

Paweł Szynkiewicz  
Jakub Tyburski

## Sprawozdanie z projektu PORR (część I)

### *Algorytm genetyczny z kodowaniem binarnym*

Prowadzący: dr Adam Woźniak

## 1. Algorytm genetyczny

Algorytmy genetyczne to proste heurystyki populacyjne. Pomysł technik ewolucyjnych pochodzi z symulacji procesu ewolucji żywych organizmów. Stan algorytmu opisuje zbiór odpowiednio zakodowanych łańcuchów (np. binarnych), będących analogiem kodów genetycznych (chromosomów). Do każdego łańcucha jest przypisana miara przystosowania (w zadaniach optymalizacji funkcja celu). Multizbiór łańcuchów (populacja) jest losowo wybierany z przestrzeni poszukiwań i poprzez mechanizmy wyboru, krzyżowania i mutacji ewoluuje w kierunku minimum globalnego funkcji celu.

Prosty algorytm genetyczny GA (Genetic Algorithm) [1] jest skonstruowany z następujących operacji:

**Inicjalizacja** - polega na wylosowaniu początkowej populacji  $P$ . Zmienne zadania optymalizacji muszą być zakodowane w postaci skończonego ciągu znaków z pewnego skończonego alfabetu, np. w postaci łańcucha binarnego: 01101 - 1, 11000 - osobnik 2.

**Selekcja** - kopiowanie z poprzedniej populacji do nowej może odbywać się według różnych reguł. Warunkiem jest jednak, aby prawdopodobieństwo wyboru osobników o większej mierze przystosowania było większe (przykład - „koło ruletki”).

**Krzyżowanie** - zazwyczaj przebiega w dwóch etapach. Najpierw kojarzymy losowo w pary ciągi z populacji rodziców. Następnie, w sposób losowy, z jednakowym prawdopodobieństwem, wybieramy miejsce przecięcia  $k$  spośród 1-1 początkowych pozycji w ciągu kodowym (np. dla osobników 01101 i 11100 oraz  $k=3$  otrzymujemy osobniki 01100 i 11101).

**Mutacja** - polega na losowej zmianie jednego bitu ciągu z populacji (odwrócenie bitu).

Krzyżowanie i mutacja są nazywane operacjami genetycznymi. Populacja potomna generowana jest w następujący sposób:

1. **Ewaluacja** – Dla każdego osobnika obliczana jest wartość funkcji przystosowania osobnika.

2. **Selekcja** – Wybór osobników do populacji potomnej, na podstawie ich przystosowania.
3. **Krzyżowanie** – Osobniki wybrane w selekcji są krzyżowane z pewnym prawdopodobieństwem (*prawdopodobieństwo krzyżowania*).
4. **Mutacja** - Każdy allel osobnika z populacji potomnej może zmutować z pewnym prawdopodobieństwem (*prawdopodobieństwo mutacji*)

## 1.1. Kodowanie

Ważną część algorytmu genetycznego stanowi kodowanie informacji o osobniku do postaci ciągu binarnego. Ponieważ celem algorytmu jest minimalizacja  $n$ -wymiarowej funkcji rzeczywistoliczbowej, pojedynczy osobnik w algorytmie reprezentuje pewien punkt w przestrzeni  $\mathbb{R}^n$ . Konieczne jest więc zakodowanie wektora  $(x_{n-1}, x_{n-2}, \dots, x_0)$  liczb rzeczywistych do ciągu binarnego  $(b_{m-1}, b_{m-2}, \dots, b_0)$ ,  $b_i \in \{0, 1\}$ . Kolejne zmienne rzeczywiste  $x_j$  są kodowane do postaci podciągów binarnych, podciągi są następnie konkatenowane do jednego ciągu binarnego. W algorytmie realizowane jest podwójne kodowanie zmiennoprzecinkowych liczb rzeczywistych do postaci binarnej.

- Kodowanie stało-przecinkowe (fix-point)  $E_{fp}$   
Zakłada się, że wartość liczby rzeczywistej będziemy kodować na  $K$  bitach. Dodatkowo liczba rzeczywista  $x_i$  musi należeć do pewnego przedziału  $lb \leq x_i \leq ub$ . Do odkodowania liczby rzeczywistej, zakodowanej przez  $E_{fp}$  stosuje się wzór:

$$x := lb + E_{fp}(x) \cdot \frac{ub - lb}{2^K - 1}$$

Kodowanie stało-przecinkowe  $E_{fp}$  zapewnia że wszystkie wartości z przedziału  $(l, u)$  mają odwzorowanie  $1 - 1$  w przestrzeni  $K$ -bitowych kodów binarnych.

- Kodowanie Graya  $E_G$   
Kodowanie Graya przekształca standardowy kod binarny na kod Gray. Kod Graya charakteryzuje się tym, że dwa kolejne słowa kodowe różnią się tylko stanem jednego bitu. Jest również kodem cyklicznym. Dzięki tej właściwości niewielkie różnice chromosomów wiążą się z niewielkimi różnicami w fenotypie osobników (co nie byłoby prawdą w przypadku zwykłego kodu binarnego).

Liczba rzeczywista kodowana jest najpierw stało-przecinkowo, a następnie do postaci kodu Graya. Kodowanie binarne w programie to :  $E := E_G \circ E_{fp}$

## 1.2. Selekcja

Selekcja to sposób wyboru osobników do populacji potomnej. Istnieje wiele metod selekcji ogólnie stosowanych w algorytmach genetycznych. W programie zaimplementowano trzy metody selekcji.

- **Proporcjonalna**

Metoda nazywana metodą ruletki. Osobnik wybierany jest do nowej populacji z prawdopodobieństwem proporcjonalnym do wielkości jego przystosowania.

- **Rankingowa**

W programie zaimplementowano uproszczoną wersję metody rankingowej. Do populacji potomnej wybierane jest  $k$  najlepiej przystosowanych osobników i są oni powielani do osiągnięcia pożądanej wielkości populacji potomnej.

- **Turniejowa**

Metoda turniejowa polega na przeprowadzeniu  $n$  turniejów na  $k$  osobnikach z populacji.  $n$  to rozmiar populacji potomnej, a  $k$  rozmiar turnieju. Uczestnicy każdego turnieju są wybierani losowo z populacji macierzystej. Zwycięzca każdego z turniejów, czyli najlepiej przystosowany osobnik zostaje wybrany do populacji potomnej. W programie zaimplementowano zmodyfikowaną wersję metody turniejowej. W kolejnych  $l$ -tym turnieju brany jest pod uwagę najlepszy osobnik wyłoniony w  $l - 1$  przeprowadzonych turniejach. Metoda powoduje, że populacja ma małą różnorodności genetyczną (efekt niepożądany w alg. genetycznym), jednak dla większych wymiarów funkcji testowych, daje najlepsze wyniki.

## 2. Funkcje testowe

Badania prowadzono dla dwóch funkcji testowych, dla różnych wymiarów zadania ( $n = 2, 10, 20, 50, 100$ ).

### 2.1. Funkcja Griewanka

Określona wzorem:

$$GR(x_i) = \frac{1}{40} \sum_{i=1}^n (x_i^2) + 1 - \prod_{i=1}^n \cos\left(\frac{x_i}{i}\right)$$

Przy ograniczeniach zmiennych:

$$-40 \leq x_i \leq 40$$

Przyjmuje minimum globalne w punkcie:

$$f_{min} = 0, \text{ dla } x_i = 0$$

## 2.1. Funkcja Ackleya

Określona wzorem:

$$AC(x_i) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^n x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^n \cos(2\pi x_i)\right) + 20 + e$$

Przy ograniczeniach zmiennych:

$$-30 \leq x_i \leq 30$$

Przyjmuje minimum globalne w punkcie:

$$f_{min} = 0, \text{ dla } x_i = 0$$

## 3. Program

W ramach projektu powstał program do minimalizacji wielowymiarowych funkcji rzeczywistych za pomocą algorytmu genetycznego wykorzystującego kodowanie binarne. Program został napisany w języku C w standardzie GNU99 i skompilowany za pomocą kompilatora GCC 4.8.2 w środowisku linux. Wersja równoległa programu wykorzystuje bibliotekę OpenMP 3.1.

### 3.1. Parametry

Program uruchamiany jest z następującymi parametrami:

1. Parametry zadania:
  - Wybór funkcji do minimalizacji
    - funkcja Griewanka
    - funkcja Ackleya
  - Wymiar funkcji (1..100)
2. Parametry algorytmu genetycznego
  - Całkowitoliczbowe
    - Ilość generacji
    - Rozmiar populacji
    - Parametr metody selekcji
  - Rzeczywistoliczbowe
    - Prawdopodobieństwo krzyżowania
    - Prawdopodobieństwo mutacji
    - Parametr metody selekcji

- Metoda selekcji
  - Proporcjonalna (ruletka)
  - Rankingowa
  - Turniejowa
- 3. Parametry wykonania
  - Verbose – komunikaty
  - Seed – ziarno dla losowania liczb pseudolosowych

### 3.2. Struktura

Główną częścią programu jest pętla powtarzana, aż do wykonania zadanej ilości generacji. W pętli realizowane są kolejne kroki algorytmu genetycznego.

**For** generacja = 0 to *MAX\_GENERACJA* **do**

selekcja	/* wybierz osobniki do nowej populacji */
generacja	/* generuj nową populację na podstawie selekcji */
krzyżowanie	/* krzyżuj losowo wybrane osobniki w populacji potomnej */
mutacja	/* mutuj losowe geny */
ewaluacja	/* oblicz przystosowanie osobników z nowej populacji */
zapamiętaj najlepszy	/* zapamiętaj najlepiej przystosowanego osobnika */

**endfor**

### 3.3. Wywołanie

Program wywołujemy poleceniem `./[omp]release` (`openmprelease` dla wersji z OpenMP). W celu zadania wartości parametrom biblioteki OpenMP należy poprzedzić wywołanie programu komendą `env`,

`env PARAMETR_OMP="<wartość parametru>" ./[omp]release`

GENALG – algorytm genetyczny do minimalizacji funkcji Griewanka i Ackleya

użycie: GENALG [argumenty]

Argumenty:

NAZWA	OPIS	DOMYŚLNA
-h	Wyświetla tą wiadomość	
-v	verbose – dodatkowe komunikaty	
-d[N]	Wymiar funkcji	2
-e[N]	Seed dla randomizacji	0
-g[N]	Maksymalna ilość generacji	50
-k[N]	Parametr metody selekcji [INT]	2
-r[N]	Parametr metody selekcji [FLOAT]	0.5
-p[N]	Rozmiar populacji	20
-m[p]	Prawd. mutacji allele	0.01
-x[p]	Prawd. krzyżowania dwóch osobników	0.2
-s[R T B]	Metoda selekcji osobników	B
-f[GR AC]	Funkcja do minimalizacji	GA

Wypis programu po uruchomieniu w konsoli z flagą -h (help)

## 4. Realizacja równoległa

Do zrównoleglenia algorytmu zastosowano dyrektywy OpenMP. Zrównoleglono najważniejsze pętle występujące w programie. W pętlach wykonywane są operacje na wszystkich osobnikach w obecnej populacji

- Ewaluacja populacji – obliczanie przystosowania osobników
  - dekodowanie genotypu
  - obliczanie wartości funkcji testowej
- Mutacja
- Krzyżowanie
- Selekcja
  - Metoda proporcjonalna
  - Metoda turniejowa(Nie zrównoleglono metody rankingowej. Zrównoleglenie metody wymagałoby zupełnie innego podejścia implementacyjnego)

Zastosowano różne możliwości rozdzielania pętli:

- schedule (static)
- schedule (dynamic)

Zrównoleglano działanie pętli *for* w funkcjach wykonujących operacje na populacji (plik *population.c*). Poniżej przedstawiony jest ogólny schemat wykorzystania dyrektyw OpenMP w programie. Metoda zrównoleglenia różni się w zależności od tego, czy w pętli generowane są liczby pseudolosowe, i czy aktualizowana jest wartość pewnych dzielonych zmiennych skalarnych:

#### **#pragma omp parallel**

```
{  
    // Jeżeli w pętli for będą generowane liczby pseudolosowe,  
    // to dla każdego wątku należy wyliczyć stan generatora pseudolosowego  
    // i zapisać go w zmiennej prywatnej każdego wątku.
```

#### **#pragma omp for**

```
for i ← osobniki w populacji {  
    // wykonaj funkcję na pojedynczym osobniku  
  
    #pragma omp critical  
    {  
        // aktualizuj wartość skalarnej zmiennej dzielonej (np. przystosowanie  
        // najlepszego osobnika)  
    }  
} /* end for */
```

#### **#pragma omp master**

```
{  
    // jeżeli w pętli generowano liczby pseudolosowe, to przypisz globalnemu  
    // stanowi generatora losowego, wartość jaką osiągnął prywatny stan generatora  
    // wątku głównego po wykonaniu pętli for.  
}  
}
```

## **5. Testy algorytmu genetycznego**

Wykonano testy algorytmu genetycznego dla funkcji Griewanka i Ackleya. Rozważano różne wymiary zadania ( $n = 2, 10, 20, 50, 100$ ).

Wszystkie testy wykonano na 4-rdzeniowym komputerze o procesorze Intel Core 2 Quadro, z pamięcią 4 GB RAM, pod systemem Linux Ubuntu 11.10

Ze względu na występowanie w algorytmie losowania (przy każdym uruchomieniu jest losowana inna populacja) większość testów została uruchomiona 5 razy. W tabelach z wynikami podane są wyniki: średni z pięciu uruchomień, najlepszy i najgorszy.

Badano dokładność rozwiązania (wyznaczone minimum globalne) oraz wydajność algorytmu.

W tabelach i na rysunkach prezentowane są wyniki badań. Symbole w tabelach oznaczają:

- $n$  – wymiar zadania,
- $P$  – licznosc populacji,
- $G(min)$  – najlepszy wynik z 5 uruchomień (wartosc funkcji testowej w minimum),
- $G(max)$  – najgorszy wynik z 5 uruchomień (wartosc funkcji testowej w minimum),
- $G(ev)$  – uśredniony wynik z 5 uruchomień,
- $T(ev)$  – uśredniony czas obliczeń,

## 1.1 Algorytm sekwencyjny

Celem testów algorytmu sekwencyjnego było:

- Zbadanie dokladności rozwiązania dla różnych wymiarów zadania.
- Zbadanie wydajności algorytmu dla różnych wymiarów zadania.
- Zbadanie wpływu rozmiaru populacji na rozwiązanie.
- Porównanie rozwiázan i wydajności algorytmu dla różnych metod selekcji.

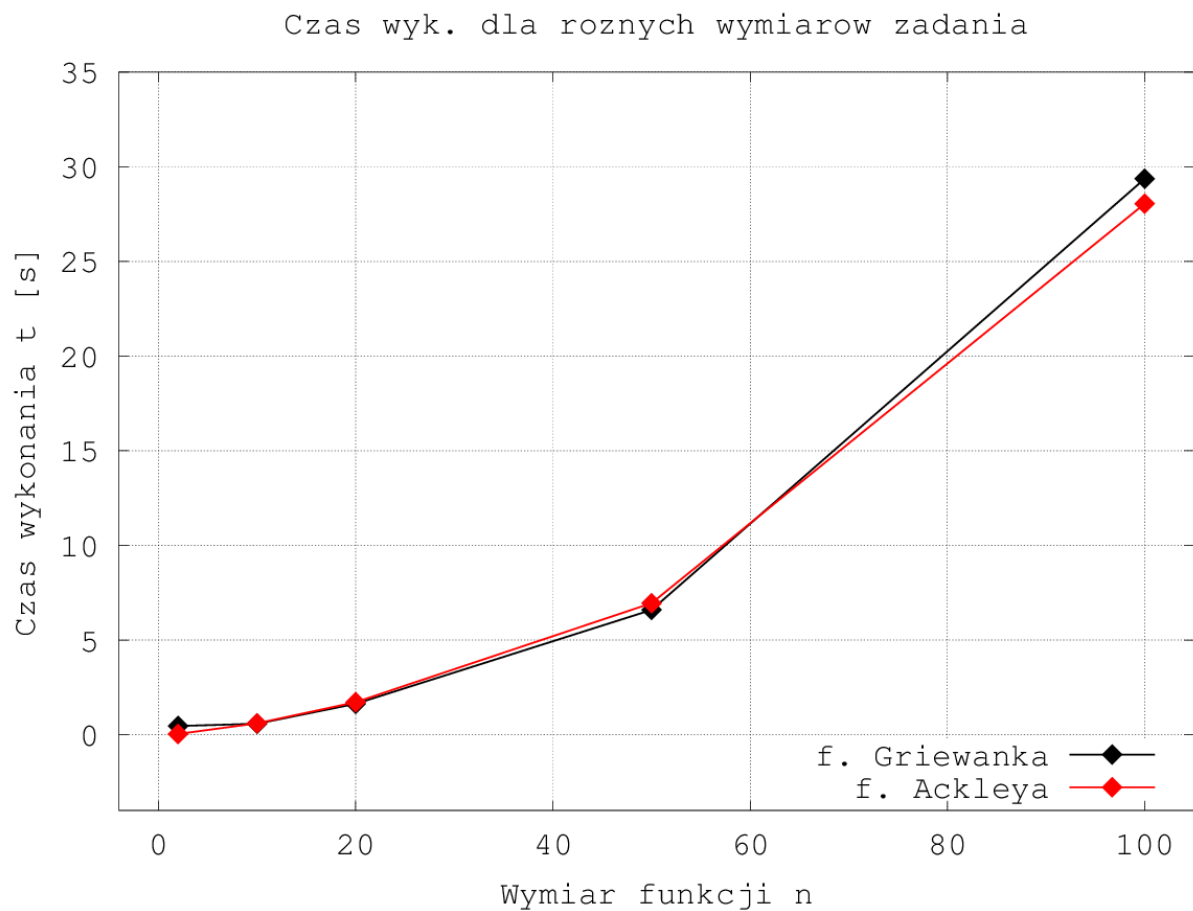
Dla wszystkich testów wykonano 100 iteracji algorytmu genetycznego. Przyjęto prawdopodobieństwo krzyżowania równe 0.25 i prawdopodobieństwo mutacji 0.003.

<i>Funkcja Griewanka</i>					
$n$	<i>populacja</i>	$G (min)$	$G (max)$	$G (ev)$	$T (ev)$
2	500	0.0	1.05	0.21	0.46
10	1000	0.0	0.0	0.0	0.58
20	2000	0.0	0.0	0.0	1.63
50	5000	0.13	1.05	0.51	6.59
100	10000	5.66	8.71	7.30	29.37

<i>Funkcja Ackleya</i>					
$n$	<i>populacja</i>	$G (min)$	$G (max)$	$G (ev)$	$T (ev)$
2	500	0.0	6.56	2.02	0.04
10	2000	0.02	2.01	0.41	0.6
20	3000	0.01	0.02	0.01	1.72
50	5000	2.79	3.93	3.3	6.95
100	10000	6.16	10.82	8.21	28.05

Tabela 1: Wyniki algorytmu (wyznaczone rozwiązania i czasy działania) dla różnych wymiarów zadania, selekcja turniejowa.



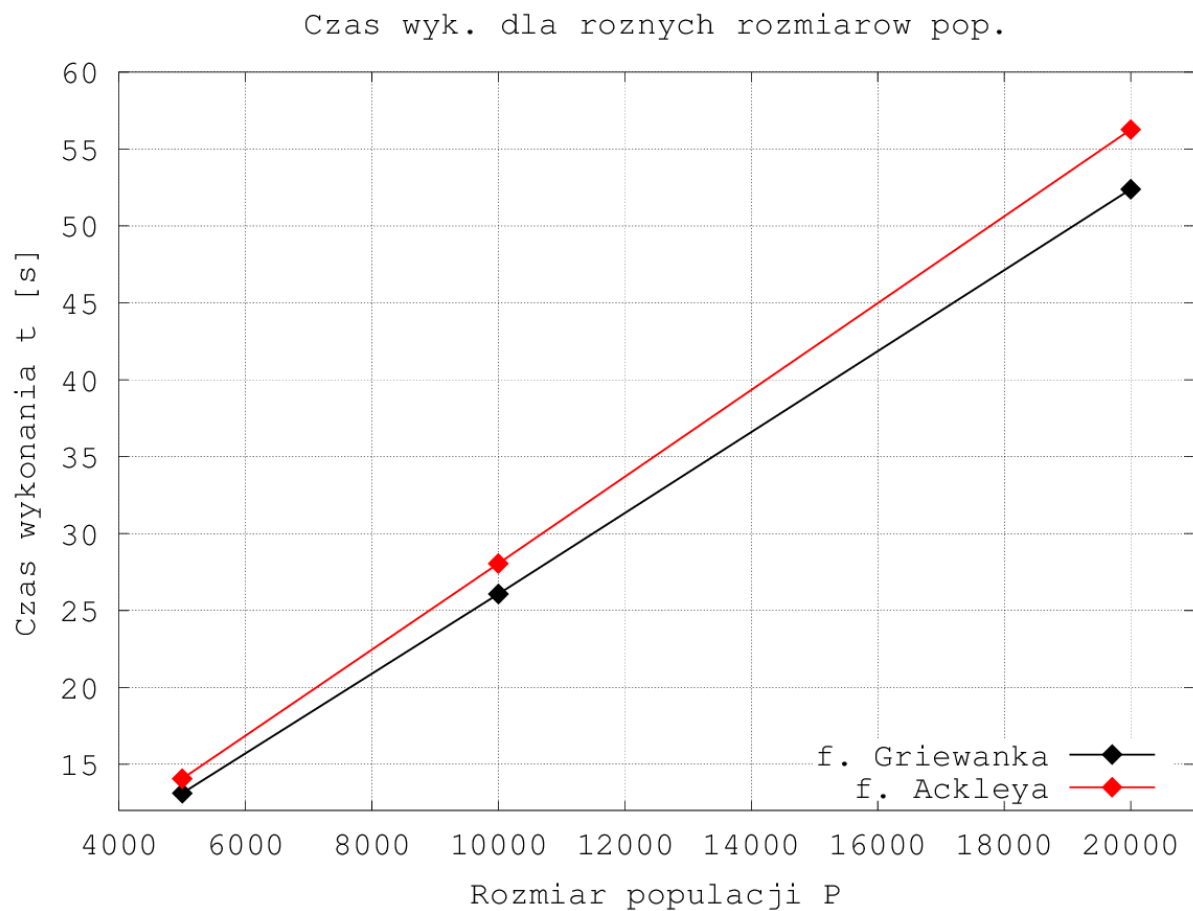


Rysunek 1: Czas wykonania funkcji testowych dla różnych wymiarów zadania

<i>Funkcja Griewanka</i>				
<i>P</i>	<i>G (min)</i>	<i>G (max)</i>	<i>G (ev)</i>	<i>T (ev)</i>
5000	10.58	20.4	12.96	13.12
10000	5.17	8.65	7.02	26.07
20000	2.27	3.84	3.41	52.39

<i>Funkcja Ackleya</i>				
<i>P</i>	<i>G (min)</i>	<i>G (max)</i>	<i>G (ev)</i>	<i>T (ev)</i>
5000	8.8	11.35	9.84	14.06
10000	6.52	9.48	8.11	28.05
20000	5.7	6.23	5.88	56.27

Tabela 2: Wpływ liczności populacji na jakość rozwiązania i czas obliczeń ( $n=100$ , selekcja turniejowa).



Rysunek 1: Czas wykonania funkcji testowych dla różnych rozmiarów populacji

<i>Funkcja Griewanka</i>				
<i>selekcja</i>	<i>G (min)</i>	<i>G (max)</i>	<i>G (ev)</i>	<i>T (ev)</i>
Proporcjonalna	644.33	735.47	701.62	31.64
Rankingowa	124.96	155.90	143.26	26.59
Turniejowa	4.16	9.88	6.28	26.15

<i>Funkcja Ackleya</i>				
<i>selekcja</i>	<i>G (min)</i>	<i>G (max)</i>	<i>G (ev)</i>	<i>T (ev)</i>
Proporcjonalna	20.9	20.7	20.94	33.79
Rankingowa	17.46	17.65	17.59	29.04
Turniejowa	7.39	8.55	7.71	28.34

Tabela 3: Wpływ metody selekcji na jakość rozwiązania i czas obliczeń ( $n=100$ ,  $P=10000$ ).

## Wnioski

Najlepsze wyniki otrzymano korzystając z turniejowej metody selekcji. Metoda ta ma też najszybszy czas wykonania. Jakość wyników spadała wraz ze wzrostem wymiarowości funkcji testowych. Wyniki próbowano poprawiać, zwiększając populację, co przyniosło względnie dobre efekty. Wpływ wielkości populacji na jakość wyników jest widoczny w *Tabeli 2*. Można jednak zauważyć, że zwiększanie populacji, przy wzroście wymiarowości zadania, znacznie spowalnia czas działania programu (*Tabela 1*).

Na podstawie otrzymanych wyników, w celu otrzymania jak najbardziej zadowolającego rozwiązania, zaleca się korzystanie z turniejowej metody selekcji. Należy dobrać wielkość populacji stosownie do wymiaru zadania. Warto mieć jednak na uwadze fakt, że metoda turniejowa; mimo iż zapewnia najszybszą zbieżność algorytmu; może okazać się niewydolna, przy minimalizacji funkcji z bardziej wydatnymi minimami lokalnymi. Wynika to z małej różnorodności genetycznej, którą metoda wprowadza do populacji.

### 1.2 Algorytm równoległy

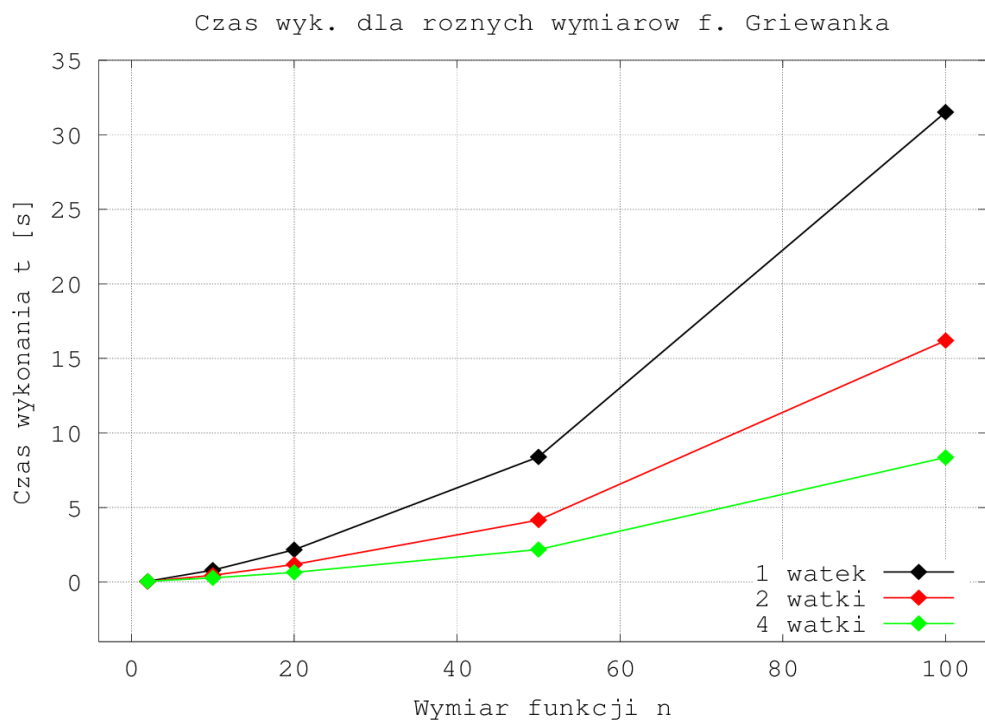
Celem testów algorytmu sekwencyjnego było:

- Zbadanie wpływu zrównoleglenia algorytmu na wydajność w zależności od wymiaru i parametrów zadania.
- Zbadanie wpływu metody rozdziału pętli „for” na szybkość rozwiązania.

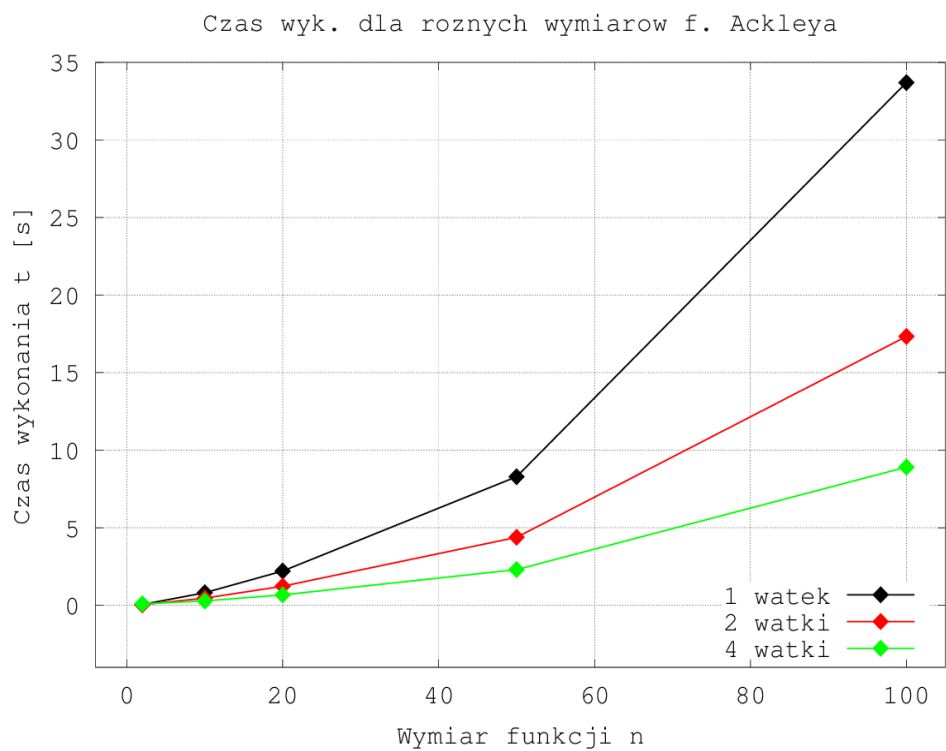
<i>Funkcja Griewanka</i>							
<i>n</i>	<i>P</i>	<i>Sekwencyjny</i>		<i>Równoległy (2 wątki)</i>		<i>Równoległy (4 wątki)</i>	
		<i>T (ev)</i>	<i>G (ev)</i>	<i>T (ev)</i>	<i>G (ev)</i>	<i>T (ev)</i>	<i>G (ev)</i>
5	500	0.05	0.0	0.04	0.0	0.05	0.0
10	2000	0.8	4.95	0.45	5.18	0.28	4.57
20	3000	2.17	32.63	1.18	30.71	0.65	30.54
50	5000	8.39	223.29	4.16	244.29	2.18	217.01
100	10000	31.52	696.13	16.2	702.25	8.36	722.71

<i>Funkcja Ackleya</i>							
<i>n</i>	<i>P</i>	<i>Sekwencyjny</i>		<i>Równoległy (2 wątki)</i>		<i>Równoległy (4 wątki)</i>	
		<i>T (ev)</i>	<i>G (ev)</i>	<i>T (ev)</i>	<i>G (ev)</i>	<i>T (ev)</i>	<i>G (ev)</i>
5	500	0.05	0.02	0.04	0.01	0.09	0.03
10	2000	0.81	3.58	0.46	4.16	0.28	3.99
20	3000	2.21	10.74	1.23	11	0.67	11.23
50	5000	8.29	20.26	4.38	20.38	2.3	20.33
100	10000	33.7	33.7	17.33	20.94	8.91	20.94

Tabela 4: Porównanie wydajności algorytmu sekwencyjnego i równoległego (różne wymiary zadania i liczności populacji, selekcja proporcjonalna).



Rysunek 1: Czas wykonania funkcji testowych dla różnych wymiarów zadania, porównanie zrównoleglenia – funkcja Griewanka

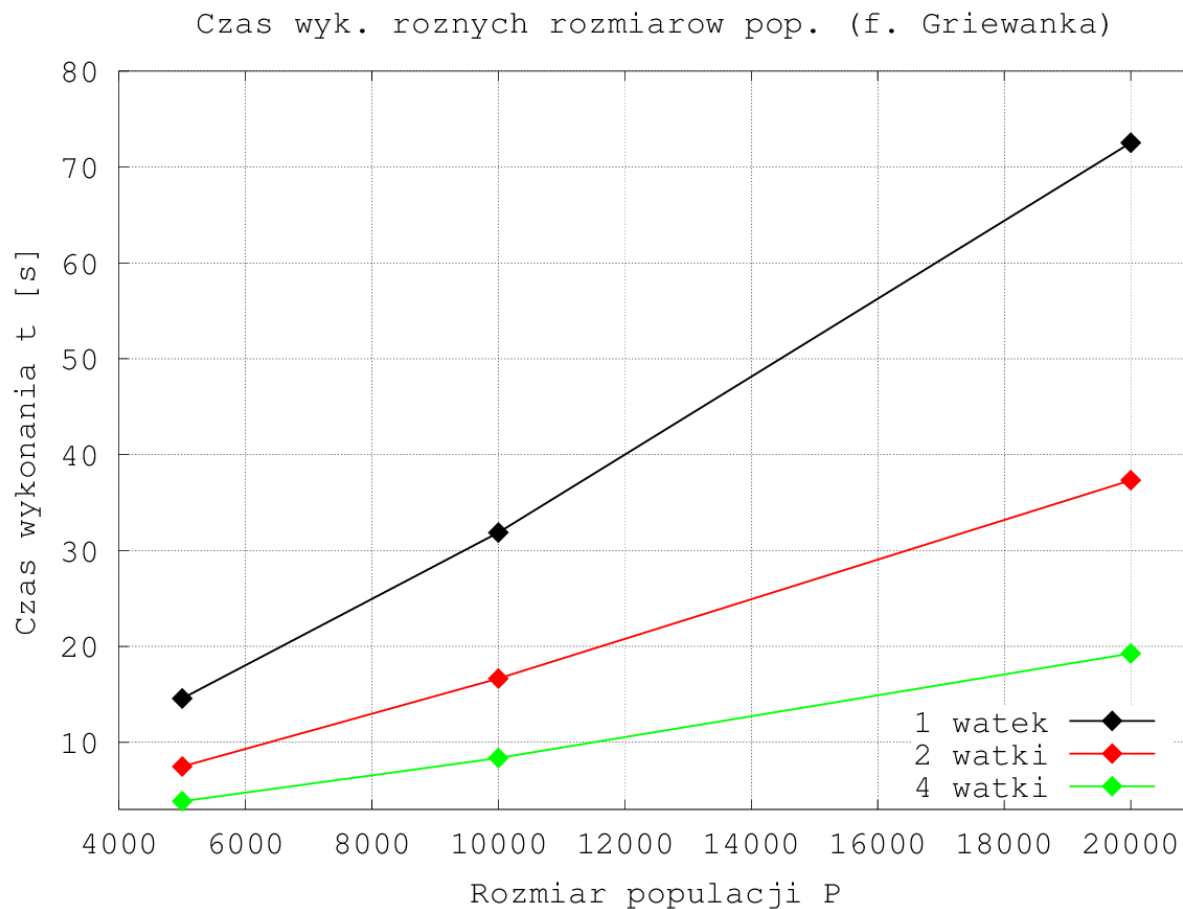


Rysunek 1: Czas wykonania funkcji testowych dla różnych wymiarów zadania, porównanie zrównoleglenia – funkcja Ackleya

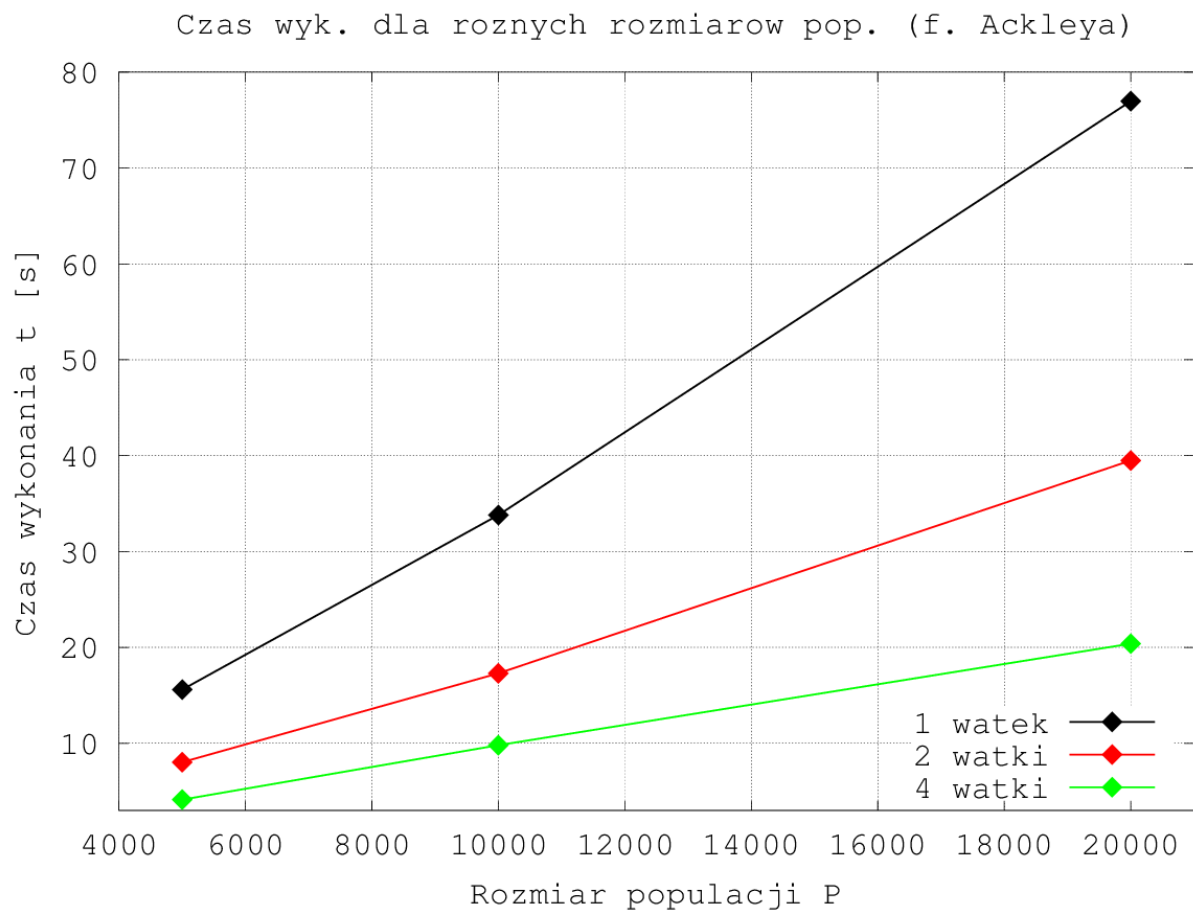
Funkcja Griewanka						
P	Sekwencyjny		Równoległy (2 wątki)		Równoległy (4 wątki)	
	T (ev)	G (ev)	T (ev)	G (ev)	T (ev)	G (ev)
5000	14.58	723.8	7.47	711.08	3.85	693
10000	31.87	707.02	16.65	688.5	8.36	704.54
20000	72.53	685.46	37.33	684.17	19.27	683.24

Funkcja Ackleya						
P	Sekwencyjny		Równoległy (2 wątki)		Równoległy (4 wątki)	
	T (ev)	G (ev)	T (ev)	G (ev)	T (ev)	G (ev)
5000	15.6	20.95	8.03	20.95	4.12	20.95
10000	33.81	20.95	17.3	20.94	9.8	20.92
20000	76.97	20.94	39.5	20.93	20.39	20.92

Tabela 5: Zależność przyspieszenia obliczeń od liczności populacji ( $n=100$ , selekcja proporcjonalna).



Rysunek 1: Czas wykonania funkcji testowych dla różnych rozmiarów populacji, porównanie zrównoleglenia – funkcja Griewanka



Rysunek 1: Czas wykonania funkcji testowych dla różnych rozmiarów populacji, porównanie zrównoleglenia – funkcja Ackleya

<i>Funkcja Griewanka</i>	
<i>Schedule</i>	<i>T (ev)</i>
static	6.96
dynamic	7.07
static, kwant = 1000	8.14
dynamic, kwant = 1000	8.08

Tabela 6: Wpływ metody rozdziału pętli na przyspieszenie obliczeń w wersji równoległej ( $n=100$ ,  $P=10000$ , selekcja proporcjonalna, 4 wątki)

## Wnioski

Zrównoleglenie programu za pomocą dyrektyw biblioteki OpenMP przyniosło zadowalające efekty. Otrzymane przyspieszenie jest bliskie  $p$ -krotnemu, gdzie  $p$  to liczba rdzeni (oraz tworzonych wątków) na maszynie testowej i wynosi średnio 3.79. Przyspieszenie działania programu zrównoleglonego zaczyna być widoczne już od  $n$  większego do 10 i rozmiaru populacji  $P$  ponad 2000. Dla problemu mniejszej skali tworzenie wątków nie jest opłacalne (Tabela 4). Dla wszystkich wielkości populacji w Tabeli 5 obserwowane jest podobne przyspieszenie wersji równoległej, dla obu funkcji testowych. Ingerowanie w sposób podziału pracy pętli for (Tabela 6) okazało się nieopłacalne. Najlepsze wyniki daje standardowy tryb *static*. Wynika to z faktu, że wszystkie iteracje pętli powinny mieć podobny czas wykonania i kompilator jest w stanie dobrze rozdzielić zadania już w czasie kompilacji.

Uzyskane przyspieszenie obliczeń  $PS$  wyznaczone z wzoru:

$$PS = \frac{\text{czas obliczeń (na jednym wątku)}}{\text{czas obliczeń (na czterech wątkach)}} \simeq 3.79 \leq 4$$

## Uwaga

Ponieważ algorytm genetyczny jest algorytm probabilistycznym, jednym z ważniejszych narzędzi koniecznych do jego poprawnego działania, jest generator liczb pseudolosowych. W równoległej wersji programu należało zaistniała potrzeba skorzystania z bezpieczniejszej wątkowo funkcji generującej liczby pseudolosowe. Popularna funkcja biblioteki standardowej języka C – *rand()* nie spełnia tych wymagań. Stan generatora tej funkcji jest zmienną globalną, niechronioną żadnymi narzędziami synchronizacji. Użycie tej funkcji wraz z dyrektywami OpenMP skutkuje bardzo wolnym działaniu programu. Dlatego w programie użyto funkcji z rodziny *dran48*, które stan generatora dostają w argumencie wywołania.

## Bibliografia

- [1] Michalewicz Z., Algorytmy genetyczne + struktury danych = programy ewolucyjne, WNT, 1996.