



**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 ... tu ciąg dalszych podziękowań które zostaną uzupełnione pod koniec :)*

## Spis treści

<b>1. Wprowadzenie .....</b>	<b>6</b>
1.1. Cele pracy .....	6
1.2. Zakres pracy .....	6
1.3. Zawartość pracy .....	7
<b>2. Rozdział o pygu i reszcie.....</b>	<b>8</b>
2.1. Platforma PyAGE .....	8
2.2. Sposób porównywania wyników .....	8
2.2.1. Funkcja Rastrigina .....	8
<b>3. Algorytmy rojowe .....</b>	<b>9</b>
3.1. Przegląd wiedzy .....	9
3.2. Algorytm roju cząstek .....	10
3.2.1. Opis algorytmu.....	10
3.2.2. Przemieszczenie .....	11
3.2.3. Zasada działania algorytmu .....	12
3.3. Modyfikacje algorytmu roju cząstek .....	13
3.3.1. Rozszerzenie wiedzy cząstek .....	13
3.3.2. Dodanie losowej składowej.....	13
3.3.3. Nadawanie wag parametrom.....	13
3.3.4. Zmiana prędkości ruchu .....	13
<b>4. Algorytmy ewolucyjne .....</b>	<b>14</b>
4.1. Przegląd wiedzy .....	14
4.2. Algorytm EMAS .....	15
4.2.1. Agent .....	15
4.2.2. Interakcje agentów .....	15
4.2.3. Ewolucja.....	15
4.2.4. Wyspy obliczeniowe .....	15
<b>5. Algorytm roju cząstek na platformie PyAGE .....</b>	<b>16</b>

6. Porównanie algorytmów genetycznego i PSO .....	17
---------------------------------------------------	----

## Spis rysunków

3.1 Wizualizacja ruchu cząstki PSO . . . . .	11
3.2 Diagram blokowy algorytmu PSO . . . . .	12
4.1 Diagram blokowy klasycznego algorytmu genetycznego . . . . .	15

# 1. Wprowadzenie

Tematem niniejszej pracy jest sprawdzenie skuteczności algorytmu roju cząstek w wybranych problemach optymalizacji ciągłej. Skuteczność rozwiązania została oceniona względem dostarczonej platformy wraz z istniejącym rozwiązaniem.

## 1.1. Cele pracy

Celem pracy jest realizacja różnych wariantów implementacji algorytmu roju cząstek oraz badanie ich skuteczności w wybranych problemach optymalizacji ciągłej. W implementacji wykorzystane zostanie istniejące środowisko wspierające budowę rozproszonych modeli obliczeniowych w języku Python (pyAgE). Skuteczność algorytmu roju cząstek badana będzie eksperymentalnie dla różnych wariantów konfiguracji obliczeń oraz różnych problemów benchmarkowych.

## 1.2. Zakres pracy

W czasie realizacji pracy, została rozszerzona platforma pyAgE o możliwość dokonywania obliczeń za pomocą algorytmu roju cząstek. Zostały zaimplementowane zarówno elementy składowe algorytmu jak i przetestowane konfiguracje pozwalające na badanie skuteczności algorytmu.

Porównanie odbywało się pomiędzy dostarczonym wraz z platformą pyAgE algorytmem genetycznym 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 2 zawiera informacje o platformie pyAgE i jej komponentach. Znajdują się tam również informacje o funkcjach benchmarkowych, ze szczególnym uwzględnieniem funkcji stosowanej do porównania w niniejszej pracy.

Następną częścią pracy (rozdziały 3 i 4) jest przybliżenie mechanizmów działania zarówno algorytmów rojowych jak i ewolucyjnych, szczególnie tych, które zostały zastosowane w implementacji. W rozdziale 3.3 zostały też częste modyfikacje algorytmu roju cząstek.

Rozdział 5 zawiera informacje o zaimplementowanych rozwiązaniach oraz wyniki badań prowadzące do uzyskania najlepszych parametrów algorytmu roju cząstek.

Pod koniec pracy znajdują się porównania algorytmów rojowego i ewolucyjnego oraz analiza skuteczności algorytmu rojowego pod kątem postawionego problemu.

## **2. Rozdział o pygu i reszcie**

### **2.1. Platforma PyAGE**

### **2.2. Sposób porównywania wyników**

#### **2.2.1. Funkcja Rastrigina**



### 3. Algorytmy rojowe

Inteligencją stadną nazywane są techniki sztucznej inteligencji bazujące na wiedzy o społecznych zachowaniach w samo zorganizowanym systemie.

Systemy rojowe są najczęściej zbudowane z populacji prostych agentów oddziałujących na siebie nawzajem oraz na środowisko w którym się znajdują. Nie posiadają scentralizowanej struktury kierującej zachowaniem indywidualnych agentów. Agenci posiadają jedynie wiedzę o akcji jaką powinni podjąć przy danych bodźcach zewnętrznych. Wzajemne oddziaływanie pomiędzy agentami prowadzi do złożonych zachowań populacji.

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

#### 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 pszczoły (ang. Bees algorithm (BA))[5] i pszczelej kolonii (ang. Artificial bee colony algorithm (ABC))[4]. Część z algorytmów mimo, że ich pierwotne wersje były dostosowane do rozwiązywania problemów dyskretnych, z czasem została zmodyfikowana 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 oddziałując, 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 agent w populacji posiada zestaw mechanizmów warunkujących jego ruch po przestrzeni rozwiązań. Ponadto ma pamięć o najlepszym miejscu w przestrzeni jakie odwiedził, oraz wiedzę o położeniu agenta 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,

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

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

$$p_{bi} = (p_{bi1}, p_{bi2}, p_{bi3}, \dots, p_{biD}) \quad (3.2)$$

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

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

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

$$x_{id}^{n+1} = x_{id}^n + v_{id}^{n+1} \quad (3.4)$$

gdzie:

$d = 1, 2, 3, \dots, D$

$i = 1, 2, 3, \dots, N$

$N$  - rozmiar populacji

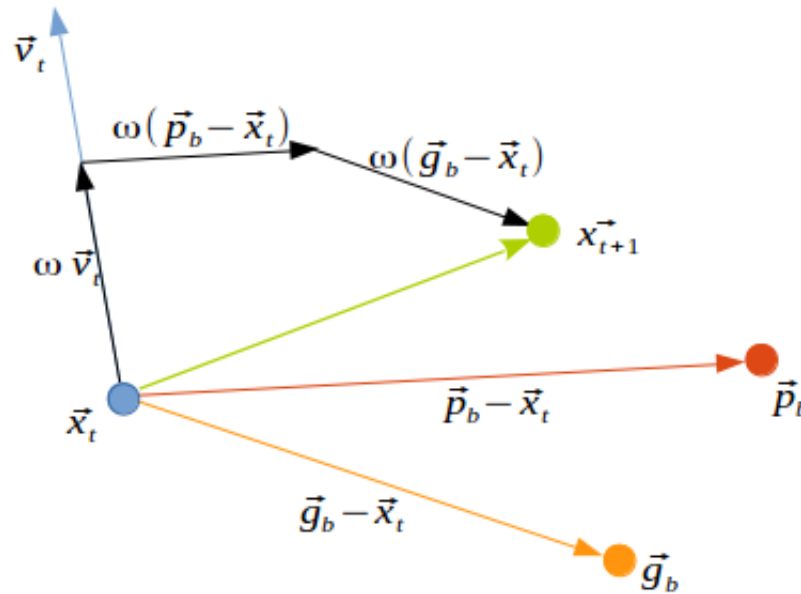
$n = 1, 2, 3, \dots$  - numer iteracji

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  $\omega$  (3.5).

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

### 3.2.2. Przemieszczenie

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

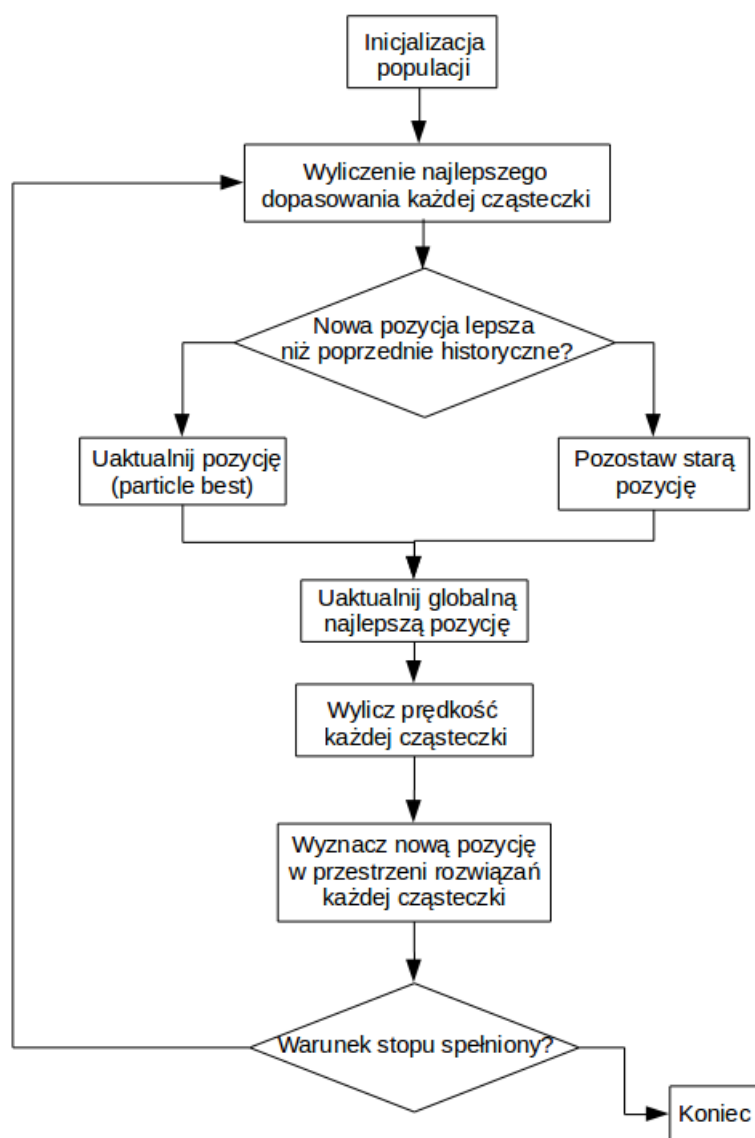


Rysunek 3.1: Wizualizacja ruchu cząstki PSO

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  $p_b$ . Agent wyznacza wektory przesunięcia względem tych punktów, oraz przemieszczenia z poprzedniej iteracji  $v_t$ . Każdy z wektorów jest mnożony przez współczynnik inercji  $\omega$ , 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 zainicjowanie startowej populacji w losowych miejscach przestrzeni rozwiązań. Następnie dla każdego agenta liczone jest dopasowanie. 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 agentów 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 agenta[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 generowana jest losowo, daje możliwość pozbycia się pewnych potencjalnych problemów. 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 agenta od roju, co może skutkować wyskoczeniem z lokalnego ekstremum i dalsze przeszukiwanie 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.

#### 3.3.4. Zmiana prędkości ruchu

Algorytm roju cząstek w swojej pierwotnej wersji nie zakładał zmiany prędkości agentów w czasie. Rozszerzenie dające możliwość manipulowania współczynnikiem inercji  $\omega$  na przestrzeni kolejnych iteracji pozwala na uniknięcie potencjalnego "przeskakiwania" poprawnego rozwiązania przez agenty. 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  $\omega$  wraz z kolejnymi iteracjami pozwoli cząstkom dokładniej przeszukać dany zakres, jednocześnie utrzymując szeroki zasięg przeszukiwań na początku działania algorytmu.

## 4. Algorytmy ewolucyjne

Algorytmami ewolucyjnymi nazywane są algorytmy, które w celu przeszukania przestrzeni rozwiązań wykorzystują mechanizmy zaczerpnięte ze zjawiska ewolucji biologicznej. Jest to ogólna nazwa dla metod takich jak algorytmy genetyczne, strategie ewolucyjne czy neuroewolucje.

Podobnie jak opisane w rozdziale 3 algorytmy rojowe, ewolucyjne również zawierają populację agentów wpływających nawzajem na siebie. Populacja agentów generowana jest losowo, wraz z pewnym zestawem cech dla każdego agenta - genotypem. Genotyp jest takim zestawem cech agenta, który umiejscawia go w pewnej przestrzeni rozwiązań, co umożliwia jego ewaluację. Podczas działania algorytmu, agenci poprzez krzyżowanie się, umieranie i rodzenie wpływają na swoje genotypy.

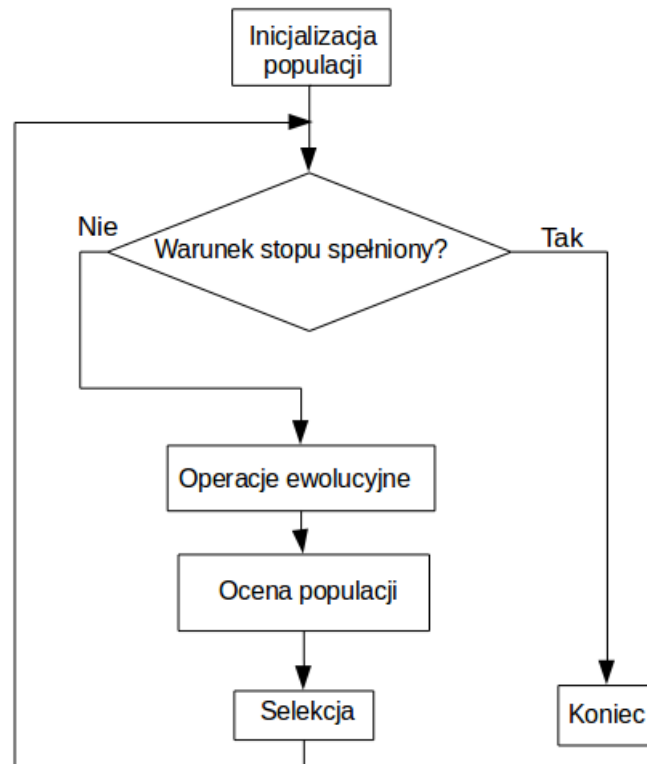
Na przestrzeni lat zostało zaproponowanych wiele algorytmów bazujących na mechanizmach genetycznych, jednak wszystkie z nich opierały się na tych samych bazowych mechanizmach. Każdy z osobników populacji mógł, zmieniając swój genotyp, przybliżyć całą populację do znalezienia optymalnego rozwiązania postawionego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

### 4.1. Przegląd wiedzy

Początki algorytmów ewolucyjnych sięgają lat 50 XX wieku[10], jednak ich idee nie były rozwijane przez wiele lat, głównie ze względu na ograniczenia sprzętowe jak i metodologiczne. Dopiero dwadzieścia lat później[11] pojawiły się prace rozwijające modele ewolucyjne. Wtedy też zostało zaproponowane twierdzenie Hollanda o schematach, które uważane jest za podstawę wyjaśnienia algorytmów genetycznych.

Znaczącą kwestią wpływającą na tempo rozwoju algorytmów genetycznych, 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 4.1, lub podobnie do niego.



Rysunek 4.1: Diagram blokowy klasycznego algorytmu genetycznego

Początkowo inicjalizowana jest losowa populacja, która wykonuje między sobą zachowania ewolucyjne aż do spełnienia warunku stopu. 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 osobników wyliczana jest funkcja przystosowania, pozwalająca ocenić jakość jego genotypu.

## 4.2. Algorytm EMAS

<https://github.com/ParaPhraseAGH/erlang-emas/wiki/EMAS-algorithm>  
<https://age.iisg.agh.edu.pl/emas/intro.html>

### 4.2.1. Agent

### 4.2.2. Interakcje agentów

### 4.2.3. Ewolucja

### 4.2.4. Wyspy obliczeniowe

## **5. Algorytm roju cząstek na platformie PyAGE**

rozdział o różnych parametrach pso, porównanie wszystkich implementacji, eksperymenty z parametrami itp.



## **6. Porównanie algorytmów genetycznego i PSO**

tutaj będą te wszystkie wykresy które do tej pory konsultowaliśmy, i wszystkie ich opisy itp. tutaj już będą tylko takie pso(pso, pso+sasiedztwo, pso+wyspy) które w poprzednim rozdziale było jako najlepsze



## 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, and 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] G.E.P. Box *Evolutionary operation: A method for increasing industrial productivity*. Appl. Statistics, vol. VI, no.2, 1957, str. 81–101
- [11] J. H. Holland *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975

- [12] 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>.