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

Spis rysunków

1. Wprowadzenie

Rozdzial o tym, o czym jest praca, cel pracy, troche o platformie pyage (czy jest, co oferuje), i jak beda porownywane wyniki miedzy soba

1.1. Cele pracy

1.2. Platforma PyAGE

1.3. Sposób porównywania wyników

1.3.1. Funkcja Rastrigina

2. 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, klucze 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.

2.1. Historia algorytmów rojowych

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 sumylacja społeczeństwa, jednak problem okazał się zbyt złożony dla takiego algorytmu. Odnalazł on jednak zastosowanie w problemach optymalizacji ciągłej.

2.2. 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 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.

2.2.1. Równania

Jeśli przestrzeń poszukiwań jest D-wymiarowa, można zaprezentować i-tą cząsteczkę populacji jako D-wymiarowy wektor 2.1,

$$x_i = (x_i1, x_i2, x_i3, \ldots, x_iD)^T \tag{2.1}$$

gdzie T oznacza numer iteracji. Najlepszą odwiedzoną pozycję (particle best) i-tej cząsteczczki można oznaczyć jako 2.2.

$$p_{b_i} = (p_{b_{i1}}, p_{b_{i2}}, p_{b_{i3}}, \ldots, p_{b_{iD}}) \tag{2.2}$$

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 2.3.

$$v_{id}^{n+1} = v_{id}^n + (p_{b_{id}}^n - x_{id}^n) + (g_{b_{id}}^n - x_{id}^n) \tag{2.3}$$

co skutkuje, że w kolejnych iteracjach jego pozycja jest uaktualniana zgodnie z równaniem 2.4

$$x_{id}^{n+1} = x_{id}^n + v_{id}^{n+1} \tag{2.4}$$

gdzie:

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

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

N - rozmiar populacji

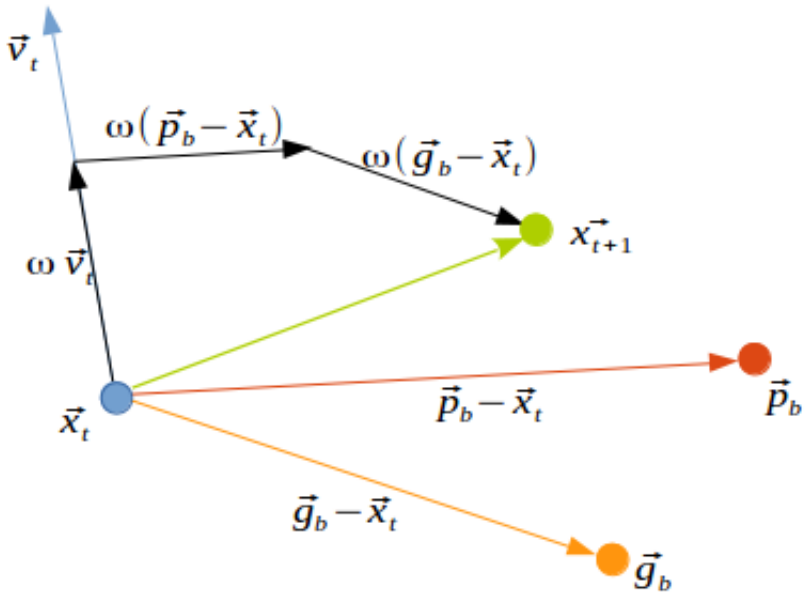
n = 1,2,3, . . . - numer iteracji

Zgodnie z równaniem 2.3, 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 (2.5).

$$v_{id}^{n+1} = \omega(v_{id}^n + (p_{b_{id}}^n - x_{id}^n) + (g_{b_{id}}^n - x_{id}^n) \tag{2.5}$$

2.2.2. Przemieszczenie

Wizualizację zgodnie z równaniami zamieszczonymi w rozdziale 2.2.1, można prześledzić na rysunku 2.1.

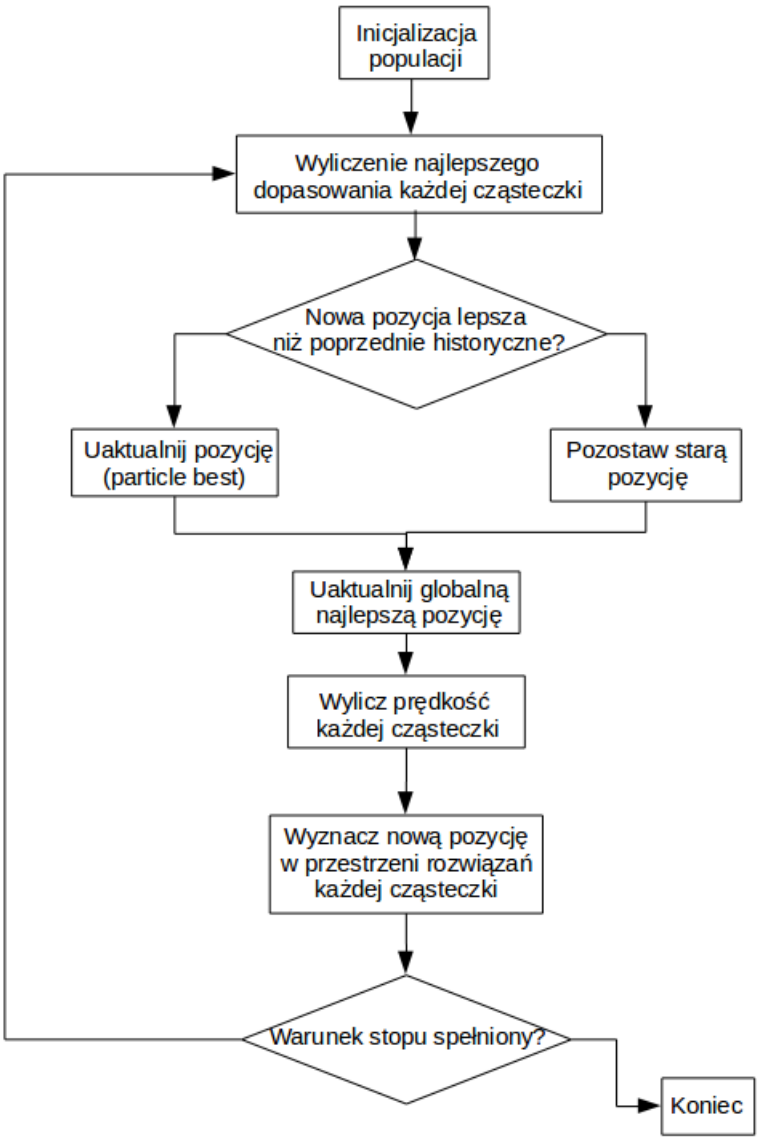


Rysunek 2.1: Wizualizacja ruchu cząstki PSO

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 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.

2.2.3. Zasada działania algorytmu

Diagram 2.2 przedstawia pełny mechanizm działania algorytmu. Pierwszym krokiem jest zainicjalizowanie 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 sewoje dopasowanie i powtarzają cały cykl.



Rysunek 2.2: Diagram blokowy algorytmu PSO

2.3. Modyfikacje algorytmu roju cząstek

W paragrafie 2.2 została przedstawiona podstawowa, najprostsza wersja algorytmu roju cząstek. Istnieje szereg rozszerzeń i modyfikacji algorytmu, które powodują zwiększenie jego dokładności i skuteczności.

2.3.1. 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. Dołożenie takiego parametru, pociąga za sobą zmianę wyliczania ruchu cząstki (rys. 2.1) o dodatkowy wektor. Podejście takie zwiększa skupienie cząsteczek i pozwala na dokładniejsze przeszukiwanie przestrzeni rozwiązań w danym zakresie.

2.3.2. 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 2.2.2, 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 przeszuiwanie przestrzeni rozwiązań w celu znalezienia ekstremum globalnego.

2.3.3. 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.

2.3.4. Zmiana prędkości ruchu

Algorytm roju cząstek w swojej pierwotnej wersji nie zakładał zmiany prędkości agentów w czasie. Rozszerzenie dające możliwość manipulowania współczynnikiem inercji ω na przestrzeni kolejnych iteracji pozwala na uniknięcie potencjalnego "przeskakiwania"poprawnego rozwiązania przez agenty. Wersja podstawowa niesie za sobą ryzyka, 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.

3. Algorytmy genetyczne

rozdzial analogiczny do tego o pso tylko troche krotszy (nie bedzie nic o modyfikacjach itp)

3.1. Historia algorytmów genetycznych

3.2. Algorytm EMAS

4. Algorytm roju cząstek na platformie PyAGE

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

5. Porównanie algorytmów genetycznego i PSO

tutaj będą te wszystkie wykresy które do tej pory konsultowaliśmy, i wszystkie ich opisy itp. tutaj juz beda tylko takie pso(pso, pso+sasiedztwo, pso+wyspy) ktore w poprzednim rozdziale bylo 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