O dwóch modyfikacjach algorytmu PSO

Michał Okulewicz

Wydział Matematyki i Nauk Informacyjnych
Politechnika Warszawska

Seminarium: Inteligencja Obliczeniowa
24 listopada 2011

Plan prezentacji

Definicja problemu

Prezentowane algorytmy rozwiązują problem poszukiwania minimum globalnego x^* funkcji ciągłej f

- $f : \mathbb{R}^n \to \mathbb{R}$
- x^*, tż. $\forall x \in \mathbb{R}^n f(x) \geq f(x^*)$.

Motywacja

- Poszukiwanie algorytmu łączącego dobrze zdolności eksploracyjne z eksploatacyjnymi
- Czy zamodelowanie bardziej skomplikowanej sytuacji społecznej poprawi wyniki PSO?
- Czy modyfikacja strategii działania algorytmu poprawi jego wyniki bardziej niż dostrajanie parametrów?

Znane metody

- Symulowane Wyżarzanie
- Algorytmy Ewolucyjne
- Wielokrotna/równoległa optymalizacja lokalna
- **Optymalizacja Rojowa**
- Algorytm Cywilizacyjny
- ...

Inteligencja rojowa

Postulaty

- Decentralizacja sterowania i samoorganizacja prostych bytów
  - Nieskomplikowany cel każdego z bytów
  - Komunikacja i współpraca pomiędzy bytami

Przykładowa realizacja

- Ant Colony Optimization - wyznaczanie optymalnej trasy
- Particle Swarm Optimization - optymalizacja globalna
- Artificial Bee Colony - optymalizacja globalna
- Symulacja zachowań tłumu - przemysł filmowy

Porównywane algorytmy

- Optymalizacja Rojem Cząstek
  - Particle Swarm Optimization (PSO)
  - Repulsive Particle Swarm Optimization (RPSO)
  - Mixed Particle Swarm Optimization (MPSO)
- Algorytm Wynalazców - Inventors Algorithm (InA)
- Algorytm Odkrywców - Explorers Algorithm (ExA)

Opis

- Algorytm opiera swoje działanie na roju cząstek
- Każda cząsteczka ma swoją prędkość
- Cząsteczki posiadają pewną bezwładność
- Cząsteczki mogą być przyciągane przez najlepszy punkt znaleziony przez swoich sąsiadów
- Cząsteczki są przyciągane przez najlepszy punkt znaleziony przez siebie
- Cząsteczki mogą być odpychane od innych cząsteczek

PSO - schemat działania algorytmu

```
PSO() {
  swarm.initializeRandomlyParticlesLocationAndVelocity();
  for i from 1 to maxIterations {
    swarm.updateBestLocation();
    for each particle in swarm {
      particle.updateVelocity();
      particle.updateLocation();
    }
  }
}
```

PSO - zmiana prędkości cząsteczki

```
Particle {
  updateVelocity() {
    for (i from 1 to dimensions) {
      this.v[i] =
        random.uniform(0,g)*(swarm.best[i] - this.x[i]) +
        random.uniform(0,l)*(this.best[i] - this.x[i]) +
        random.uniform(0,r)*(this.x[i] - swarm.random().x[i]) +
        a * this.v[i] +
        y * random.normal(0,1);
    }
  }
```

PSO - zmiana położenia cząsteczki

```
  updateLocation() {
    for (i from 1 to dimensions) {
      this.x[i] = this.x[i] + this.v[i];
    }
    if (f(this.best) > f(this.x)) {
      this.best = this.x;
    }
  }
}
```

Opis

- Pewną część roju stanowią cząstki wynalazcy
- Odkrywcy poszukują nowych nie eksploatowanych aktualnie rozwiązań (zachowują się jak cząsteczki w algorytmie RPSO)

Inventors Algorithm - zmiana prędkości cząsteczki

```
InventorsParticle: Particle {
  updateVelocity() {
    for (i from 1 to dimensions) {
      this.v[i] =
        random.uniform(0,r)*(swarm.best[i] - this.x[i]) +
        random.uniform(0,l)*(this.best[i] - this.x[i]) +
        random.uniform(0,g)*(this.x[i] - swarm.random.x[i]) +
        a * this.v[i] +
        y * random.normal(0,1);
    }
  }
}
```

Opis

- Pewną część roju stanowią cząsteczki odkrywcy
- Cząsteczki odkrywcy początkowo zachowują się tak samo jak cząsteczki wynalazcy
- Cząsteczka odkrywca może ulec "znudzeniu" i zacząć poszukiwać maksimum funkcji
- Cząsteczka odkrywca poszukująca maksimum może ulec "zmęczeniu" i zacząć poszukiwać minimum funkcji

Explorers Algorithm - zmiana prędkości cząsteczki

```
ExplorersParticle: InventorsParticle {
  updateLocation() {
    for (i from 1 to dimensions) {
      this.x[i] = this.x[i] + this.v[i];
    }
    if (f(this.best[t]) > f(this.x) && this.isSerchingForMinimum) {
      this.best[t] = this.x;
    }
    if (f(this.best[t]) < f(this.x) && !this.isSerchingForMinimum) {
      this.best[t] = this.x;
    }
    if (!this.isSerchingForMinimum) {
      if (fatigue++ > 2 * this.samples_to_switch) {
        fatigue = 0;
        this.isSerchingForMinimum = true;
        this.samples_to_switch *= 2;
      }
    }
    if (|this.best[t-samples_to_switch] - this.best[t]| < precision_to_switch) {
      fatigue = 0;
      this.isSerchingForMinimum = !this.isSerchingForMinimum;
    }
  }
}
```

Testowane scenariusze

Porównanie jakości działania algorytmu bazowego i jego modyfikacji dla następujących przypadków

- Standardowa początkowa przestrzeń przeszukiwań
- Początkowa przestrzeń przeszukiwań nie zawierająca minimum globalnego
- Początkowa przestrzeń przeszukiwań zawierająca się w bliskim sąsiedztwie minimum lokalnego

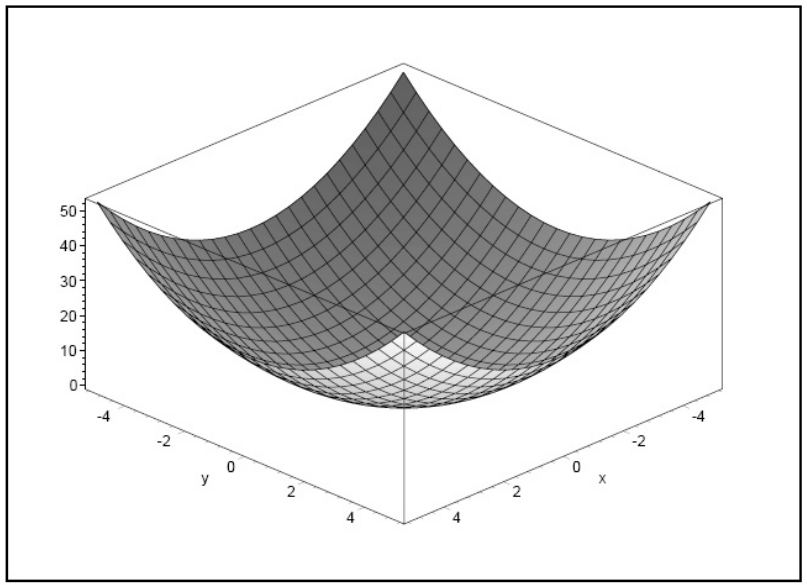
Warunki eksperymentu

- Średnia ze 100 przebiegów każdego z algorytmów
- Każdy przebieg miał 100 cząsteczek i 100 iteracji
- Przebieg był uznawane za sukces jeżeli $\|x_A^* - x^*\| < 0.05$ lub $\sqrt{f(x_A^*) - f(x^*)} < 0.05$ a miarą sukcesu jest liczba obliczeń optymalizowanej funkcji potrzebna do osiągnięcia sukcesu

Funkcje testowe

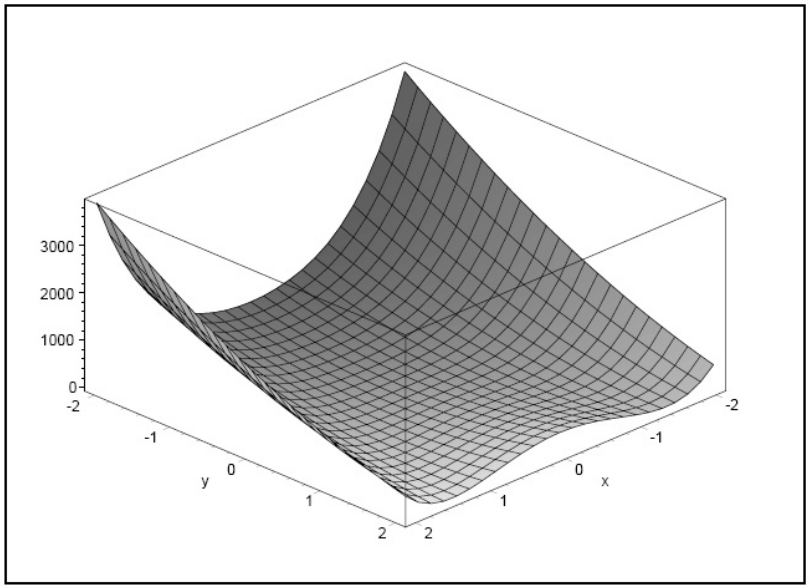
- Pierwsza funkcja de Jonga
- Dolina Rosenbrocka
- Piąta funkcja de Jonga
- Funkcja Rastrigina

Pierwsza funkcja de Jonga



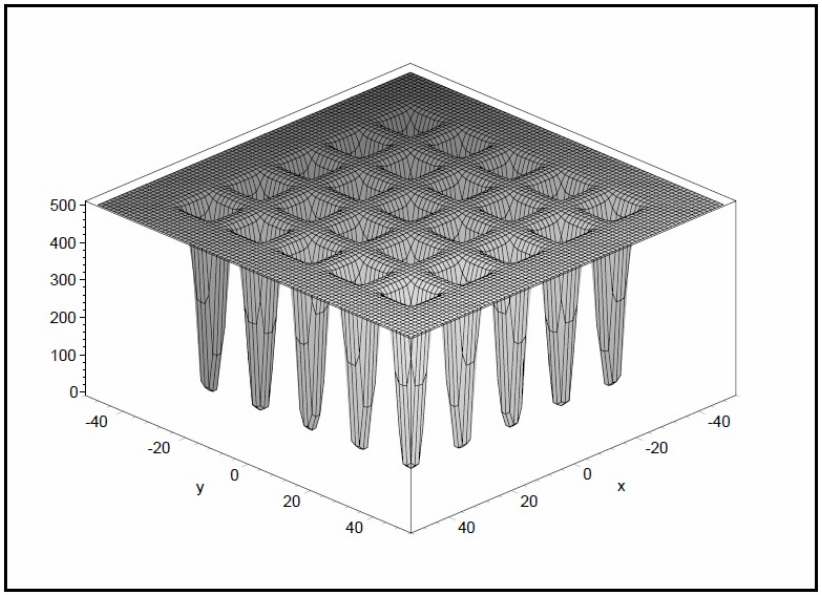
- $f(x) = \sum_{i=1}^{n} x_i^2$.
- $x^* = (0, 0, \ldots, 0)$.
- $f(x^*) = 0$.

Dolina Rosenbrocka



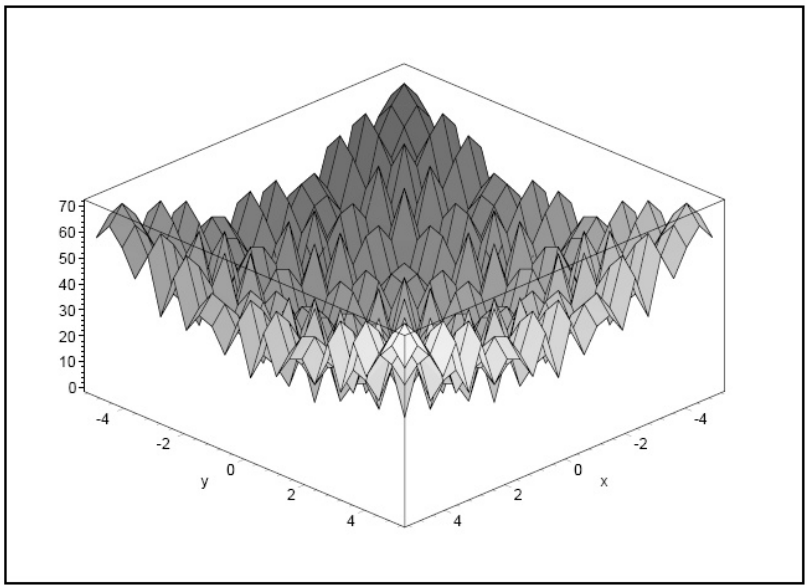
- $f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$
- $x^* = (1, 1, \ldots, 1)$.
- $f(x^*) = 0$.

Piąta funkcja de Jonga



- $f(x) = \left(\sum_{i=-2}^{2} \sum_{j=-2}^{2} \left(5(i+2) + j + 3 + (x_1 - 16j)^6 + (x_2 - 16i)^6\right)^{-1} \right)^{-1}$.
- $x^* = (-32, -32)$.
- $f(x^*) = 0.998$.

Funkcja Rastrigina



- $f(x) = 10n + \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i)]$
- $x^* = (0, 0, \ldots, 0)$.
- $f(x^*) = 0$.

Wyniki - procent sukcesów I

Funkcja	Algorytm	Procent sukcesów		
		Test 1	Test 2	Test 3
Pierwsza funkcja de Jonga	PSO	100%	100%	-
	RPSO	0%	0%	-
	MPSO	0%	1%	-
	InA	100%	100%	-
	ExA	100%	100%	-
Dolina Rosenbrocka	PSO	1%	0%	-
	RPSO	0%	0%	-
	MPSO	0%	0%	-
	InA	2%	1%	-
	ExA	1%	0%	-

Wyniki - procent sukcesów II

Funkcja	Algorytm	Procent sukcesów		
		Test 1	Test 2	Test 3
Piąta funkcja de Jonga	PSO	96%	12%	0%
	RPSO	2%	0%	1%
	MPSO	28%	15%	7%
	InA	71%	21%	25%
	ExA	73%	24%	21%
Funkcja Rastrigina	PSO	12%	0%	0%
	RPSO	0%	0%	0%
	MPSO	0%	0%	0%
	InA	4%	1%	0%
	ExA	5%	0%	2%

Wyniki - uzyskana wartość funkcji I

Funkcja	Algorytm	Wartość funkcji		
		Test 1	Test 2	Test 3
Pierwsza funkcja de Jonga	PSO	0.00	0.00	-
	RPSO	0.25	1.00	-
	MPSO	0.01	0.00	-
	InA	0.00	0.00	-
	ExA	0.00	0.00	-
Dolina Rosenbrocka	PSO	0.05	0.20	-
	RPSO	78.53	9.83	-
	MPSO	0.91	0.33	-
	InA	0.05	0.05	-
	ExA	0.03	0.13	-

Wyniki - uzyskana wartość funkcji II

Funkcja	Algorytm	Wartość funkcji		
		Test 1	Test 2	Test 3
Piąta funkcja de Jonga	PSO	1.00	1.00	23.81
	RPSO	1.00	1.00	1.00
	MPSO	1.00	1.00	1.00
	InA	1.00	1.00	1.00
	ExA	1.00	1.00	1.00
Funkcja Rastrigina	PSO	0.01	1.01	4.14
	RPSO	2.19	12.14	5.35
	MPSO	1.22	1.98	1.30
	InA	0.11	0.13	1.08
	ExA	0.03	1.01	0.15

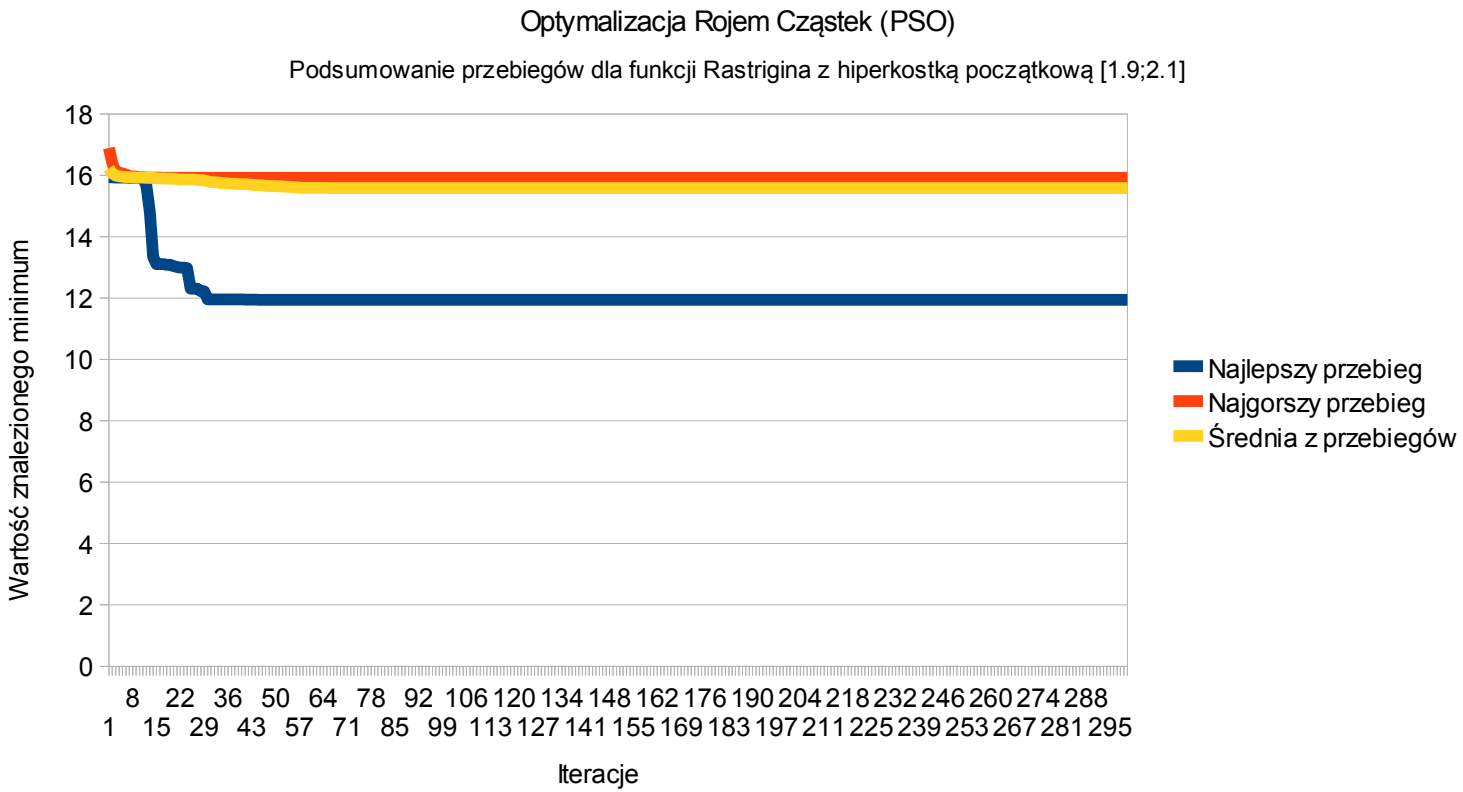
Wyniki - liczba kroków I

Funkcja	Algorytm	Średnia liczba kroków		
		Test 1	Test 2	Test 3
First de Jong	PSO	22	26	-
	RPSO	x	x	-
	MPSO	x	70	-
	InA	27	32	-
	ExA	28	32	-
Rosenbrock's	PSO	26	x	-
	RPSO	x	x	-
	MPSO	x	x	-
	InA	48	95	-
	ExA	97	x	-

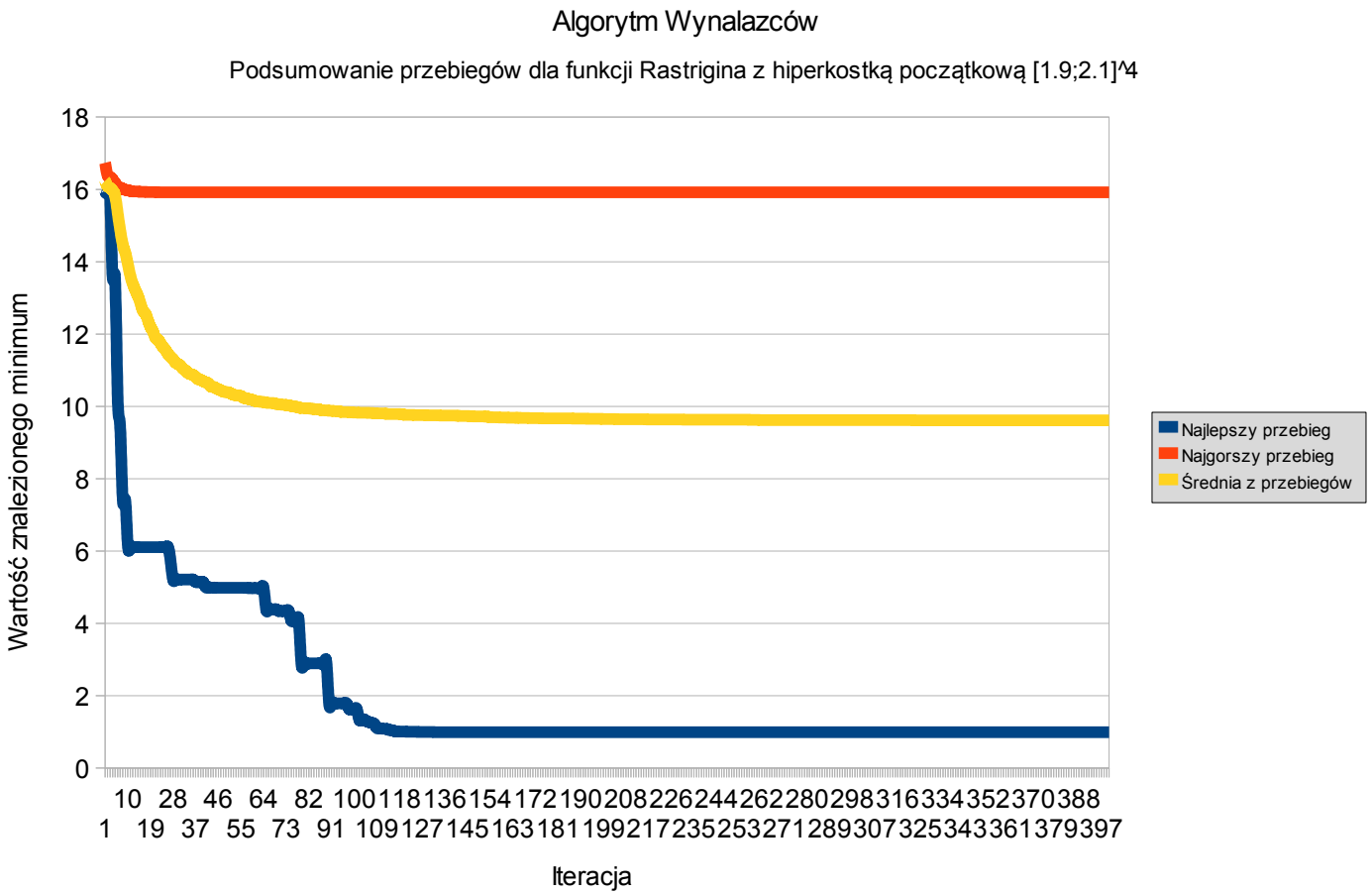
Wyniki - liczba kroków II

Funkcja	Algorytm	Średnia liczba kroków		
		Test 1	Test 2	Test 3
Fifth de Jong	PSO	27	49	x
	RPSO	14	x	73
	MPSO	45	36	77
	InA	32	43	66
	ExA	39	45	39
Rastrigin's	PSO	41	x	x
	RPSO	x	x	x
	MPSO	x	x	x
	InA	30	21	x
	ExA	40	x	62

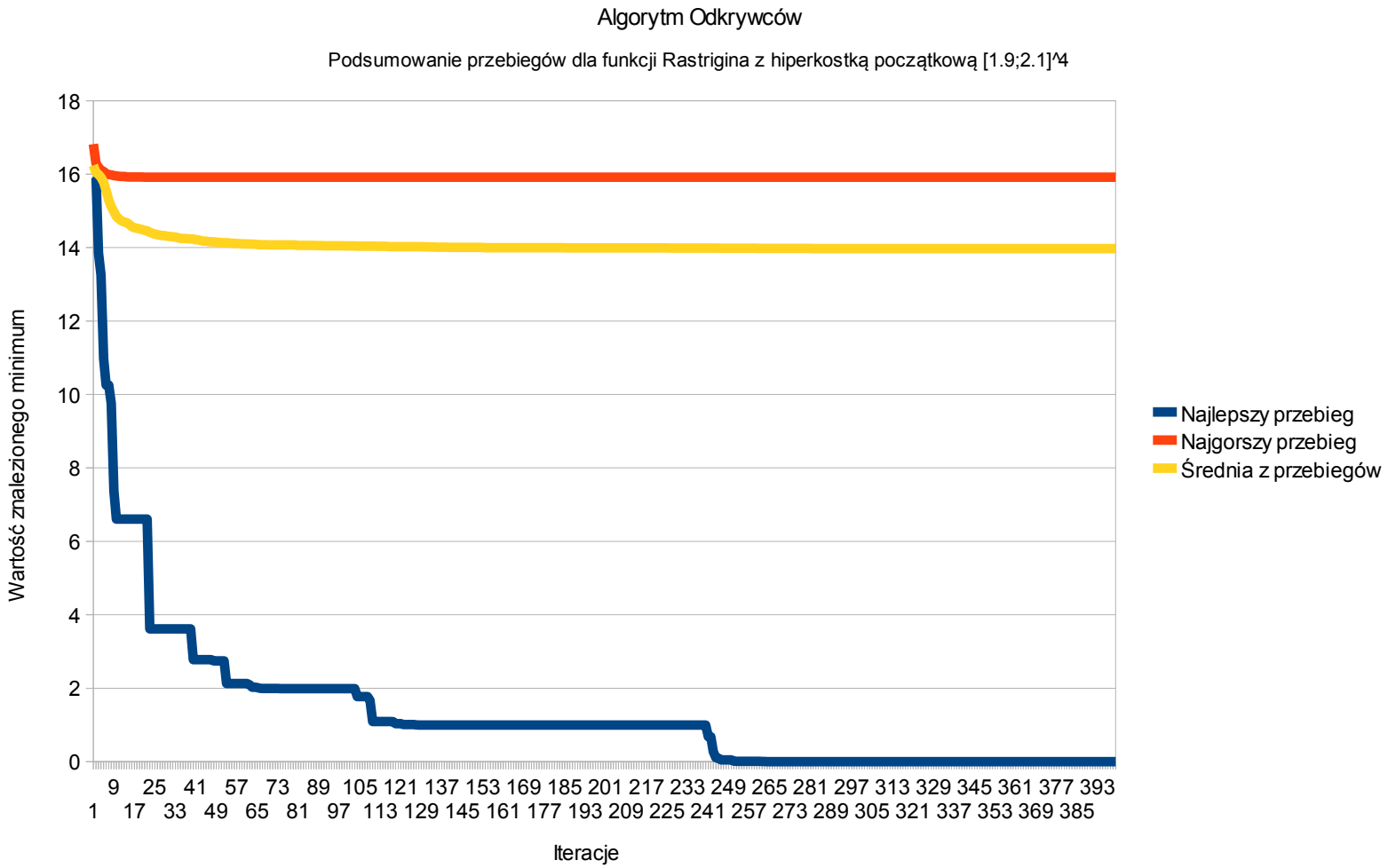
Wyniki - przykładowy przebieg



Wyniki - przykładowy przebieg



Wyniki - przykładowy przebieg



Wnioski

- Dla przypadków o niewielkiej liczbie wymiarów modyfikacje algorytmu sprostały postawionemu zadaniu wydostania się z minimum lokalnego
- Modyfikacja nie obniżyły w sposób istotny jakości działania algorytmu bazowego dla prostszych funkcji

Dalsze działania

- Zaimplementowanie ograniczenia prędkości cząstki
- Wykorzystanie metaoptymalizacji
  - Jako strategii optymalizacyjnej
  - Jako pomocy w dynamicznym ustalaniu parametrów
  - Jako pomocy w ustalaniu parametrów dla zadania danej klasy

Dalsze badania

- Zmierzenie rozproszenia roju w czasie
- Zmierzenie średniej prędkości cząstek roju w czasie
- Przetestowanie algorytmów benchmarkiem COmparing Continuous Optimisers w ramach Black-Box Optimization Benchmarking 2012
- Zaprezentowanie problemu doboru architektury i konfiguracji serwerów mapowych w postaci funkcji i próba jej optymalizacji przy użyciu algorytmów

Źródła I


Standard PSO 2006, 2006.


Ioan Cristian and Trelea.
The particle swarm optimization algorithm: convergence analysis and parameter selection.
Information Processing Letters, 85(6):317 – 325, 2003.


Kenneth Alan De Jong.
An analysis of the behavior of a class of genetic adaptive systems.
PhD thesis, 1975.


Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger.
Real-parameter black-box optimization benchmarking 2010: Noiseless functions definitions, 2011.


D. Karaboga and B. Basturk.
On the performance of artificial bee colony (abc) algorithm.
Applied Soft Computing, 8(1):687 – 697, 2008.


J. Kennedy and R. Eberhart.
Particle swarm optimization.
Proceedings of IEEE International Conference on Neural Networks. IV, pages 1942–1948, 1995.


Marcin Molga and Czesław Smutnicki.
Test functions for optimization needs, kwiecień 2005.
http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf.

Źródła II

H. Mühlenbein, D. Schomisch, and J. Born.

The Parallel Genetic Algorithm as Function Optimizer.
Parallel Computing, 17(6-7):619–632, 1991.

A. Immanuel Selvakumar and K. Thanushkodi.

Optimization using civilized swarm: Solution to economic dispatch with multiple minima.
Electric Power Systems Research, 79(1):8 – 16, 2009.

Y. Shi and R.C. Eberhart.

A modified particle swarm optimizer.
Proceedings of IEEE International Conference on Evolutionary Computation, page 69–73, 1998.

Y. Shi and R.C. Eberhart.

Parameter selection in particle swarm optimization.
Proceedings of Evolutionary Programming VII (EP98), page 591–600, 1998.

A. Törn and A. Zilinskas.

Global Optimization.
Lecture Notes in Computer Science, 350, 1989.