikowanie parametrów. Algorytmy ewolucyjne czerpią z obserwowanego w przyrodzie procesu ewolucji - podczas działania algorytmu jego najlepiej przystosowane składowe są iteracyjnie modyfikowane, natomiast te przystosowane najgorzej są odrzucane. Algorytm roju cząstek bazuje na zachowaniach stadnych. W każdej kolejnej iteracji elementy algorytmu wpływają na siebie na podstawie swojego położenia w przestrzeni rozwiązań.

1.1. Cele pracy

Celem pracy była implementacja algorytmu roju cząstek oraz zbadanie jego skuteczności w wybranych problemach benchmarkowych. W implementacji zostało wykorzystane istniejące środowisko agentowe - pyAgE. Wspiera ono budowę rozproszonych modeli obliczeniowych oraz dostarcza zestaw narzędzi umożliwiających porównanie zaimplementowanych rozwiązań.

Finalnym celem było porównanie otrzymanych rozwiązań z istniejącym algorytmem ewolucyjnym - EMAS. Aby wykonać porównanie zostały przeprowadzone analizy składowych parametrów wykona-

nych modyfikacji algorytmu rojowego, a następnie zostały wykorzystane te, które dawały najdokładniejsze rozwiązanie.

1.2. Zakres pracy

W czasie realizacji pracy, zostały stworzone komponenty na platformę pyAgE dające możliwość dokonywania obliczeń za pomocą algorytmu roju cząstek. Zostały zaimplementowane zarówno elementy składowe algorytmu, jak i przetestowane konfiguracje pozwalające na badanie jego skuteczności.

Porównanie odbywało się na platformie pyAgE pomiędzy istniejącym algorytmem ewolucyjnym EMAS, a różnymi konfiguracjami algorytmu roju cząstek.

Algorytm roju cząstek został przystosowany, aby sprawdzić jego skuteczność w kilku wersjach:

- Podstawowa wersja, zawierająca możliwość sterowania:
  - Prędkością
  - Wagami składowych
- Rozszerzoną o kroki algorytmu ewolucyjnego wykonywane razem z rojowymi
- Rozszerzoną o informację o sąsiadach, dostarczaną przez platformę pyAgE
- Rozszerzoną o wyspy obliczeniowe

1.3. Zawartość pracy

Rozdział 4 zawiera informacje o platformie pyAgE i jej komponentach. Znajdują się w nim również informacje o funkcji benchamrkowej stosowanej do porównania wyników niniejszej pracy.

Następną częścią pracy (rozdziały 3 i 2) jest przybliżenie mechanizmów działania zarówno algorytmów rojowych, jak i ewolucyjnych - szczególnie tych, które zostały zastosowane w implementacji. W rodziale 3.3 zostały opisane częste modyfikacje algorytmu roju cząstek.

Rozdział 5 opisuje sposób zbierania i porównywania otrzymanych wyników. W rozdziale 6 znajdują się informacje o zaimplementowanych rozwiązaniach oraz wyniki badań prowadzące do uzyskania najlepszych parametrów algorytmu roju cząstek.

Zakończenie pracy poświęcone zostało porównaniu algorytmów rojowego i ewolucyjnego oraz analiza skuteczności algorytmu rojowego pod kątem postawionego problemu.

2. Algorytmy ewolucyjne

Algorytmami ewolucyjnymi nazywne są algoytmy, które w celu przeszukania przestrzeni rozwiązań wykorzystują mechanizmy zaczerpnięte ze zjawiska ewolucji biologicznej. Jest to ogólna nazwa dla metod takich jak algorytmy ewolucyjne, strategie ewolucyjne czy neuroewolucje. Przykładowe rozwiązanie danego problemu reprezentowane jest przez genotyp każdego z osobników. Finalne rozwiązanie wybierane jest spośród genotypów wszystkich osobników w danej populacji.

Podobnie jak opisane w rozdziale 3 algorytmy rojowe, ewolucyjne również zawierają populację osobników wpływających nawzajem na siebie. Populacja generowana jest losowo, wraz z pewnym zestawem cech dla każdego osobnika - genotypem. Genotyp jest takim zestawem cech, który umiejscawia go w pewnej przestrzeni rozwiązań, co umożliwia jego ewaluację. Podczas działania algorytmu, nowe osobniki tworzone są poprzez krzyżowanie się istniejących, najgorzej przystosowane umierają, a poprzez mutację wpływają na swoje genotypy.

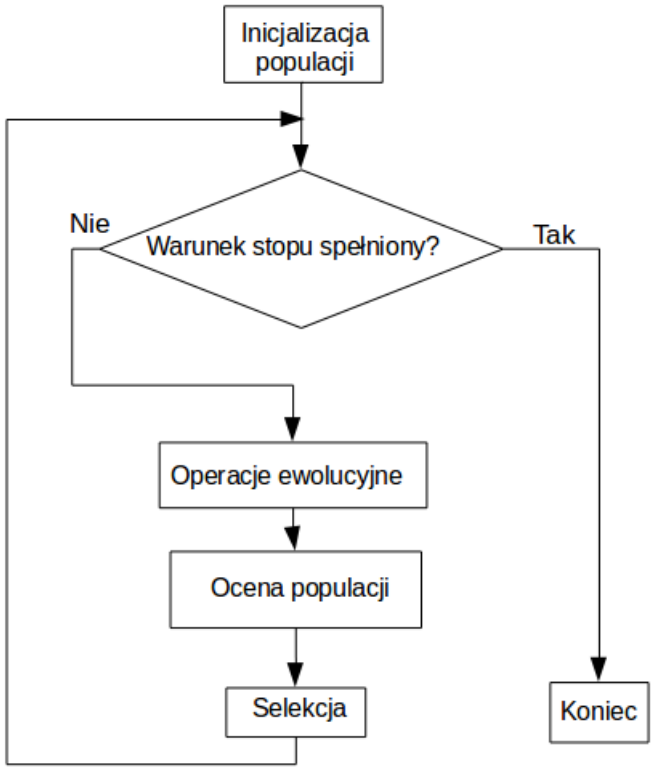
Na przestrzeni lat zostało zaproponowanych wiele algorytmów bazujących na mechanizmach genetycznych, jednak wszystkie z nich opierały się na tych samych bazowych mechanizmach. Każdy z osobników populacji mógł, zmieniając swój genotyp, przybliżyć całą populację do znalezienia optymalnego rozwiązania postawionego problemu. Większość współczesnych rozwiązań stosuje również krzyżowanie się osobników, jako drugą główną składową działania algorytmów tego typu.

2.1. Przegląd wiedzy

Początki algorytmów ewolucyjnych sięgają lat 50. XX wieku [13], jednak ich idee nie były rozwijane przez wiele lat, głównie ze względu na ograniczenia sprzętowe, jak i metodologiczne. Dopiero dwadzieścia lat później [14] pojawiły się prace rozwijające modele ewolucyjne. Wtedy też zostało zaproponowane twierdzenie Hollanda o schematach, które uważane jest za podstawę wyjaśnienia algorytmów ewolucyjnych.

Znaczącą kwestią wpływającą na tempo rozwoju algorytmów ewolucyjnych było użycie techniki uwzględniającej ewolucję zarówno przez mutację, jak i krzyżowanie się osobników z danej populacji. Podczas kolejnych lat badań, algorytmy tego typu zostały poszerzone o kod genetyczny pozwalający reprezentować strukturę każdego problemu.

Klasyczne algorytmy ewolucyjne działają zgodnie z algorytmem przedstawionym na diagramie 2.1 lub podobnie do niego.



Rysunek 2.1: Diagram blokowy klasycznego algorytmu ewolucyjnego

Początkowo inicjalizowana jest losowa populacja, która aż do spełnienia warunku stopu, oddziałuje na siebie zgodnie z pewnymi zasadami. Warunkiem stopu, podobnie jak w algorytmie roju cząstek, może być osiągnięcie limitu iteracji lub zadowalającego wyniku. Podczas każdej iteracji z całej populacji wybierana jest część osobników, która zostanie poddana krzyżowaniu się między sobą. Następnie te osobniki poddawane są mutacji. Dla każdego z nich wyliczana jest funkcja przystosowania pozwalająca ocenić jakość jego genotypu.

2.2. Algorytm EMAS

Algorytm EMAS (ang. Evolutionary Multi-Agent System) jest paradygmatem obliczeniowym zaproponowanym w 1996 roku [16]. Jest to połączenie algorytmu ewolucyjnego z systemem wieloagentowym. Idea algorytmu opiera się na koncepcji, która mówi, że agenci w środowisku mogą się spotykać, reprodukować, mutować i umierać.

Podczas pracy algorytmu nie jest wymagana żadna globalna wiedza o problemie, wszystkie obliczenia są zdecentralizowane. Agenci są niezależni oraz zdolni do podejmowania własnych decyzji dotyczących ich akcji. Wykorzystywanie takich struktur organizacji agentów jak wyspy obliczeniowe pozwala na łatwą skalowalność algorytmu.

Dziedziczenie i selekcja to dwa główne elementy algorytmów ewolucyjnych, które w algorytmie EMAS realizowane są za pomocą zjawisk reprodukcji i śmierci. Agenci o najlepszym przystosowaniu są zachowani i mogą produkować swoje potomstwo. Agenci o najgorszych parametrach są całkowicie usuwani z otoczenia. Takie zachowanie zmusza populację do ewolucji oraz w każdym kroku zbliża ją do lokalnego ekstremum [17].

Każdy z agentów posiada energię, która warunkuje wykonywane przez niego akcję. Jeśli energia agenta spadnie do zera, oznacza to jego śmierć i skutkuje usunięciem z populacji. Energia jest wymieniana przez agentów podczas spotkań. Agent z mniejszym poziomem dopasowania oddaje swoją energię agentowi z większym poziomem dopasowania.

2.2.1. Agent

Każdy z agentów charakteryzuje się trzema parametrami:

– genotyp (ang. genotype)

– dopasowanie (ang. fitness)

– energia (ang. energy)

Genotyp agenta jest pojedynczą instancją rozwiązania zadanego problemu populacji. Jest to podstawa do obliczenia dopasowania. Genotyp jest cechą dziedziczoną podczas reprodukcji i może ulec jednorazowej mutacji podczas powstawania agenta.

Kolejną cechą jest dopasowanie, które jest liczbą reprezentującą jakość genotypu. Lepsze genotypy mają lepsze wartości dopasowania i mają większe prawdopodobieństwo aby być wybranym przy procesie reprodukcji, ponieważ najczęściej osobniki z najlepszym genotypem mają najwyższą wartość energii (warunkującej reprodukcję osobnika). Wartość dopasowania obliczana jest na podstawie genotypu. Dopasowanie danego agenta, tak samo jak jego genotyp, nie zmienia sie w czasie jego życia.

Ostatnią cechą wpływającą na sposób selekcji jest energia. Ze względu na brak globalnej wiedzy agentów, nie jest możliwa ocena ich wszystkich w tym samym czasie. Ponieważ proces ewolucji jest asynchroniczny, metody selekcji znane z klasycznych algorytmów ewolucyjnych nie mogły zostać użyte. Z tego powodu została wprowadzona energia. Można opisać ten parametr jako stan agenta. Podczas spotkań agent może energię zyskiwać lub tracić, zależnie od jakości jego genotypu (czyli wartości dopasowania). Agenci z lepszym genotypem są bardziej skłonni do gromadzenia energii. Całkowita ilość energii w populacji jest stała.

Ponieważ agenci w algorytmie EMAS są całkowicie autonomiczni, decyzję o swoim zachowaniu podejmują na podstawie poziomu energii. Jeśli jej liczba przekracza pewien próg, to mogą się reprodukować, a jeśli osiągnie zero, to agent umiera [18]. Energia wymieniana jest podczas spotkań agentów.

2.2.2. Interakcje agentów

Istnieją trzy możliwe działania agenta, które może podjąć w danym kroku iteracji:

– śmierć (ang. death)

– reprodukcja (ang. reproduction)

– spotkanie (ang. meeting)

Śmierć to usunięcie agenta z populacji, spowodowane jest poprzez osiągnięcie przez danego agenta zerowego poziomu energii.

Reprodukcja jest procesem tworzenia nowych agentów. Wymaga wystąpienia jednego lub dwójki rodziców i skutkuje jednym nowopowstałym agentem. W przypadku zaistnienia dwójki rodziców, potomek ma cechy będące pochodną obojga z nich. Jak zostało wspomniane w rozdziale 2.2.1, rozwiązanie oraz dopasowanie są stałe podczas życia agenta, więc ich wartość u rodziców się nie zmienia. Jedyną zmianą parametru jest zmniejszenie ich poziomu energii, która jest przekazywana nowopowstałym agentom. Genotypy nowonarodzonych agentów są tworzone poprzez wykorzystanie genotypów ich rodziców.

Spotkanie jest działaniem pozwalającym na wymianę energii. Dzięki niemu agenci o lepszej wartości dopasowania są w stanie pobrać energię od tych z gorszą. Spotkanie sprowadza się do porównania dopasowania dwóch agentów i w jego efekcie transferu energii. Jest to szybki sposób porównania i nagradzania najlepszych rozwiązań. Jak zostało wspomniane wcześniej, ilość energii jest stała w całym systemie.

Wszystkie akcje, jakie agenci mogą pomiędzy sobą wykonać mogą zaistnieć wyłącznie wtedy, gdy agenci sąsiadują ze sobą. Pod koniec każdej z iteracji następuje przemieszczenie się agentów. Pozwala to na uniknięcie ciągłej interakcji tych samych agentów ze sobą.

2.2.3. Wyspy obliczeniowe

Populacją nazywany jest cały zbiór agentów, jej początkowy rozmiar jest parametryzowany i zmienia się w czasie wykonywania programu.

Podczas pojedynczego przebiegu programu można podzielić populację na grupy nazywane wyspami [18], na których populacje (ich liczebność, jakość) zmieniają się niezależnie od siebie w tym samym czasie. Zaistnienie migracji agenta pomiędzy wyspami możliwe jest po przekroczeniu pewnego poziomu jego energii, który jest parametrem wejściowym dla algorytmu.

Poprzez wprowadzenie migracji między różnymi wyspami, możliwe jest eksportowanie pewnych rozwiązań pomiędzy nimi. Ma to pozytywny wpływ na minimalizowanie tendencji populacji do wpadania w lokalne ekstrema poprzez zwiększenie różnorodności. Dobre rozwiązanie opracowane w jednej populacji może być wprowadzone do innej, co sprzyja różnorodności przykładowych rozwiązań.

Istnienie tego typu rozwiązania może wpłynąć na zrównoleglenie, a co za tym idzie - na czas działania algorytmu. Niezależne od siebie wyspy obliczeniowe można bowiem uruchomić na osobnych węzłach obliczeniowych.

3. Algorytmy rojowe

Algorytmy rojowe, w tym algorytm roju cząstek, należą do algorytmów wykorzystujących inteligencję stadną. Nazywane tak są techniki sztucznej inteligencji bazujące na wiedzy o społecznych zachowaniach w samozorganizowanym systemie [30].

Systemy rojowe są najczęściej zbudowane z populacji prostych osobników oddziałujących na siebie nawzajem oraz na środowisko w którym się znajdują. Członkowie populacji nie wykonują skomplikowanych obliczeń, ich wiedza o problemie jest znikoma oraz nie przechowują prawie żadnych informaji. Ponadto nie istnieje scentralizowana struktura kierująca zachowaniem indywidualnych osobników. Jednostki posiadają jedynie wiedzę o akcji jaką powinny podjąć przy danych bodźcach zewnętrznych. Wzajemne oddziaływanie pomiędzy osobnikami prowadzi do złożonych zachowań populacji.

Przykłady takich zachowań są bardzo liczne w naturze. Zaliczyć do nich można kolonie mrówek, klucze ptaków, ławice ryb czy roje pszczół. W przypadkach tych mamy doczynienia z licznymi osobnikami, które wpływając na siebie nawzajem osiągają złożone populacje zdolne realizować skomplikowane zadania.

Algorytmy rojowe dają możliwość szybkiego rozwiązania, jednak ponieważ należą do heurystyk, znalezienie rozwiązania nie jest gwarantowane. Poprzez wykorzystywanie osobników nieposiadających wiedzy o problemie, można wykorzystywać je do rozwiązywania różnych zadań, modyfikując jedynie informację o problemie, jego przestrzeni rozwiązań i funkcji dopasowania.

3.1. Przegląd wiedzy

Pierwszy raz idea wykorzystania zachowań zaobserwowanych w naturze została zaproponowana w 1987 roku [1]. Dzięki wykorzystaniu kilku względnie prostych reguł udało się osiągnąć w animacjach symulowanie bardziej skomplikowanych, realistycznie wyglądających zachowań stada ptaków.

Pojęcie inteligencji stadnej zostało po raz pierwszy użyte w 1989 roku [2] w kontekście zrobotyzowanych systemów komórkowych, natomiast w roku 1995 [3] pojawiła się pierwsza wersja algorytmu roju cząstek (ang. Particle Swarm Optimization (PSO)). Pierwotnym założeniem algorytmu PSO była symulacja społeczeństwa, jednak problem okazał się zbyt złożony dla takiego algorytmu. Odnalazł on jednak zastosowanie w problemach optymalizacji ciągłej.

Wraz ze wzrostem zainteresowania algorytmami rojowymi, pojawiło się wiele badań i publikacji nad różnymi algorytmami. Do najpopularniejszych i najdokładniej opisanych należy zaliczyć takie jak

algorytm mrówkowy (ang. Ant colony optimization (ACO)) [7] czy algorytmy pszczele (ang. Bees algorithm (BA)) [5] i pszczelej kolonii (ang. Artifical bee colony algorithm (ABC)) [4]. Część z algorytmów, mimo że ich pierwotne wersje były dostosowane do rozwiązywania problemów dyskretnych, z czasem zostały zmodyfikowane również do rozwiązywania problemów ciągłych. Przykładem takiego algorytmu może być algorytm kolonii mrówek, którego dostosowanie zostało opublikowane ponad dziesięć lat od zaproponowania pierwotnej wersji [6].

Ze względu na różnorodność otaczającej przyrody kolejne algorytmy inspirowane naturą są nieustannie proponowane i rozwijane. W momencie pisania niniejszej pracy (rok 2015), jednym z nowszych zaproponowanych jest algorytm lwich mrówek (ang. Ant lion optimizer (ALO)) [8], którego działanie oparte jest na mechanizmach polowania lwich mrówek.

3.2. Algorytm roju cząstek

Algorytm roju cząstek naśladuje zachowania stadne. Podczas zdobywania doświadczenia (wiedzy), cząsteczki wzajemnie na siebie oddziałują, jednocześnie przesuwając się w coraz lepszy obszar przestrzeni rozwiązań.

Optymalizacja nie jest wymagająca pod względem zasobów. Zapotrzebowanie na pamięć i czas procesora w każdej iteracji są niskie, a sama zmiana parametrów cząstki odbywa się przy wykorzystaniu podstawowych operatorów matematycznych. Algorytm roju cząstek jest wystarczający dla wielu problemów i nie wymaga skomplikowanych metod używanych na przykład przez algorytmy ewolucyjne.

Każdy osobnik w populacji posiada zestaw mechanizmów warunkujących jego ruch w przestrzeni rozwiązań. Ponadto ma pamięć o najlepszym miejscu w przestrzeni jakie odwiedził oraz wiedzę o położeniu jednostki z najlepszym rozwiązaniem z populacji.

3.2.1. Opis algorytmu

Jeśli przestrzeń poszukiwań jest D-wymiarowa, można zaprezentować i-tą cząsteczkę populacji jako D-wymiarowy wektor 3.1,

$$x_i = (x_{i1}, x_{i2}, x_{i3}, \ldots, x_{iD})^T \tag{3.1}$$

gdzie T oznacza numer iteracji. Najlepszą odwiedzoną pozycję (particle best) i-tej cząsteczczki można oznaczyć jako 3.2.

$$p_{b_i} = (p_{b_{i1}}, p_{b_{i2}}, p_{b_{i3}}, \ldots, p_{b_{iD}}) \tag{3.2}$$

Na podobnej zasadzie, wprowadzając oznaczenie g_b (global best), oznaczona zostaje najlepsza pozycja w przestrzeni rozwiązań w stadzie. Przyjmując takie oznaczenia, każdy osobnik populacji przemieszcza się według równania ruchu 3.3.

$$v_{id}^{n+1} = v_{id}^{n} + (p_{b_{id}}^{n} - x_{id}^{n}) + (g_{b_{id}}^{n} - x_{id}^{n}) \tag{3.3}$$

co skutkuje, że w kolejnych iteracjach jego pozycja jest uaktualniana zgodnie z równaniem 3.4

$$x_{id}^{n+1} = x_{id}^{n} + v_{id}^{n+1} \tag{3.4}$$

gdzie:

d = 1, 2, 3, . . . , D

i = 1, 2, 3, . . . , N

N - rozmiar populacji

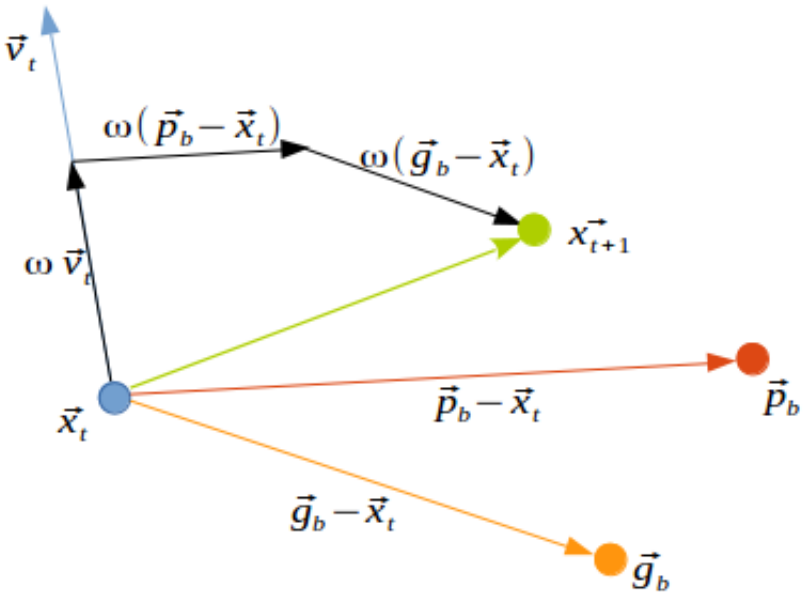
n = 1,2,3, . . . - numer iteracji

Zgodnie z równaniem 3.3, każda nowa pozycja w przestrzeni rozwiązań zależy wyłącznie od poprzednich wartości cząsteczki oraz wartości jej sąsiadów. W celu manipulowania prędkością przesunięcia w konkretnej iteracji, równanie ruchu cząsteczki zostało rozszerzone o współczynnik inercji ω (3.5).

$$v_{id}^{n+1} = \omega(v_{id}^{n} + (p_{b_{id}}^{n} - x_{id}^{n}) + (g_{b_{id}}^{n} - x_{id}^{n})) \tag{3.5}$$

3.2.2. Przemieszczenie

Wizualizację zgodnie z równaniami zamieszczonymi w rozdziale 3.2.1, można prześledzić na rysunku 3.1.



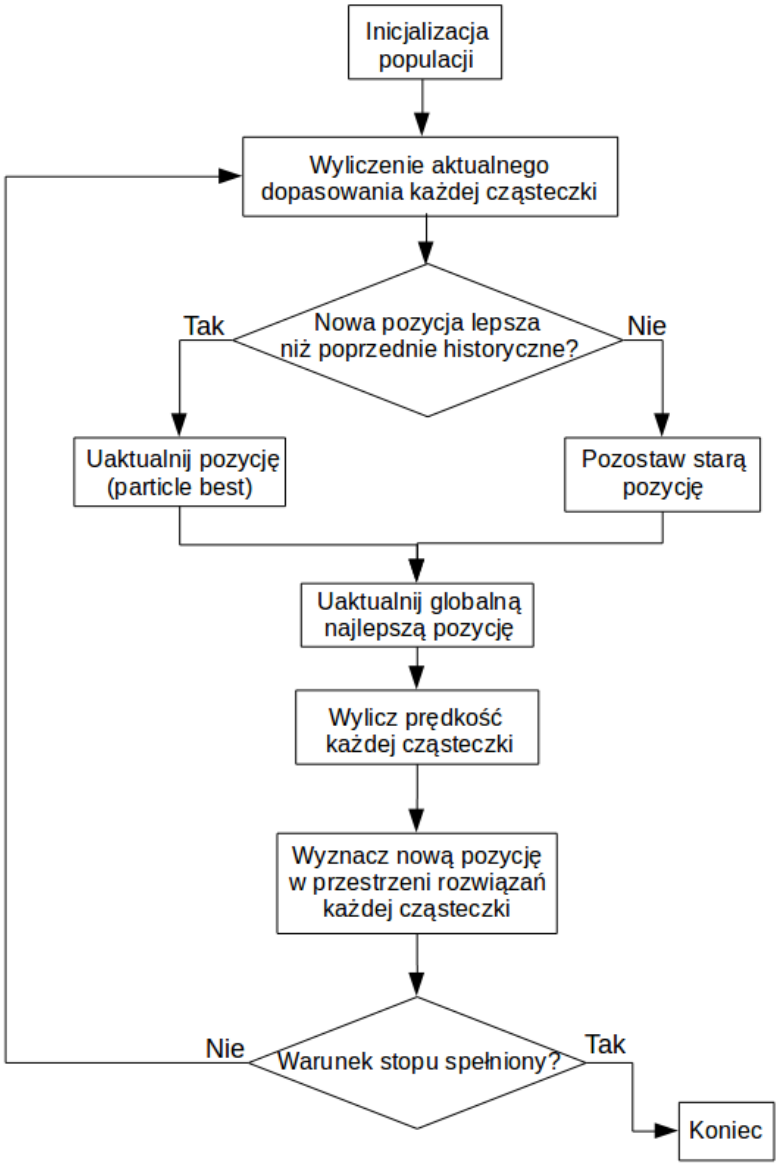
Rysunek 3.1: Wizualizacja ruchu cząstki PSO

W danej iteracji, cząsteczka znajduje się w konkretnym punkcie x_t, zna położenie najlepszej cząsteczki z całej populacji g_b oraz pamięta swoją najlepszą odwiedzoną do tej pory pozycję p_b. Osobnik

wyzacza wektory przesunięcia względem tych punktów oraz przemieszczenia z poprzedniej iteracji v_t. Każdy z wektorów jest mnożony przez współczynnik inercji ω, a następnie wszystkie wektory są ze sobą składane. Wynikiem złożenia jest nowa pozycja cząsteczki w przestrzeni rozwiązań.

3.2.3. Zasada działania algorytmu

Diagram 3.2 przedstawia pełny mechanizm działania algorytmu. Pierwszym krokiem jest zainicjalizowanie startowej populacji w losowych miejscach przestrzeni rozwiązań. Następnie dla każdego osobnika liczone jest dopasowanie, czyli jakość jego rozwiązania. Jeśli aktualne dopasowanie jest lepsze niż zapamiętane, uaktualniana jest pamiętana pozycja w przestrzeni rozwiązań. W przypadku, gdy dopasowanie jest gorsze, zapamiętana informacja pozostaje bez zmian. Kolejnym krokiem jest uaktualnienie informacji najlepszej pozycji z całej populacji. Posiadając wszystkie informacje, cząsteczka wyznacza swoją prędkość w danej iteracji, a następnie przemieszcza się zgodnie z nią w przestrzeni rozwiązań. Aż do spełnienia warunku stopu (uzyskanie pożądanego dopasowania lub osiągnięcia limitu iteracji), cząsteczki ponownie wyliczają swoje dopasowanie i powtarzają cały cykl.



Rysunek 3.2: Diagram blokowy algorytmu PSO

3.3. Modyfikacje algorytmu roju cząstek

W paragrafie 3.2 została przedstawiona podstawowa, najprostsza wersja algorytmu roju cząstek. Istnieje szereg rozszerzeń i modyfikacji algorytmu, które powodują zwiększenie jego dokładności i skuteczności.

3.3.1. Rozszerzenie wiedzy cząstek

Rozszerzając wiedzę jednostek o dodatkowe informacje o otoczeniu, często okazuje się, że zwiększa się dokładność, jak i prędkość w jakiej otrzymywany jest satysfakcjonujący wynik. Jednym ze sposobów,

jest dodanie wiedzy o położeniu najlepszej cząsteczki w pewnym otoczeniu danego osobnika [9]. Dołożenie takiego parametru pociąga za sobą zmianę wyliczania ruchu cząstki (rys. 3.1) o dodatkowy wektor. Podejście takie zwiększa skupienie cząsteczek i pozwala na dokładniejsze przeszukiwanie przestrzeni rozwiązań w danym zakresie.

3.3.2. Dodanie losowej składowej

Uwzględnienie podczas ruchu cząstki dodatkowego składowego wektora, którego wartość oraz kierunek generowane są losowo, daje możliwość pozbycia się pewnych potencjalnych problemów [10]. Cząstki które poruszają się w sposób opisany w rozdziale 3.2.2 oraz te rozszerzone o informacje o sąsiedztwie obarczone są ryzykiem wpadania w lokalne ekstrema. Dodając losową składową, wymuszany jest ruch często oddalający osobnika od roju, co może skutkować wyskoczeniem z lokalnego ekstremum i dalszym przeszukiwaniem przestrzeni rozwiązań w celu znalezienia ekstremum globalnego.

3.3.3. Nadawanie wag parametrom

Podstawowa wersja algorytmu roju cząstek zakłada jednakową wagę każdego z wektorów podczas wyliczania nowej pozycji cząstki. Modyfikacja wprowadzająca dla każdego wektora parametr definiujący jego wagę pozwala na lepsze dostosowanie algorytmu dla danego problemu [11].

3.3.4. Zmiana prędkości ruchu

Algorytm roju cząstek w swojej pierwotnej wersji nie zakładał zmiany prędkości osobników w czasie. Rozszerzenie dające możliwość manipulowania współczynnikiem inercji ω na przestrzeni kolejnych iteracji pozwala na uniknięcie potencjalnego „przeskakiwania" poprawnego rozwiązania przez członków populacji. Wersja podstawowa niesie za sobą ryzyko, że po pewnej ilości iteracji cząsteczki będą krążyły wokół ekstremum, jednak długość wektora będzie zbyt duża, aby udało się im „trafić" w rozwiązanie. Zmniejszanie współczynnika ω wraz z kolejnymi iteracjami pozwoli cząstkom dokładniej przeszukać dany zakres, jednocześnie utrzymując szeroki zasięg przeszukiwań na początku działania algorytmu [12].

4. Wieloagentowy system pyAgE

W roku 1997 została zaproponowana definicja agenta, która definiuje go jako „system, który usytuowany jest w pewnym środowisku, i którego jednocześnie jest częścią; agent obserwuje (odbiera, odczuwa) to środowisko oraz działa w nim: w czasie, według własnego planu, wpływając na to, co będzie mógł zaobserwować w przyszłości" [21]. Definicja powstała na skutek wnikliwej analizy różnych podejść agentowości.

Zgodnie z powyższą definicją, za najistotniejsze cechy agenta należy uznać [20]:

– usytuowanie - agent jest częścią środowiska, w którym się znajduje

– autonomia - agent ma pełną kontrolę nad swoim stanem wewnętrznym oraz akcjami

– reaktywność - agent postrzega środowisko i zmiany zachodzące w nim i reaguje na nie

– zdolności socjalne - agent współdziała z innymi agentami

System agentowy to system komputerowy, którego główną abstrakcją jest pojęcie agenta [22]. System składający się z wielu współdziałających agentów nazywany jest systemem wieloagentowym. Interakcje między agentami, mogące przyjąć formę kooperacji, koordynacji bądź negocjacji są najistotniejszą cechą charakterystyczną tej klasy systemów i stanowią o ich sile.

4.1. Platforma pyAgE

Platformą, na której zostały uruchomione porównywane algorytmy był napisany w języku Python pyAgE [29] [23]. Jest to środowisko wieloagentowe, bazujące na implementacji pod nazwą AgE (ang. Agent-based Evolution). Platforma AgE powstała na podstawe założeń i wymagań przedstawionych powyżej.

Najważniejszą częścią systemu jest dostarczenie mechanizmu wykonywania obliczeń. Podstawowe elementy implementacji systemu stanowią bazę dla realizacji różnej klasy rozwiązań. Ponadto, dzięki odpowiedniej konfiguracji rozwiązania mogą zostać uruchomione jako sieciowa usługa obliczeniowa.

Jak zostało wspomniane wcześniej, podstawową jednostką składową jest agent, gdzie każdy agent jest unikalny w skali całego systemu. W środowisku można wyróżnić również agregaty, które mogą „posiadać" agentów, którzy współdziałają ze sobą. Agregaty zarządzają działaniem podległych im agentów,

między innymi pośredniczą w komunikacji między nimi, nadzorują cykl ich życia czy wykonywanie akcji.

W środowisku dostępne są następujące funkcjonalności:

– zlecenie przez agenta wykonania pewnej akcji

– zapytanie przez agenta o własności innych agentów

– dodanie nowego agenta

– migracja agenta

– śmierć agenta

Platforma została stworzona z myślą o algorytmach ewolucyjnych, między innymi algorytmie EMAS. Dzięki temu dostarcza mechanizmu wysp obliczeniowych, które są jedną ze składowych części algorytmu EMAS, szerzej opisanego w rozdziale 2.2. To właśnie pomiędzy tymi wyspami możliwa jest migracja poszczególnych agentów.

4.2. Rozwinięcie platformy pyAgE

Jednym z celów pracy było wykonanie implementacji pozwalającej na wykonywanie obliczeń za pomocą algorytmów rojowych. W tym celu zostały wykonane komponenty działające na dostarczonej platformie umożliwiające wykorzystanie takiej klasy algorytmów.

W analizowanym podejściu każda cząsteczka jest zastępowana przez pojedynczego agenta. Wymagało to rozszerzenia wiedzy agenta o informacje wymagane przez algorytmy rojowe, takie jak najlepsza pozycja w stadzie czy historycznie najlepsza pozycja. Skutkowało to dodaniem dla każdego agenta struktur przechowujących pewne informacje.

Kolejnym etapem było dodanie konfiguracji uruchomieniowej, uwzględniającej możliwość modyfikacji wszystkich istotnych parametrów. Konfiguracja ta, bazująca na już istniejących, nadała możliwość łatwego i szybkiego porównywania dobieranych parametrów, ale także analizy rozwiązań względem istniejących algorytmów.

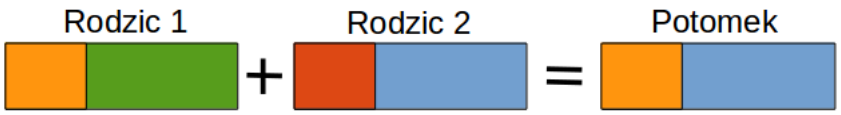
5. Eksperymenty

Wszystkie konfiguracje implementacji zostały przetestowane i porównane na podstawie wielowymiarowej funkcji Rastrigina, opisanej w rozdziale 5.2 oraz przy wykorzystaniu takiego samego sprzętu. Na sprzęcie był zainstalowany system operacyjny Ubuntu 14.04LTS. Obliczenia były wykonywane na procesorze Intel Core 2 Duo CPU T9600 taktowanym częstotliwością 2.80GHz na każdy z dwóch rdzeni i 8GB pamięci RAM DDR3 taktowanej 1333MHz.

Startowa populacja dla wszystkich konfiguracji wynosiła 360 agentów. Warunkiem stopu dla algorytmów było osiągnięcie globalnego minimum (które dla funkcji Rastrigina wynosi 0) lub przekroczenie limitu 3000 iteracji. Wszystkie eksperymenty zostały powtórzone trzydziestokrotnie, a przedstawiane wyniki zostały uśrednione z zawartym odchyleniem standardowym.

Dla algorytmu EMAS, mutacja była wykonywana poprzez losową modyfikację każdego z genów w genotypie. Krzyżowanie odbywało się poprzez podzielenie genotypów rodziców w jednym miejscu, a potomek otrzymywał po jednej części od każdego z rodziców. Można to zaobserwować na rysunku 5.1. Agenci byli zainicjalizowani z energią równą 100, a pozostałe wartości energii prezentowały się następująco:

- śmierć agenta = 0
- minimum dla reprodukcji = 90
- minimum dla migracji = 120
- energia dla nowonarodzonych agentów = 100
- transfer energii = 40



Rysunek 5.1: Krzyżowanie

5.1. Sposób porównywania wyników

W celu porównania jakości algorytmów, podczas obliczeń zbierane były informacje o:

- Wartości funkcji dopasowania - fitness
- Różnorodności agentów w populacji:
  - Różnorodność MSD
  - Różnorodność MOI
- Liczebności populacji
- Czasie obliczeń

Wartość funkcji dopasowania jest głównym kryterium oceny jakości danego algorytmu. Na jej podstawie podczas eksperymentów zostały odrzucane pewne rozwiązania, a inne były rozwijane. Jej wartość informuje jak blisko optymalnego rozwiązania znajduje się rozwiązanie wypracowane przez populację. Im wartość funkcji bliższa zeru, tym jakość rozwiązania jest lepsza.

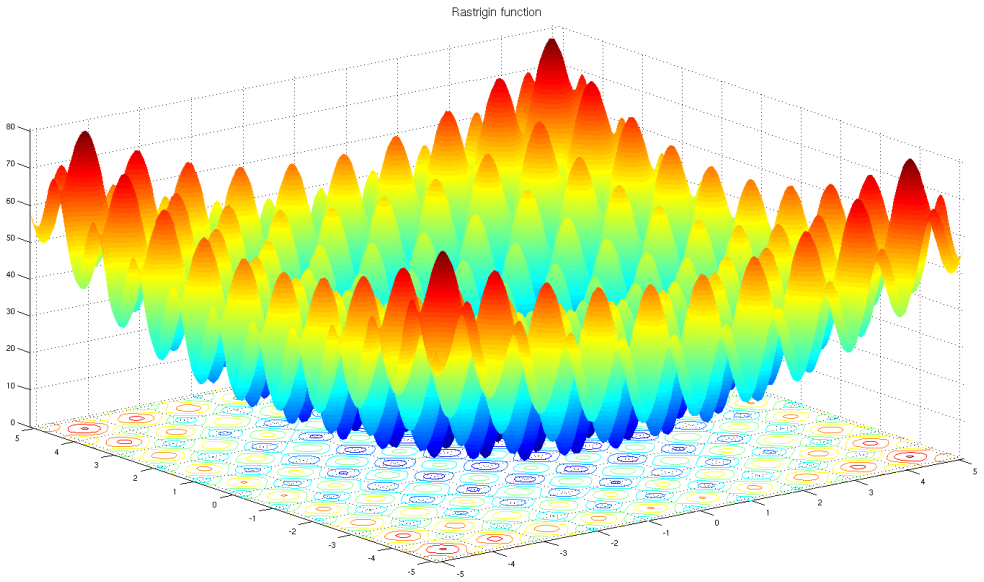
Różnorodność populacji była mierzona za pomocą dwóch kryteriów, MSD oraz MOI, które zostały szczegółowo opisane w rozdziale 5.3. Wartość różnorodności informuje jak bardzo genotyp agentów różni się pomiędzy sobą. Większa różnorodność informuje o szerszym przeszukiwaniu przestrzeni rozwiązań przez populację oraz zmniejsza ryzyko utknięcia w lokalnym ekstremum.

Podczas działania algorytmu EMAS liczebność populacji zmienia się w skutek śmierci i rozmnażania agentów. W przypadku algorytmu roju cząstek populacja początkowa nie zmienia swojej liczebności. W rozdziale 7, w którym porównywane są algorytmy EMAS i PSO, znajdują się informacje o zmianie liczebności populacji w kolejnych iteracjach.

Dla każdej konfiguracji mierzony był czas obliczeń, który prezentowany jest w formie uśrednionego wyniku.

5.2. Funkcja Rastrigina

Funkcja Rastrigina, zaproponowana w 1974 roku [24], używana jest jako problem testu wydajności dla algorytmów optymalizacji. Jest to funkcja nieliniowa. Znalezienie minimum funkcji jest stosunkowo trudnym problemem, ze względu na dużą przestrzeń poszukiwań i istnienie wielu minimów lokalnych, co można zaobserwować na rysunku 5.2 wizualizującym funkcję dla dwóch zmiennych.



Rysunek 5.2: Funkcja Rastrigina dwóch zmiennych [24]

Funkcja została zgeneralizowana do n zmiennych w 1991 roku [25] i jej ogólną wersję można opisać wzorem 5.1.

$$f(x) = An + \sum_{i=1}^{n}(x_i^2 - Acos(2\pi x_i)) \tag{5.1}$$

We wszystkich badaniach porównujących jakość zaimplementowanych rozwiązań używana jest 40-wymiarowa funkcja Rastrigina. Jej parametr A jest równy 10, natomiast wartości przyjmowane przez x należą do przedziału $[-10, 10]$.

5.3. Badanie różnorodności populacji

W celu porównania rozwiązań agentów w populacji zastosowane zostały dwie miary różnorodności: MSD i MOI. Większa różnorodność populacji informuje o szerszym przeszukiwaniu przestrzeni rozwiązań, co pozwala uniknąć utknięcia w lokalnych ekstremach danego problemu.

5.3.1. Różnorodność MSD

Pierwszym sposobem mierzenia różnorodności jest MSD (ang. maximum standard deviation), czyli maksymalne odchylenie standardowe.

Maksymalne odchylenie standardowe każdego genu obliczone dla wszystkich agentów w populacji skupia się na dyspersji średnich wartości obliczonych dla poszczególnych genów [26].

5.3.2. Różnorodność MOI

Drugim rodzajem różnorodności jest różnorodność MOI (Morrison-De Jong) [27]. W sposobie tym pomiar oparty jest na koncepcji momentu bezwładności dla środka ciężkości populacji. Środek ciężkości obliczany jest dla punktów rozproszonych w wielowymiarowej przestrzeni. Podejście takie pozwala na skuteczny pomiar rozkładu masy w dowolnie wielowymiarowych przestrzeniach.

Miara opierająca się o inercję została zdefiniowana wzorem 5.2, gdzie c_j to współrzędne środka masy, natomiast x_{ij} oznacza wartość i-tego genu w j-tym chromosomie.

$$I = \sum_{i=1}^{n} \sum_{j=1}^{N} (x_{ij} - c_j)^2 \tag{5.2}$$

6. Algorytm roju cząstek na platformie PyAGE

W celu znalezienia najlepszej konfiguracji algorytmu roju cząstek został wykonany szereg jego modyfikacji. Został wykonany podstawowy algorytm, bazujący na trzech parametrach:

– najlepsza pozycja w stadzie (ang. global best)

– najlepsza pozycja danej cząsteczki (ang. local best)

– poprzedni wektor przemieszczenia

Następnie algorytm został rozszerzony o możliwość zmiany prędkości cząstek w czasie obliczeń. Kolejnymi krokami było dodanie losowego wektora, o który przemieszcza się dana cząsteczka oraz nadanie wag konkretnym parametrom.

Algorytm w wersji podstawowej, który jest podstawą do porównania wszystkich modyfikacji nie posiadał przemieszczenia o losowy wektor, w czasie trwania obliczeń prędkość cząstek nie zmieniała się, a wszystkie wykorzystywane parametry miały taką samą wagę równą jeden.

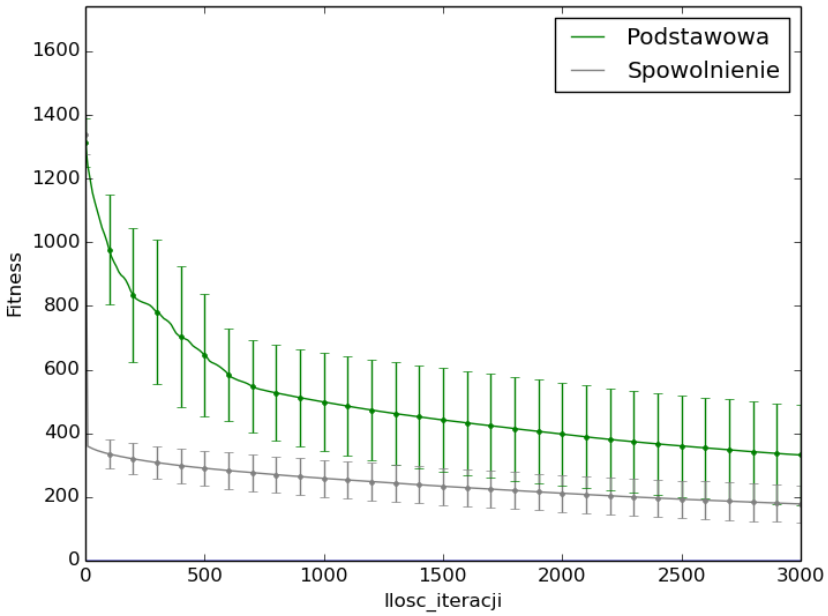
W czasie działania algorytmu liczebność populacji jest stała, żadne akcje wykonywane przez agentów nie prowadzą do ich śmierci ani utworzenia nowego agenta.

6.1. Zmiana prędkości w czasie

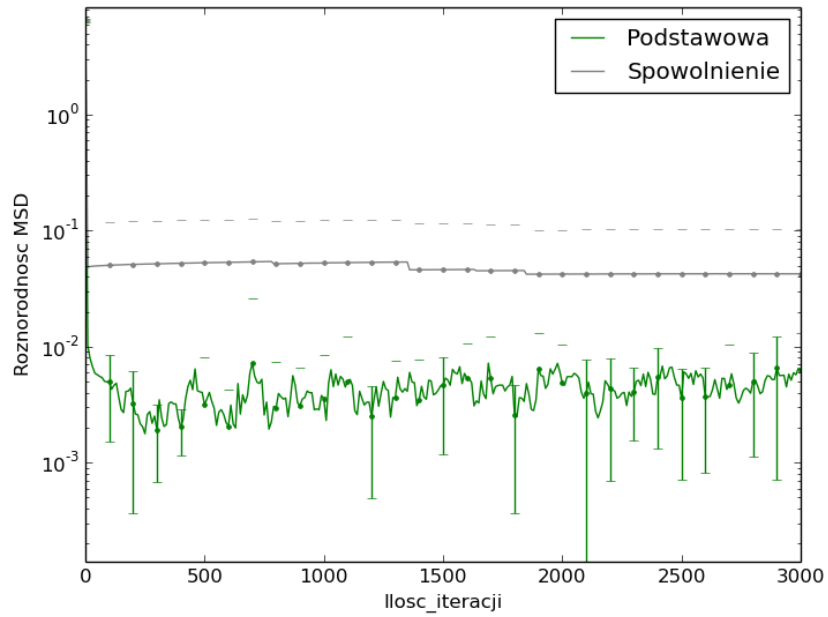
Pierwszą wykonaną modyfikacją była zmiana prędkości cząstek w czasie. Wraz z postępem algorytmu prędkość przemieszczania się cząsteczek mnożona jest o współczynnik ω wyznaczany ze wzoru 6.1, gdzie T jest numerem aktualnej iteracji, a T_{max} ilością iteracji podaną jako warunek stopu.

$$\omega = 1 - \frac{T}{T_{max}} \tag{6.1}$$

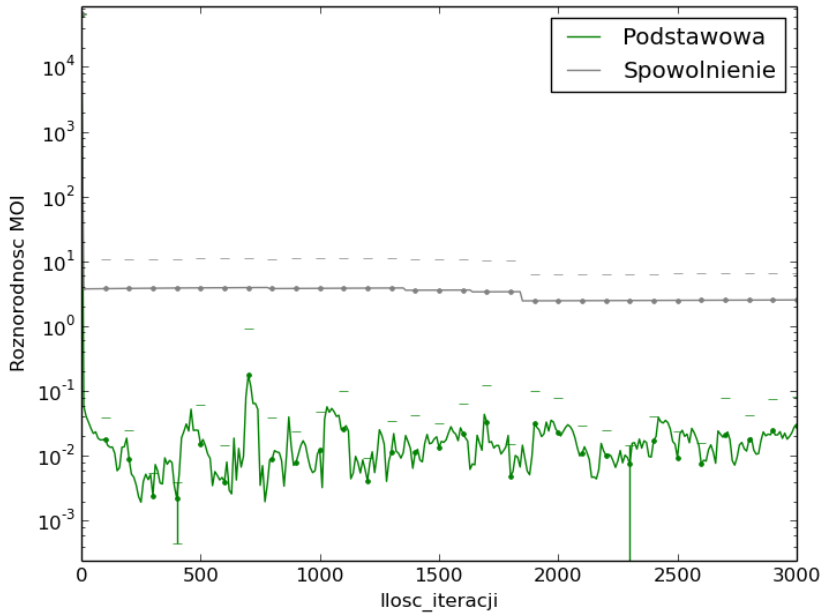
Wykresy 6.1, 6.2 i 6.3 obrazują różnice w wartościach funkcji dopasowania oraz różnorodności opracowanych przez cząsteczki rozwiązań pomiędzy podstawową wersją algorytmu, a tą spowalniającą cząsteczki.



Rysunek 6.1: Porównanie dopasowania przy spowolnieniu cząsteczek



Rysunek 6.2: Porównanie różnorodności MSD przy spowolnieniu cząsteczek



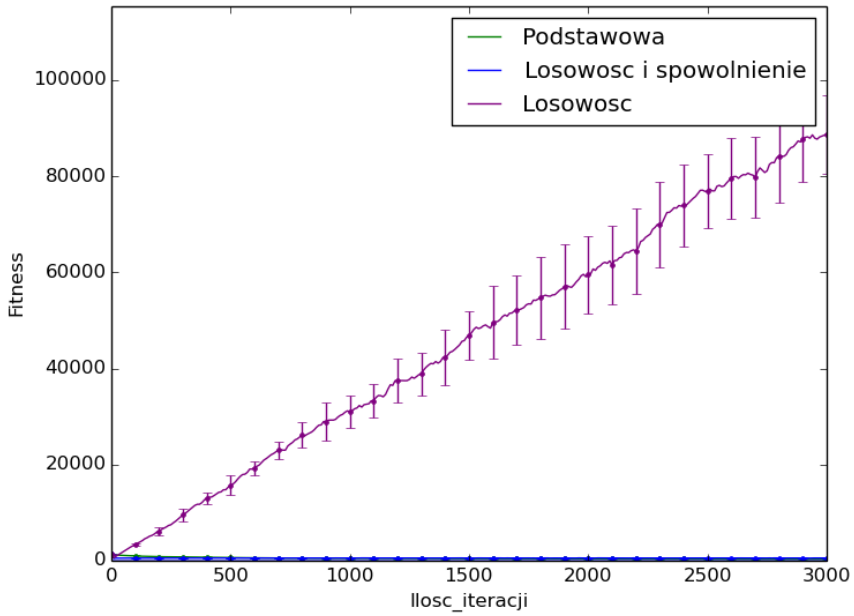
Rysunek 6.3: Porównanie różnorodności MOI przy spowolnieniu cząsteczek

Na przedstawionych wykresach można zauważyć, że dodanie zmiany prędkości w czasie powoduje bisko dwukrotną poprawę jakości rozwiązania. Jednocześnie zmiana ta pozwoliła algorytmowi już w pierwszych iteracjach na znaczące zbliżenie się do poprawnego rozwiązania. Poprzez zastosowanie spowalniania, cząsteczki wykazywały się większą różnorodnością niż bez jego użycia.

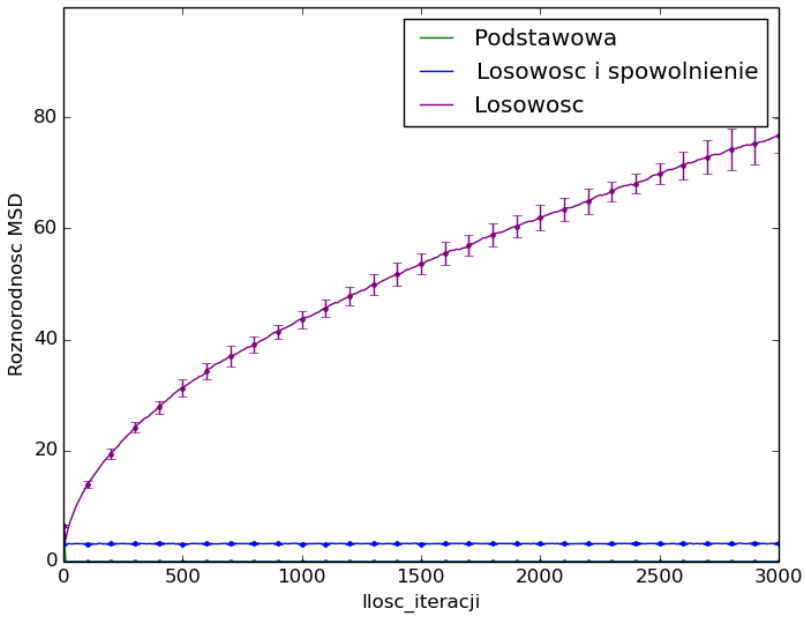
6.2. Dodanie losowego parametru

Innym potencjalnym sposobem polepszenia jakości algorytmu było dodanie losowego składowego wektora, o który przemieściłaby się cząsteczka w przestrzeni rozwiązań. Wykresy 6.4, 6.5 i 6.6 obrazują jakość takiej modyfikacji.

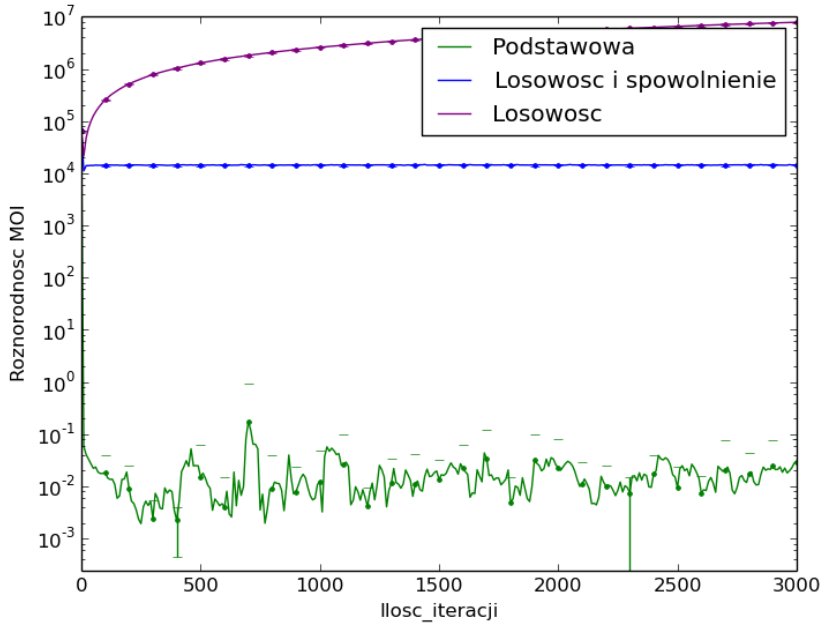
Jak widać na wykresie 6.4, dodanie przesunięcia o losowy wektor bez zastosowania spowalniania cząsteczek spowodowało znaczące pogarszanie się jakości rozwiązania wraz z postępem iteracji. Jednocześnie można zauważyć znaczący wzrost różnorodności, co spowodowane jest wymuszeniem losowego ruchu cząsteczek od siebie z każdym kolejnym krokiem.



Rysunek 6.4: Porównanie dopasowania przy dodaniu losowego wektora



Rysunek 6.5: Porównanie różnorodności MSD przy dodaniu losowego wektora



Rysunek 6.6: Porównanie różnorodności MOI przy dodaniu losowego wektora

6.3. Dodanie wag dla parametrów

Ostatnią z modyfikacji było nadanie wszystkim parametrom wag w taki sposób, aby odzwierciedlić faktyczną ważność danego parametru względem przesuwania się cząsteczki po przestrzeni rozwiązań.

Rysunki 6.7, 6.8 i 6.9 obrazują dopasowanie i różnorodność cząsteczek w przypadku, gdy dla wszystkich parametrów zostały nadane wagi oraz różnicę w zastosowaniu jednoczesnego spowolnienia opisanego w rozdziale 6.1.

Podczas pracy nad algorytmem zostały zbadane eksperymentalnie różne wartości wag dla parametrów - w niniejszej pracy zostały przedstawione te, które dawały najlepsze wyniki.

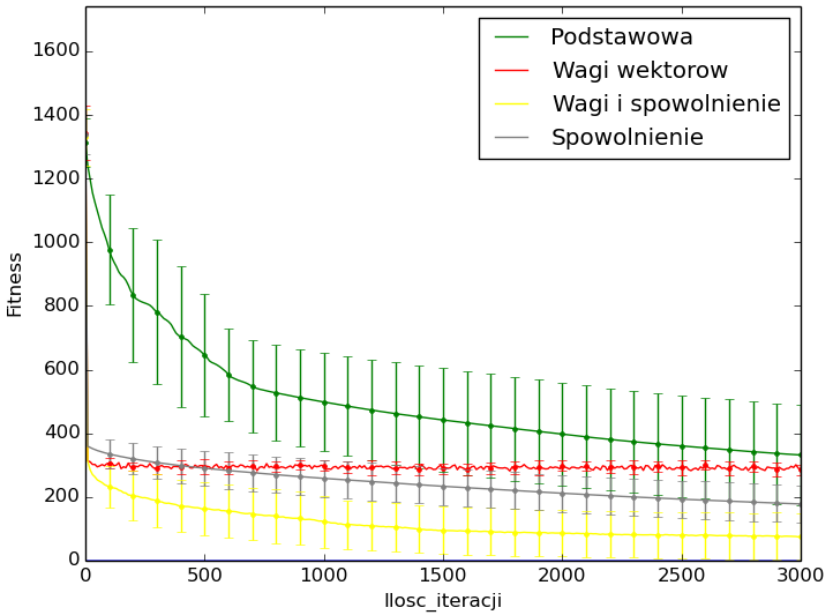
Poszczególnym parametrom zostały nadane następujące wagi:

– pozycja najlepszej cząsteczki z populacji: 0.4

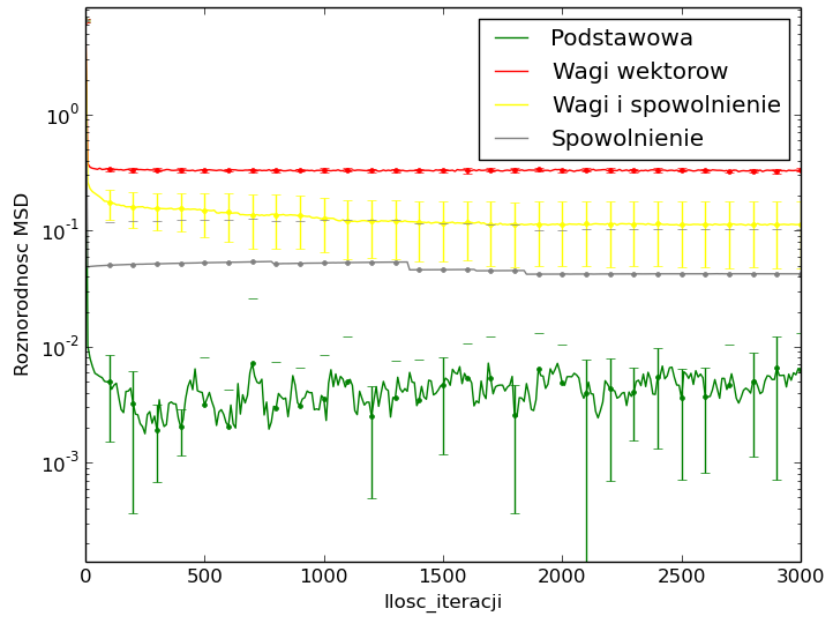
– historycznie najlepsza pozycja danej cząsteczki: 0.2

– poprzednia pozycja cząsteczki: 0.5

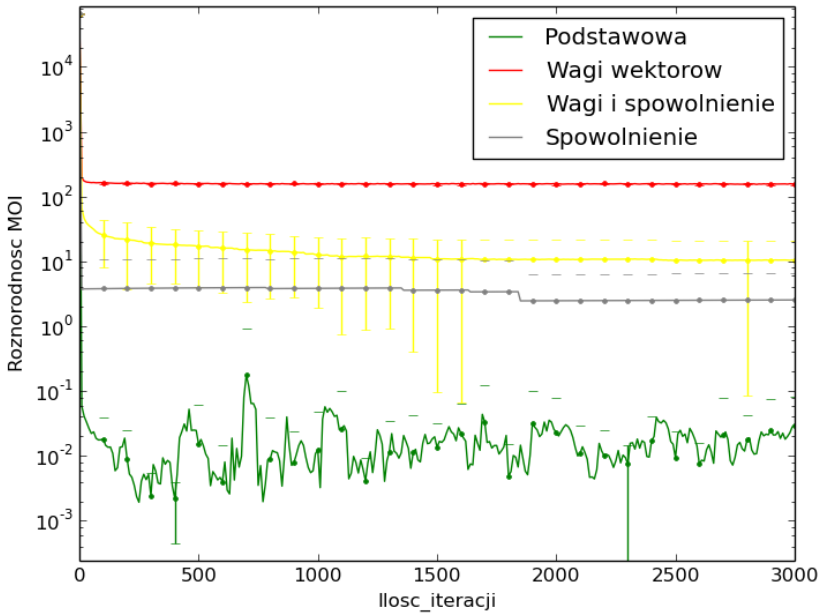
– losowy wektor: 0.1



Rysunek 6.7: Porównanie dopasowania przy dodaniu wag parametrom



Rysunek 6.8: Porównanie różnorodności MSD przy dodaniu wag parametrom



Rysunek 6.9: Porównanie różnorodności MOI przy dodaniu wag parametrom

Wykorzystanie wag parametrów pozwoliło znacząco poprawić jakość rozwiązania w początkowych iteracjach, jednak wraz z postępem algorytmu wynik utrzymywał się na stałym poziomie. Dopiero wykorzystanie wraz z wagami spowolnienia pozwoliło osiągać coraz dokładniejszy wynik w kolejnych iteracjach. Wykorzystanie jedynie wag parametrów zwiększyło również różnorodność pomiędzy agentami.

6.4. Podsumowanie

Jak można zauważyć na wykresie 6.10, najlepsze dopasowanie zostało osiągnięte, gdy zostały połączone ze sobą wszystkie trzy modyfikacje, czyli spowolnienie prędkości cząsteczek w czasie, dodanie losowego wektora oraz nadanie wag wszystkim parametrom.

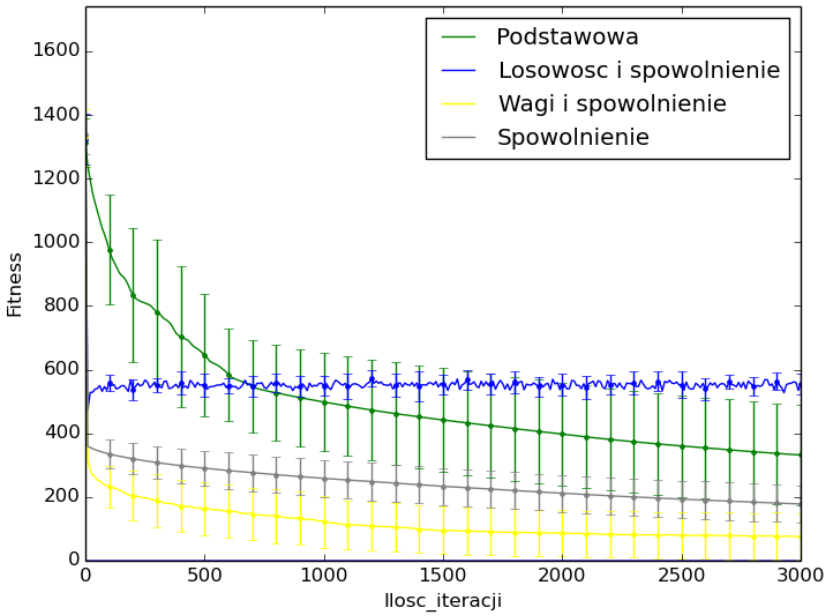
Zastosowanie spowolnienia pozwoliło cząsteczkom na początkowe szybkie, ale niedokładne przeszukiwanie przestrzeni rozwiązań. Wraz z postępem algorytmu ich wolniejsza prędkość przemieszczania pozwoliła na dokładniejsze przeszukiwania w momencie, gdy rój znajdował się bliżej prawidłowego rozwiązania. Jednocześnie pozwoliło to cząsteczkom na uniknięcie „przeskakiwania" prawidłowego rozwiązania, czyli omijania go w przypadku, gdy były już bardzo blisko niego.

Wykorzystanie losowego wektora pozwoliło cząsteczkom na uniknięcie ryzyka pozostania w lokalnym ekstremum. Nadawany dodatkowy kierunek przemieszczenia wymuszał na części roju rozpoczęcie przeszukiwania innego fragmentu przestrzeni rozwiązań.

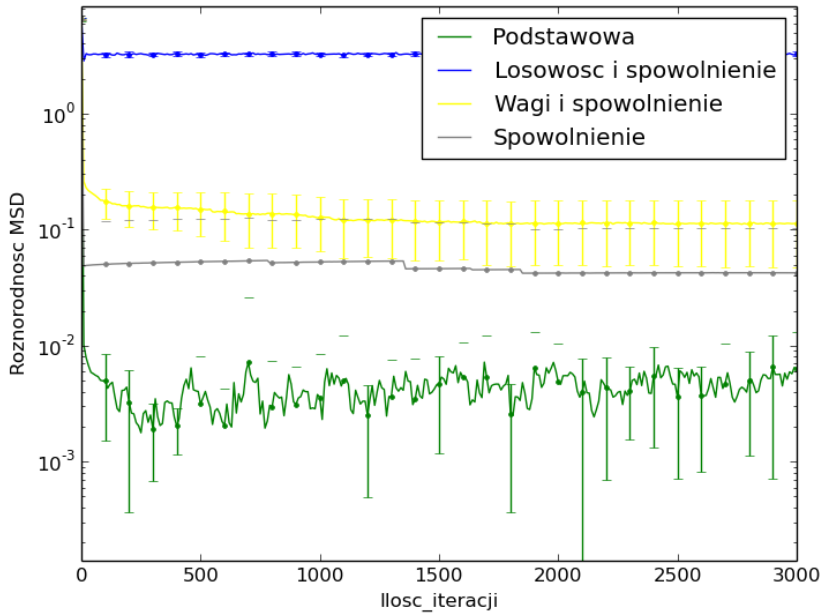
Nadanie wag parametrom rozwiązało problemy zobrazowane na wykresie 6.4, kiedy to losowy wektor zbyt mocno wpływał na ruch cząsteczki i nie pozwalał jej na znalezienie prawidłowego rozwiązania. Zwiększenie wagi najlepszej pozycji stada względem historycznie najlepszej danej cząsteczki pozwoliło na mocniejsze zgrupowanie roju. Jednoczesna wysoka waga poprzedniego ruchu pozwalała cząsteczce na przeszukiwania po własnej ścieżce, eliminując ryzyko zebrania się roju wokół jednego punktu.

Wykres 6.11, obrazujący różnorodność cząsteczek opartą o odchylenie standardowe, oraz 6.12, oparty na momencie bezwładności środka ciężkości układu punktów, obrazują najmniejszą różnorodność cząsteczek w podstawowej wersji algorytmu. Nadawanie wag wektorom bez jednoczesnego ich spowalniania spowodowało największą różnorodność, co jest wynikiem najmniejszej szansy na grupowanie się cząsteczek. Większość analizowanych algorytmów, wraz z postępem iteracji, utrzymuje różnorodność na stałym poziomie, wynika to z faktu scalenia roju od losowego położenia i następnie przemieszczania się już wspólnie.

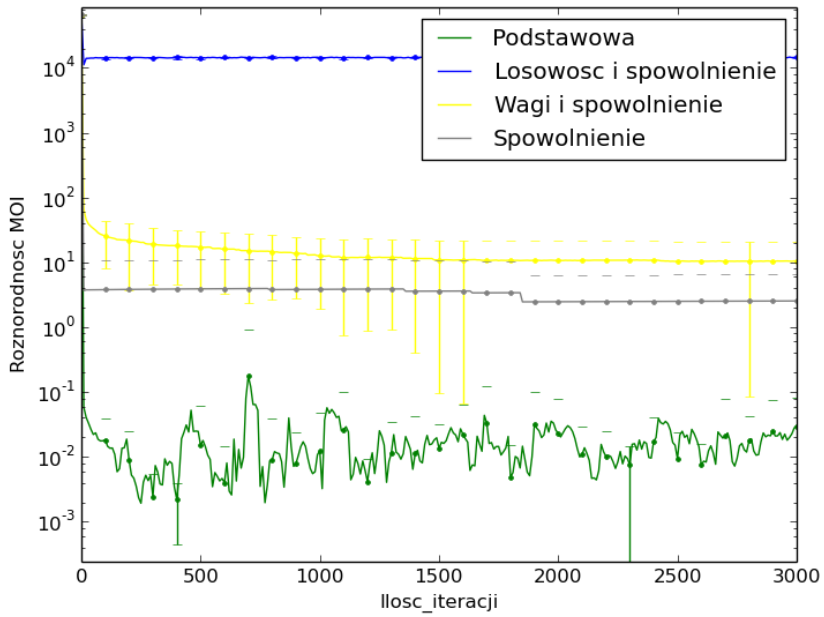
Analizując czasy wykonanania poszczególnych konfiguracji zobrazowanych na wykresie 6.13 można zauważyć, że wraz z dokładaniem kolejnych składowych algorytmu czas ich wykonywania rośnie. Różnica między najszybciej wykonującą się konfiguracją (podstawową), a najdłużej wykonującą się (z dodanym losowym wektorem, wagami parametrów i zmianą prędkości) wynosi jedynie niecałe trzy sekundy, co stanowi około 7%. Jednocześnie różnica w jakości rozwiązania jest prawie dwukrotna na korzyść wolniejszego z nich.



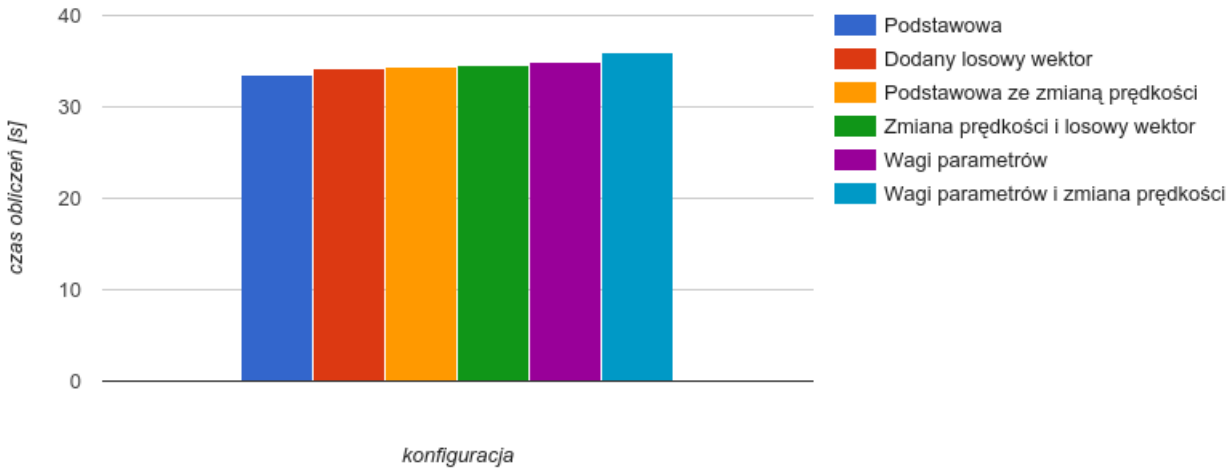
Rysunek 6.10: Porównanie dopasowania najlepszych konfiguracji



Rysunek 6.11: Porównanie różnorodności MSD najlepszych konfiguracji



Rysunek 6.12: Porównanie różnorodności MOI najlepszych konfiguracji



Rysunek 6.13: Porównanie czasu wykonania dla każdej konfiguracji

7. Porównanie algorytmów ewolucyjnego i roju cząstek

Najlepsza konfiguracja algorytmu rojowego, która została wybrana na podstawie eksperymentów opisanych w rozdziale 6 została poddana dalszym modyfikacjom. Nowopowstałe konfiguracje wykorzystywały takie elementy platformy pyAgE jak sąsiedztwo agentów czy wyspy obliczeniowe.

W rozdziałach od 7.2 do 7.6 znajdują się porównania jakości algorytmów w zależności od ilości wysp obliczeniowych. Rozmiar populacji początkowej we wszystkich eksperymentach wynosił 360 agentów, którzy w momencie wykorzystywania wysp obliczeniowych byli równomiernie między nimi rozdzielani.

Konfiguracja algorytmu EMAS dostarczona przez platformę pyAgE widnieje na wykresach jako "femas single", natomiast podstawowa konfiguracja algorytmu roju cząstek wypracowana na podstawie eksperymentów - jako "PSO basic".

7.1. Modyfikacje algorytmu PSO z wykorzystaniem platformy pyAgE

Zostały wykonane dwie modyfikacje algorytmu roju cząstek w oparciu o mechanizmy i gotowe komponenty platformy pyAgE.

7.1.1. Połączenie algorytmów EMAS i PSO

Pierwszą modyfikacją było połączenie kroków algorytmu ewolucyjnego z krokami algorytmu rojowego. Po zastosowaniu tej modyfikacji każdy agent w pojedynczej iteracji na początku wykonywał wszystkie akcje wynikające z mechanizmów algorytmu EMAS, po czym przemieszczał się po przestrzeni rozwiązań zgodnie z mechanizmami algorytmu roju cząstek.

Takie podejście powodowało modyfikację genotypu danego agenta i pozwalało nawet tym agentom z niskim poziomem energii po czasie ją odzyskać. W związku z tym oczekiwane było polepszenie jakości rozwiązania wypracowanego przez taką populację, jednak kosztem czasu wykonywanych obliczeń. Modyfikacja ta opisywana jest na wykresach jako "Femas PSO"

7.1.2. Wykorzystanie sąsiedztwa agentów w środowisku

Drugim elementem dostarczonym przez platformę, a użytym do modyfikacji algorytmu, było wykorzystanie sąsiedztwa agentów wewnątrz wysp obliczeniowych. Położenie agenta wewnątrz wyspy w podstawowej wersji algorytmu rojowego nie powodowało modyfikacji jego genotypu. Algorytm ewolu-

cyjny wykorzystywał położenie agenta wewnątrz wyspy do obsługi interakcji pomiędzy agentami, która może zaistnieć tylko w przypadku, gdy agenci bezpośrednio sąsiadują ze sobą.

Siatka, na której rozmieszczeni są agenci, opisana jest na torusie. Zastosowanie takiej figury eliminuje problemy związane z napotkaniem brzegu siatki przez agenta oraz daje możliwość stosowania tych samych zasad sąsiedztwa na całej powierzchni.

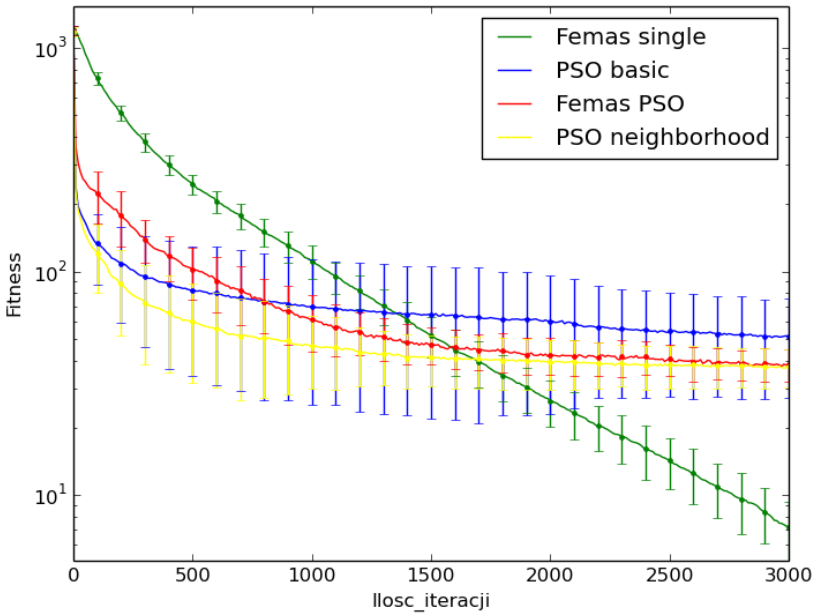
Wykonana modyfikacja wykorzystywała sąsiedztwo agentów wewnątrz wysp obliczeniowych. Najlepszy genotyp wśród sąsiadujących agentów był dodawany jako kolejna składowa powodującą przesunięcie się cząsteczki po przestrzeni rozwiązań. Dodatkowemu parametrowi została eksperymentalnie nadana waga równa 0.4, która skutkowała najlepszą wartością funkcji dopasowania.

Spodziewanym efektem tej modyfikacji był nieznaczny wzrost czasu obliczeń, jednak rekompensowany przez drobną poprawę jakości wypracowanego przez populacje rozwiązania. Na ponizszych wykresach modyfikacja została opisana jako „PSO neighborhood”.

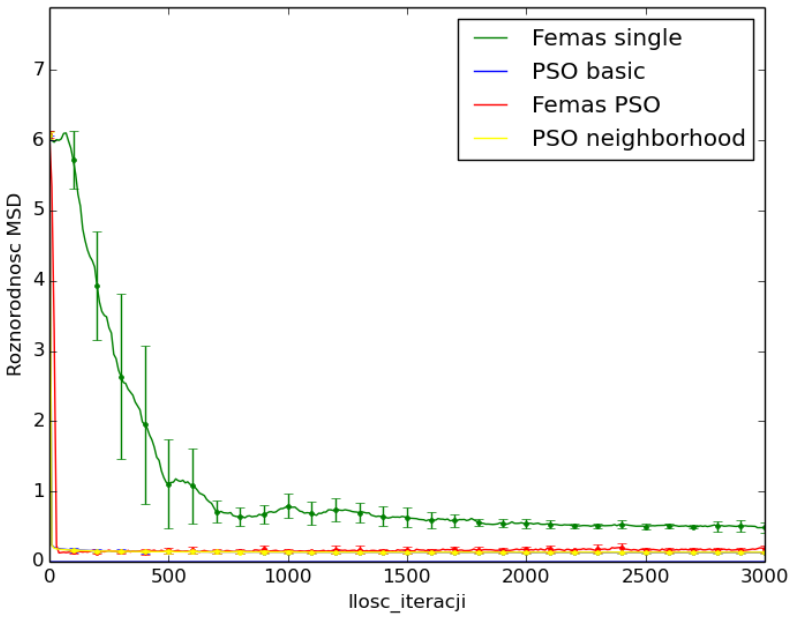
7.2. Jedna wyspa obliczeniowa

Wyresy od 7.1 do 7.4 prezentują wyniki zebrane w przypadku wykorzystania jednej wyspy obliczeniowej. Na wykresie 7.4 można zwrócić uwagę, że algorytmy ewolucyjne przez pewną ilość iteracji zmniejszają swoją populację, po czym stabilizują się na stałym poziomie. Algorytm łączący w sobie kroki ewolucyjne i rojowe również wykazuje podobne zachowanie.

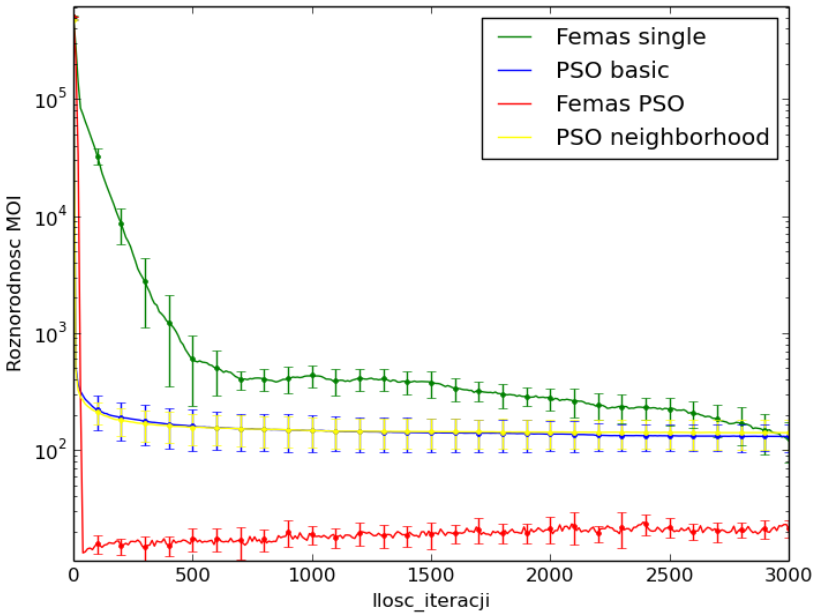
Można zauważyć, że algorytm EMAS w początkowych krokach osiąga dużo niższe wartości dopasowania niż pozostałe algorytmy. Od pewnego momentu jednak poprawa jakości rozwiązań pozostałych algorytmów utrzymuje się na podobnym poziomie, podczas gdy EMAS nadal udoskonala swoje rozwiązanie.



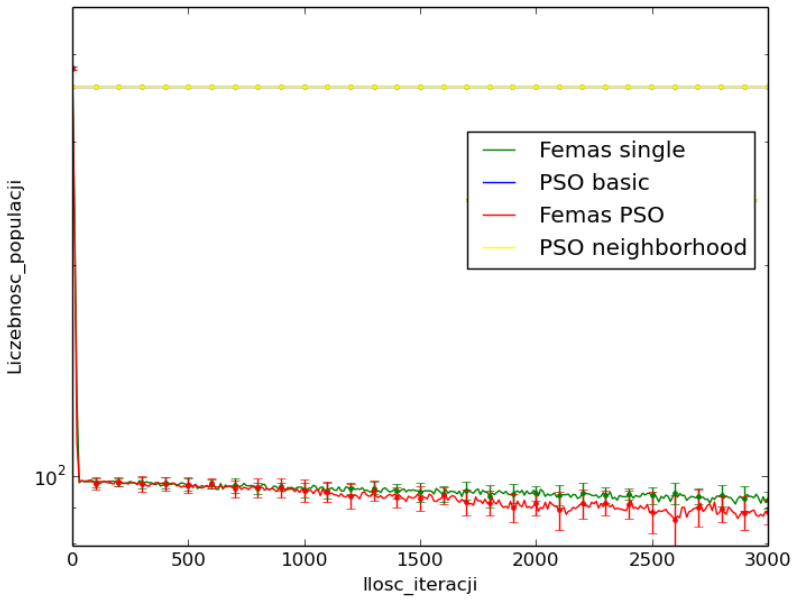
Rysunek 7.1: Porównanie dopasowania przy jednej wyspie



Rysunek 7.2: Porównanie różnorodności MSD przy jednej wyspie



Rysunek 7.3: Porównanie różnorodności MOI przy jednej wyspie

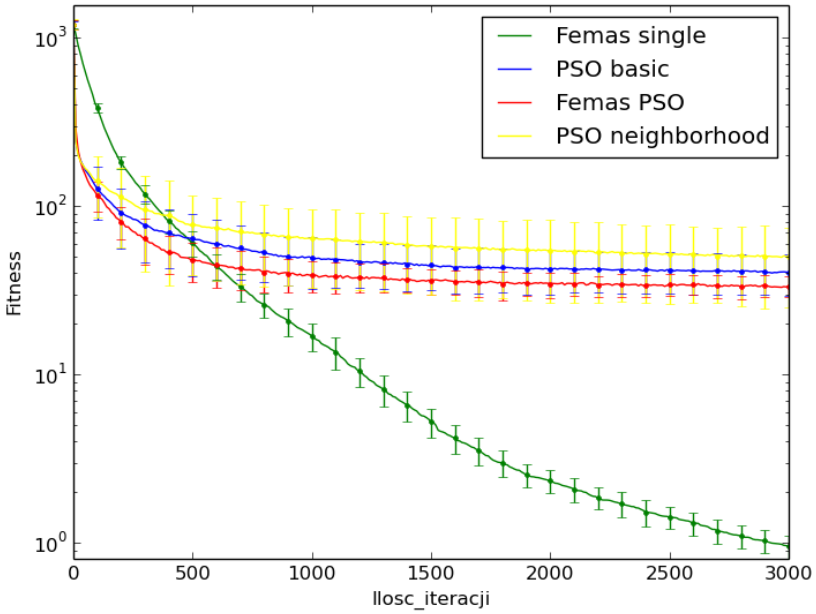


Rysunek 7.4: Porównanie liczebności przy jednej wyspie

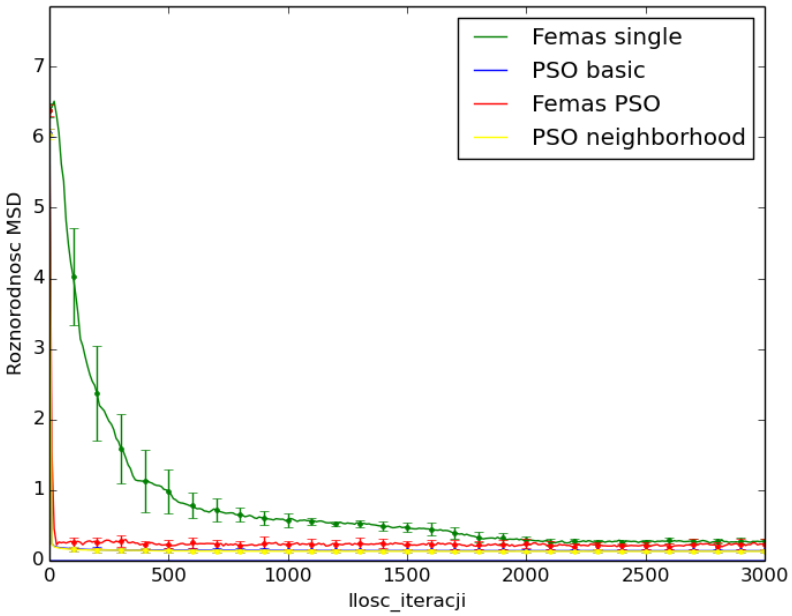
7.3. Trzy wyspy obliczeniowe

Podział populacji na trzy wyspy obliczeniowe (wyniki którego można zaobserwować na wykresach 7.5, 7.6, 7.7 i 7.8) spowodował zwiększenie różnorodności populacji. Wzrost ten wynika to z faktu, że dzięki takiemu mechanizmowi agenci w algorytmach ewolucyjnych zmieniają się w trzech bardziej hermetycznych grupach, algorytmy rojowe natomiast, zachowują się jak trzy całkowicie niezależne populacje.

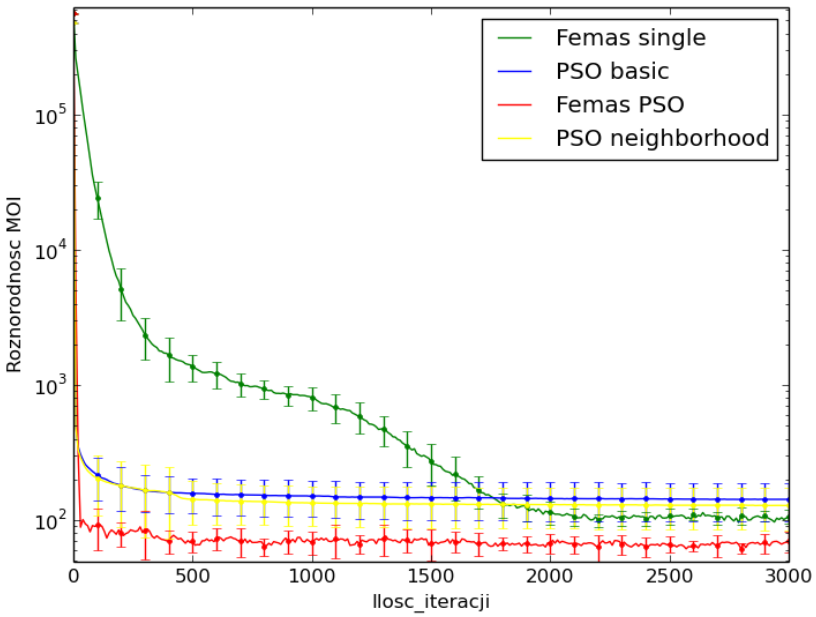
Zauważalny jest wzrost prędkości z jaką algorytm EMAS osiąga bardzo dobre wyniki dopasowania. Już po wykonaniu połowy kroków założonych jako warunek stopu osiągnięty został wynik bliski minimum. Dla pozostałych algorytmów zmiana nie jest aż tak radykalna, jednak również ich jakość została poprawiona.



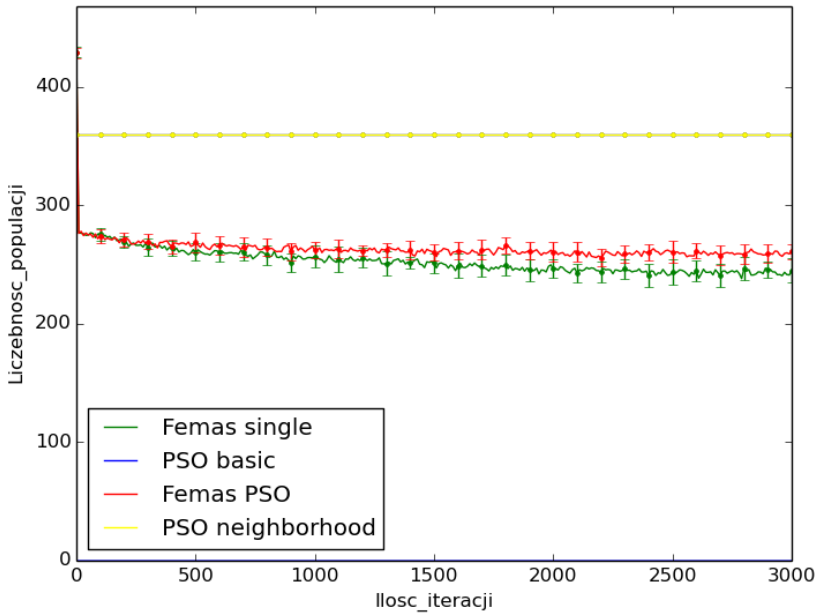
Rysunek 7.5: Porównanie dopasowania przy trzech wyspach



Rysunek 7.6: Porównanie różnorodności MSD przy trzech wyspach



Rysunek 7.7: Porównanie różnorodności MOI przy trzech wyspach

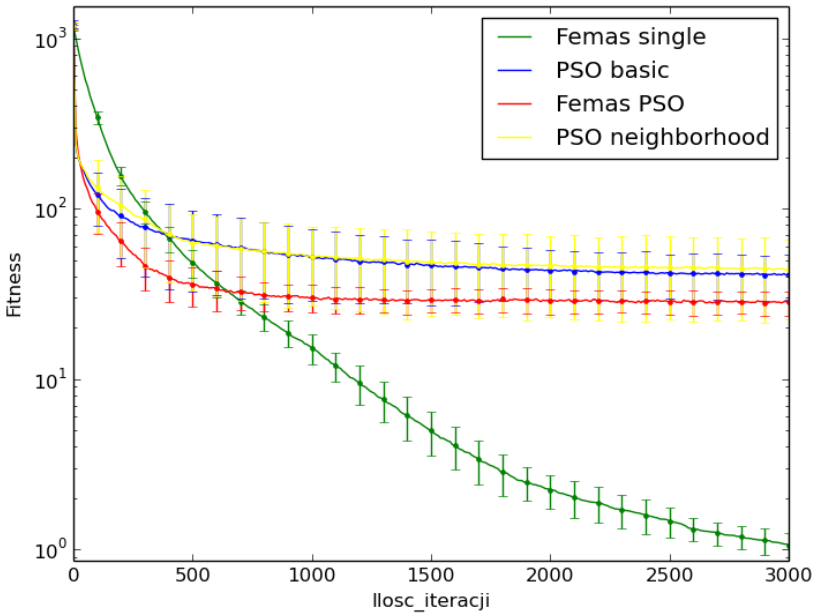


Rysunek 7.8: Porównanie liczebności przy trzech wyspach

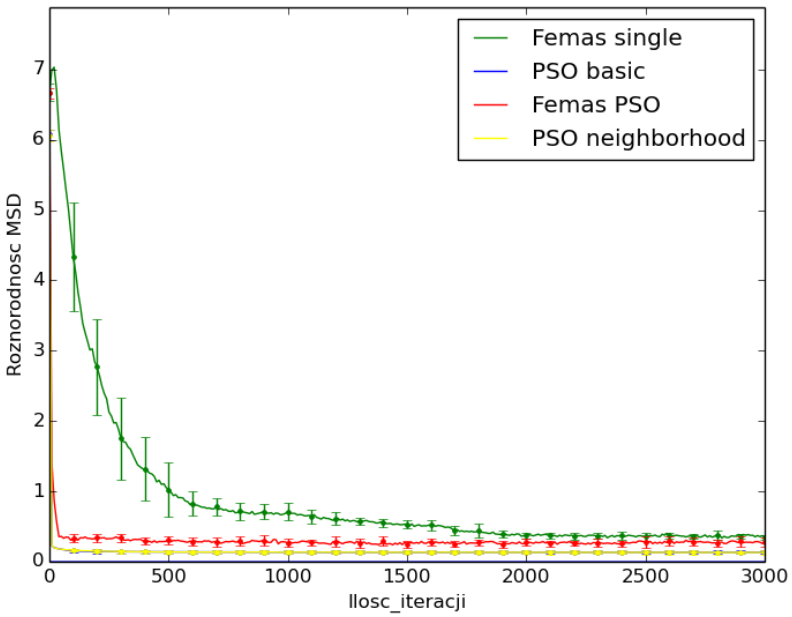
7.4. Sześć wysp obliczeniowych

Wraz ze wzrostem ilości wysp obliczeniowych nadal zwiększa się różnorodność populacji, co można zaobserwować na wykresac 7.10 i 7.11. Różnica w jakości rozwiązania podstawowego algorytmu EMAS (co obrazuje wykres 7.9) nie jest już aż tak zauważalna jak przy przejściu z jednej do trzech wysp obliczeniowych.

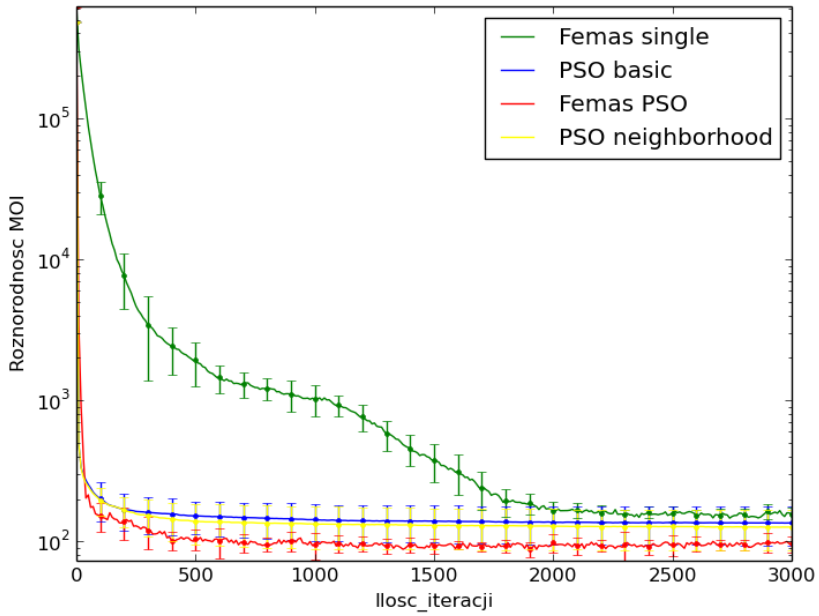
Zmianę liczebności populacji można zaobserwować na wykresie 7.12. Algorytm wykorzystujący połączenie kroków ewolucyjnych i rojowych po początkowych zmianach w pierwszych iteracjach utrzymuje stałą liczebność zbliżoną do startowej.



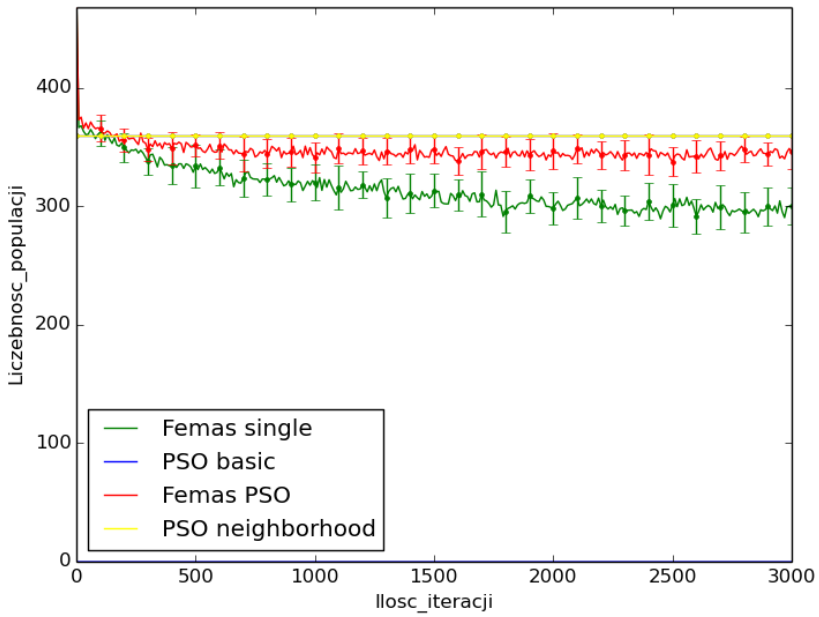
Rysunek 7.9: Porównanie dopasowania przy sześciu wyspach



Rysunek 7.10: Porównanie różnorodności MSD przy sześciu wyspach



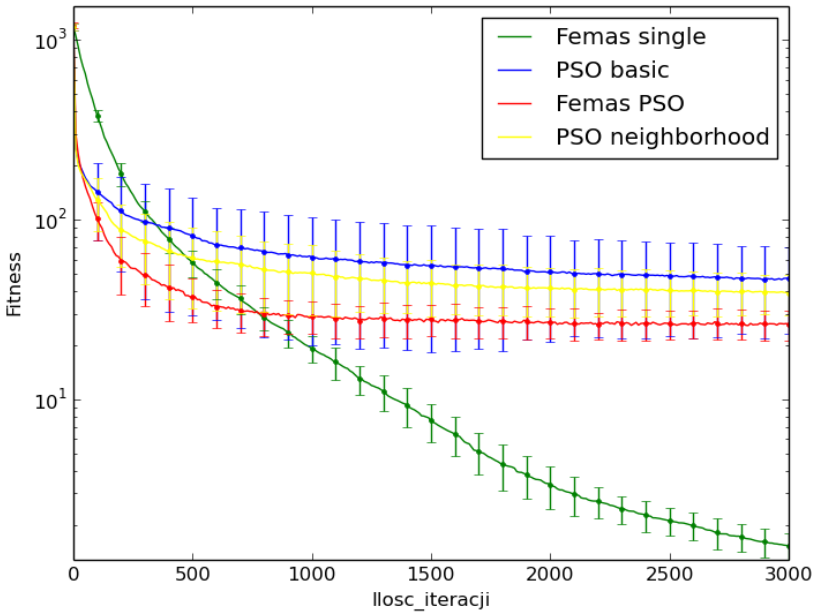
Rysunek 7.11: Porównanie różnorodności MOI przy sześciu wyspach



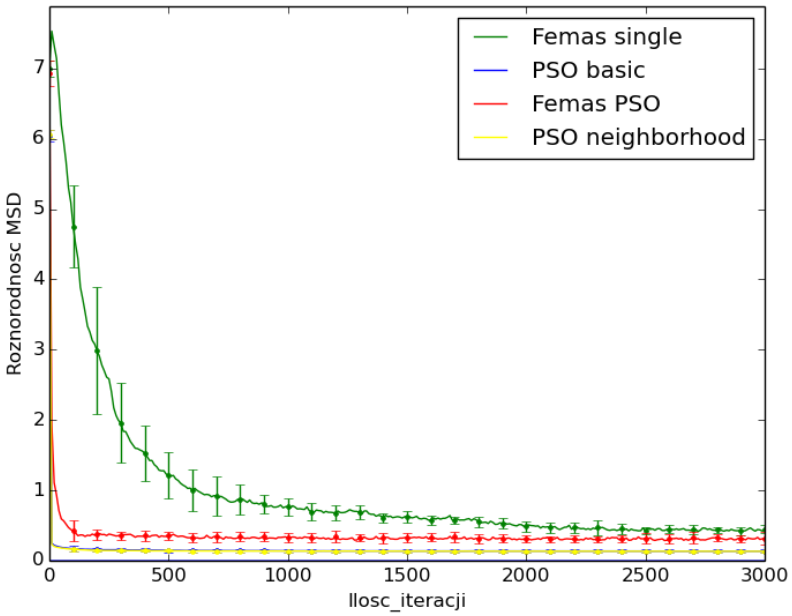
Rysunek 7.12: Porównanie liczebności przy sześciu wyspach

7.5. Dziewięć wysp obliczeniowych

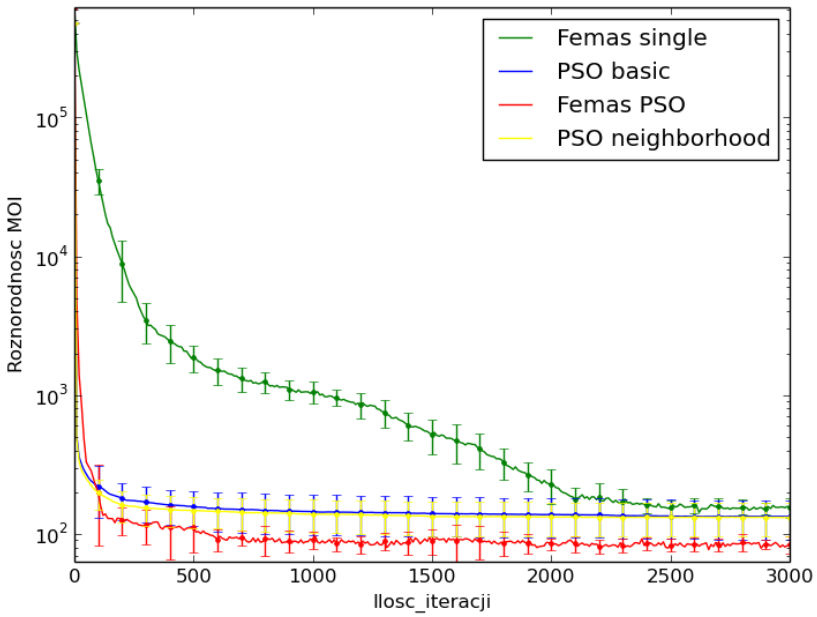
Dalsze zwiększanie ilości wysp obliczeniowych skutkuje poprawą wyniku algorytmu łączącego kroki ewolucyjne i rojowe. Jednocześnie zauważalny jest ciągły wzrost różnorodności populacji we wszystkich badanych algorytmach. Zmienianie się mierzonych wartości wraz z postępem iteracji można zaobserwować na wykresach 7.13, 7.14, 7.15 i 7.16.



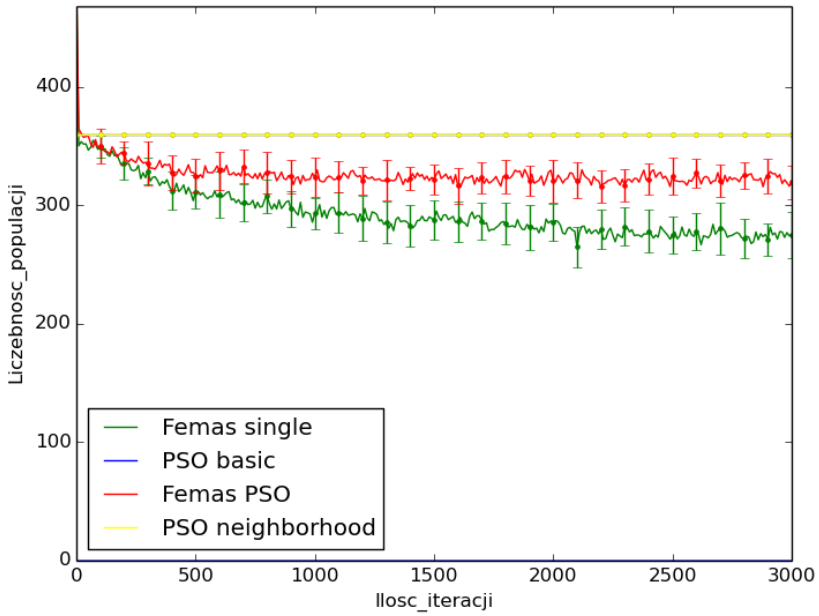
Rysunek 7.13: Porównanie dopasowania przy dziewięciu wyspach



Rysunek 7.14: Porównanie różnorodności MSD przy dziewięciu wyspach



Rysunek 7.15: Porównanie różnorodności MOI przy dziewięciu wyspach

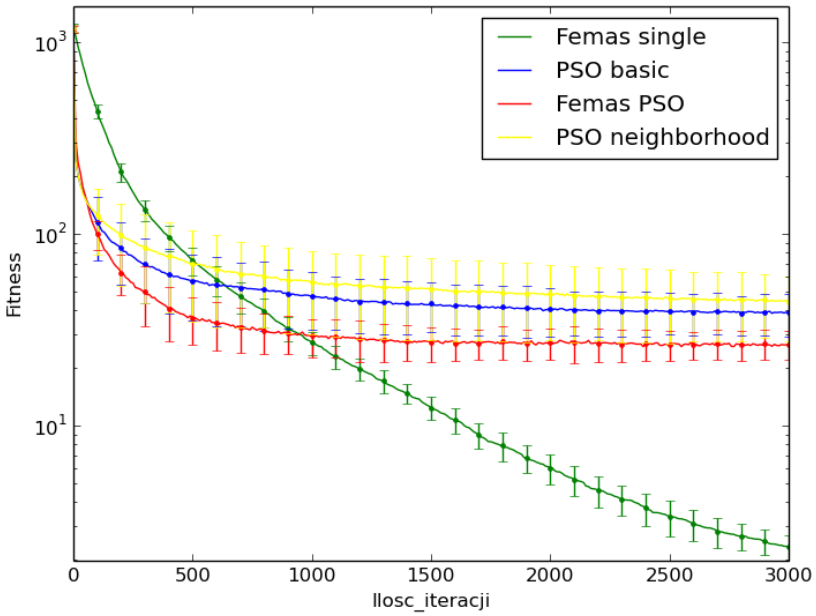


Rysunek 7.16: Porównanie liczebności przy dziewięciu wyspach

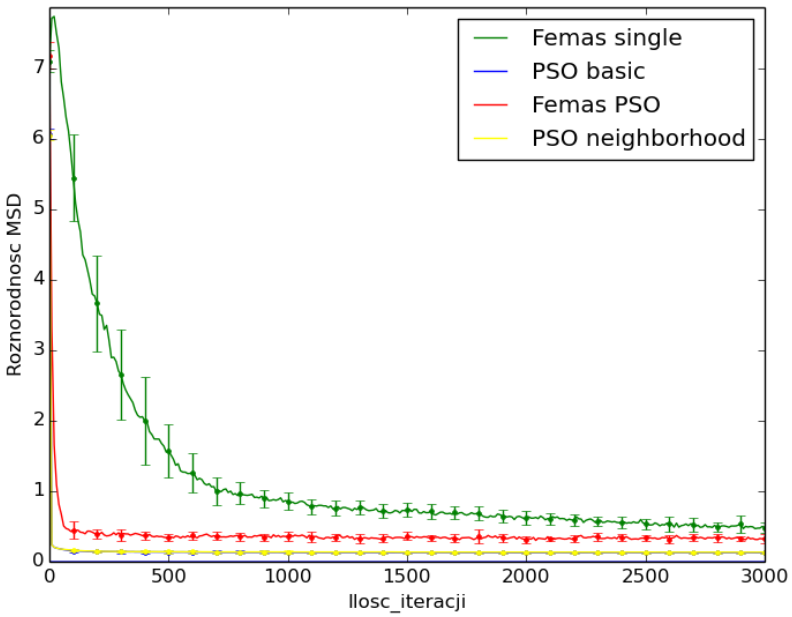
7.6. Dwanaście wysp obliczeniowych

Ostatnią porównywaną ilością wysp obliczeniowych jest dwanaście, oznacza to, że na każdą z wysp przypada na początku 30 agentów. Wartość dopasowania poszczególnych algorytmów znajduje się na wykresie 7.17.

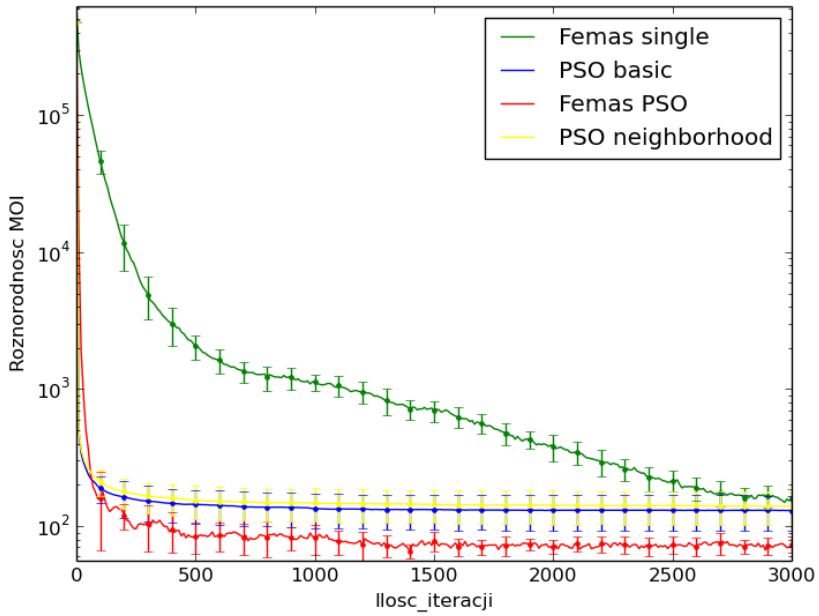
Ponadto, można zauważyć bardzo wysoką różnorodność (wykresy 7.18 i 7.19) - zwiększenie ilości wyp obliczeniowych do dwunastu spowodowało większe zmniejszenie liczebność populacji w przypadku algorytmów wykorzystujących operacje ewolucyjne (wykres 7.20), niż miało to miejsce w przypadku mniejszej liczby wysp.



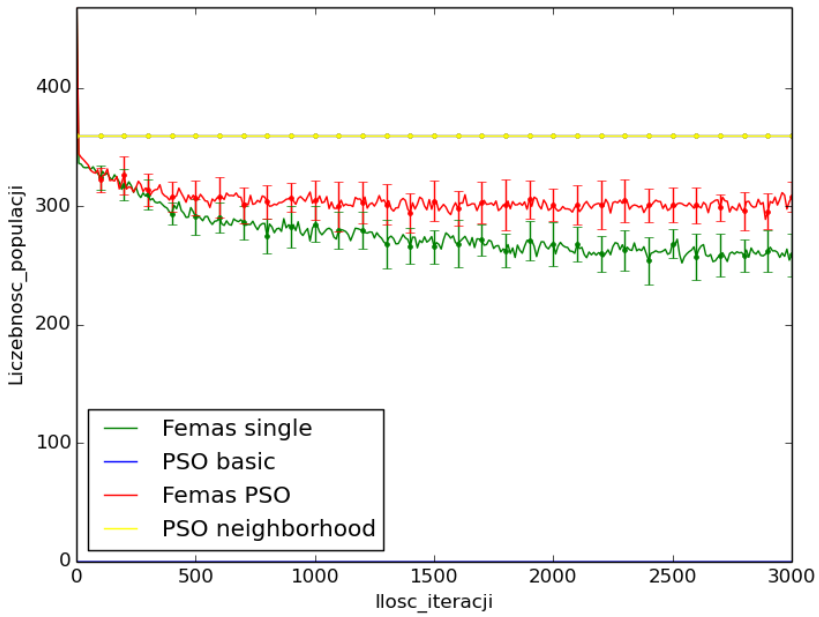
Rysunek 7.17: Porównanie dopasowania przy dwunastu wyspach



Rysunek 7.18: Porównanie różnorodności MSD przy dwunastu wyspach



Rysunek 7.19: Porównanie różnorodności MOI przy dwunastu wyspach



Rysunek 7.20: Porównanie liczebności przy dwunastu wyspach

7.7. Podsumowanie

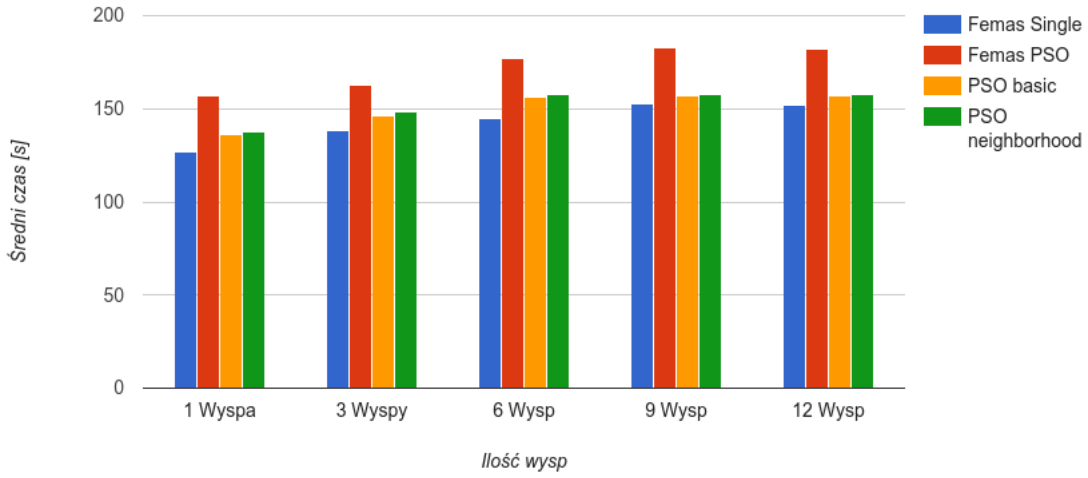
Na wykresie 7.21 przedstawione zostały średnie czasy wykonania dla każdej z analizowanych ilości wysp obliczeniowych. Zauważalny jest narzut wynikający z migracji pomiędzy wyspami, jednak wraz ze wzrostem ilości wysp niwelowany jest on przez zmniejszenie ilości agentów na każdą wyspę, a co za tym idzie, zmniejszeniem ilości akcji wykonywanych.

Spodziewanym rezultatem jest dłuższy czas wykonywania algorytmu zawierającego dodatkowy wektor przesunięcia cząsteczki w przestrzeni rozwiązań, ponieważ wymaga to dodatkowego porównania i obliczenia na każdą cząsteczkę w każdej iteracji. Jest to zauważalne na wykresie, gdzie wyraźnie widać, że czas wykonania algorytmu wykorzystującego sąsiedztwo jest nieznacznie większy od podstawowego algorytmu rojowego.

Algorytm wykorzystujący kroki algorytmu ewolucyjnego przed rojowymi zauważalnie wyróżnia się dłuższym czasem wykonania, jednak jednocześnie idzie za tym jego lepsza jakość niż pozostałych algorytmów wykorzystujących rój. Obserwacja ta potwierdziła tezę postawioną na początku rozdziału, mówiącą że połączenie tych dwóch algorytmów miało skutkować lepszą jakością rozwiązania kosztem czasu wykonywania.

Zauważalny jest znaczący wzrost różnorodności populacji zależnie od ilości wysp obliczeniowych, jest to efekt spodziewany, którego źródłem jest rozdzielenie grup agentów od siebie i ich częściowo niezależne przeszukiwanie przestrzeni rozwiązań. Dzięki rozdzieleniu agenci wpływają na sobie w mniejszych grupach, przez co są w stanie przeszukać różne częsci przestrzeni rozwiązań.

Wykorzystanie dodatkowego parametru w postaci sąsiedztwa poprawiło jakość rozwiązania przy jednoczesnym niewielkim wzroście czasu obliczeń. Wykorzystanie większej ilości wysp obliczeniowych nie skutkowało znaczącą poprawą jakości, jak miało to miejsce w przypadku algorytmów wykorzystujących kroki ewolucyjne.



Rysunek 7.21: Porównanie czasu obliczeń

8. Podsumowanie wyników

Wszystkie założone cele pracy, opisane w rozdziale 1.1, zostały osiągnięte. Do platformy pyAgE zostały dodane komponenty umożliwiające dokonanie obliczeń za pomocą algorytmów rojowych.

Został przeanalizowany szereg konfiguracji i rozszerzeń algorytmu roju cząstek. Szczegółowe wyniki porównujące jakość zaimplementowanego algorytmu pod kątem różnych jego modyfikacji znajdują się w rozdziale 6.

Spośród sprawdzanych rozwiązań zostało wybrane dające najlepszą wartość dopasowania, a następnie porównane z istniejącą implementacją algorytmu EMAS. Aby w pełni wykorzystać platformę obliczeniową, zostały wykonane modyfikacje algorytmu roju cząstek w oparciu o elementy przez nią dostarczane. W rozdziale 7 znajduje się porównanie jakości rozwiązań otrzymywanych poprzez wykorzystanie takich modyfikacji.

Wykonane analizy skuteczności algorytmu roju cząstek wykazały, że wykorzystanie wieloagentowej platformy do implementacji algorytmów rojowych daje wyniki mogące konkurować z algorytmami ewolucyjnymi.

8.1. Zalety wykorzystania algorytmu roju cząstek

Jak można zaobserwować na podstawie danych przedstawionych w rozdziale 7, algorytm PSO radzi sobie niewiele gorzej niż algorytm EMAS. Zdecydowaną zaletą badanego rozwiązania jest ilość iteracji w jakiej algorytm rojowy osiąga stosunkowo zadowalającą wartość funkcji dopasowania. Algorytm EMAS na osiągnięcie podobnej jakości potrzebuje ponad dwukrotnie większej ilości iteracji.

Przy wykorzystaniu jednej wyspy obliczeniowej już w momencie pięćsetnej iteracji wartość dopasowania dla rozwiązania jest zauważalnie lepsza niż dla algorytmu EMAS, co pozwala na szybkie prototypowanie przy rozwiązywaniu bardziej skomplikowanych problemów. Z drugiej jednak strony, wraz z postępem obliczeń wzrost jakości nie jest już tak szybki.

Wykorzystywanie rozwiązań dostarczonych przez platformę pyAgE, jak również połączenie algorytmów rojowego z ewolucyjnym dało pozytywny skutek. Rozszerzenie wiedzy agenta będącego cząsteczką roju o wiedzę na temat rozwiązania wypracowanego przez jego sąsiadów i dodanie najlepszego z nich jako wektora składowego pozwoliło na skuteczne konkurowanie z rozwiązaniami wypracowywanymi przez algorytm EMAS. Wraz z postępem iteracji jakość rozwiązania opracowywana przez rój jest stale

poprawiana i przy wykorzystaniu jednej wyspy obliczeniowej jest w stanie konkurować z algorytmem ewolucyjnym.

Wykonywanie przez każdego agenta na początku każdego kolejnego kroku elementów algorytmu ewolucyjnego, a następnie rojowego zgodnie z oczekiwaniami spowodowało wzrost czasu wykonywania, jednak pozwoliło na opracowywanie najlepszych rozwiązań. Połączenie algorytmów pozwala na początkowo szybsze przybliżanie się do globalnego ekstremum niż algorytm EMAS, a wraz z postępem iteracji pozwala na szybsze dalsze udoskonalania iteracji niż algorytm PSO.

8.2. Możliwy rozwój

Ponieważ algorytm roju cząstek jest algorytmem umożliwiającym łatwe modyfikacje, możliwe jest wprowadzenie dodatkowych parametrów warunkujących ruch cząstki. Jedną z przykładowych możliwych modyfikacji może być wykorzystanie sąsiedztwa innych cząstek w przestrzeni rozwiązań jako kolejnej składowej.

Podczas eksperymentów zauważalny był spadek jakości rozwiązań algorymtu rojowego względem ewolucyjnego w przypadku zwiększenia ilości wysp obliczeniowych. Jest to pole do dalszych ulepszeń, które mogą skutkować lepszym wykorzystaniem dostarczanych przez platformę mechanizmów. Umożliwienie migracji cząstek pomiędzy wyspami spowodowałoby zmniejszenie ryzyka utknięcia roju w lokalnym minimum.

Inteligencja stadna jest szeroko rozwijaną koncepcją, o czym świadczyć może długa i ciągle rozszerzająca się lista algorytmów inspirowanych naturą [28]. Dzięki prostemu interfejsowi dostarczanemu przez platformę, jak i prostym mechanizmom konfigurowania i testowania rozwiązań, możliwe jest szybkie sprawdzanie skuteczności nowych algorytmów stadnych.

Bibliografia

[1] C. W. Reynolds *Flocks, Herds, and Schools: A Distributed Behavioral Model*. Computer Graphics, 21(4), Lipiec 1987, str. 25-34

[2] G. Beni, J. Wang *Swarm Intelligence in Cellular Robotic Systems*. NATO Advanced Workshop on Robots and Biological Systems, Włochy, Lipiec 1989

[3] J. Kennedy, R. Eberhart *Particle Swarm Optimization*. Proceedings of IEEE International Conference on Neural Networks IV, 1995, str. 1942–1948

[4] D. Karaboga *An Idea Based On Honey Bee Swarm for Numerical Optimization*. Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, Turcja, 2005

[5] D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri , S. Rahim , M. Zaidi *The Bees Algorithm – A Novel Tool for Complex Optimisation Problems*. Technical Note, Manufacturing Engineering Centre, Cardiff University, Wielka Brytania, 2005

[6] X. Hu, J. Zhang, Y. Li *Orthogonal methods based ant colony search for solving continuous optimization problems*. Journal of Computer Science and Technology, Springer, Styczeń 2008, str. 2-28

[7] M. Dorigo, L.M. Gambardella *Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem*. IEEE Transactions on Evolutionary Computation, 1997, str. 53-66

[8] S. Mirjalili *The Ant Lion Optimizer*. Advances in Engineering Software 83, 2015, str. 80–98.

[9] P.N. Suganthan *Particle swarm optimiser with neighbourhood operator*. Evolutionary Computation, IEEE, Washington, 1999

[10] M. Clerc *Standard Particle Swarm Optimisation*. HAL open access archive, 2012

[11] Y. Shi, R.C. Eberhart *Parameter selection in particle swarm optimization*. Evolutionary Computation, IEEE, Washington, 1999 Proceedings of Evolutionary Programming VII (EP98), 1998

[12] R.C. Eberhart, Y. Shi *Comparing inertia weights and constriction factors in particle swarm optimization*. Proceedings of the Congress on Evolutionary Computation 1, 2000

[13] G.E.P. Box *Evolutionary operation: A method for increasing industrial productivity*. Appl. Statistics, vol. VI, no.2, 1957, str. 81–101

[14] J. H. Holland *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975

[15] A. Byrski, R. Dreżewski, M. Kisiel-Dorohinicki, L. Siwik *Evolutionary Multi-Agent Systems*. AGH University of Science and Technology, 2012
`https://age.iisg.agh.edu.pl/emas/emas.html`.

[16] K. Cetnarowicz, M. Kisiel-Dorohinicki, E. Nawarecki *The application of evolution process in multi-agent world (MAW) to the prediction system*. M. Tokoro, editor, Proc. of the 2nd Int. Conf. on Multi-Agent Systems (ICMAS'96). AAAI Press, 1996

[17] A. Byrski, R. Drezewski, L. Siwik, M. Kisiel-Dorohinicki *Evolutionary multiagent systems*. The Knowledge Engineering Review, 2013

[18] M. Kisiel-Dorohinicki *Agent-oriented model of simulated evolution*. In William I. Grosky and Frantisek Plasil, editors, SofSem 2002: Theory and Practice of Informatics, volume 2540 of LNCS. Springer-Verlag, 2002.

[19] Wikimedia Commons *Rastrigin function*.
`http://commons.wikimedia.org/wiki/File:Rastrigin function.png`.

[20] M. Kisiel-Dorohinicki *Agentowe architektury populacyjnych systemów inteligencji obliczeniowej*. Rozprawy monograficzne 269, Wydawnictwo AGH, Kraków, 2013

[21] S. Franklin, A. Graesser *Is it an agent or just a program?: A taxonomy for antonomous agents.*. Intelligent Agents III vol. 1193, LNAI. Springer-Verlag, 1997

[22] M. Wooldridge *Agent-based Software Engineering.*. IEEE Trans. on Software Engineering, vol. 144, no. 1, 1997

[23] M. Kaziród, W. Korczynski, A. Byrski. *Agent-oriented computing platform in python.*. Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014 IEEE/WIC/ACM International Joint Conferences on, vol. 3, pages 365-372. IEEE, 2014

[24] L. A. Rastrigin *Systems of extremal control.*. Nauka, Moskwa, 1974

[25] H. Mühlenbein, D. Schomisch, J. Born *The Parallel Genetic Algorithm as Function Optimizer.*. Parallel Computing, 17, 1991

[26] M. Kisiel-Dorohinicki *Evolutionary multi-agent systems in non-stationary environments.* Computer Science 14, 2013

[27] R. W. Morrison, K. A. D. Jong *Measurement of Population Diversity. In: Artificial Evolution.* 5th Int. Conf., Evolution Artificielle, EA 2001, P. Collet, et al., eds., LNCS, vol. 2310, Springer, 2002

[28] I. Fister, X.S. Yang, J. Brest, D. Fister *A Brief Review of Nature-Inspired Algorithms for Optimization.* Elektrotehniski Vestnik, 80 , 2013

[29] M. Kaziród *pyAgE github*. `https://github.com/maciek123/pyage`.

[30] U. Boryczka *Inteligencja stadna*. `http://155.158.112.34/algorytmyewolucyjne/materialy/inteligencja stadna.pdf`.