МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра СТ

Звіт

з лабораторної роботи №1

з дисципліни «Нейромережі та генетичні алгоритми»

з теми «Оптимізація складних функцій за допомогою генетичних алгоритмів»

|  |  |
| --- | --- |
| Виконав:  ст. гр. КСУАм-14-1  Щербатенко В.С. | Перевірила:  Імангулова З.А. |

Харків 2014

1 ОПТИМІЗАЦІЯ СКЛАДНИХ ФУНКЦІЙ ЗА ДОПОМОГОЮ ГЕНЕТИЧНИХ АЛГОРИТМІВ

1.1 Мета роботи

Вивчення принципів роботи генетичних алгоритмів. Набуття навичок реалізації генетичних алгоритмів для оптимізації складних багатопараметричних функцій.

1.2 Постановка задачі та вхідні дані

В ході роботи необхідно реалізувати генетичний алгоритм для розв’язку оптимізаційної задачі пошуку максимального значення функції. Основні принципи ГА використовують пряму аналогію з біологічними процесами і грунтуються на таких механізмах: схрещування, мутація та добір (рис. 1.1). Особливості конкретної реалізації наведені в індивідуальному завданні (табл. 1.1).



Рисунок 1.1 – Структурна схема генетичного алгоритму

#### Таблиця 1.1 – Індивідуальне завдання

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Варіант | Функція | Схрещування | Вибір батьків | Механізм добору |
| 11 |  | Двоточкове | Інбридинг | З витис-ненням |

1.3 Хід роботи

Для розв’язку оптимізаційної задачі написали програму, яка реалізовує генетичний алгоритм. Код основних механізмів приведено на рисунках 1.2–1.5.

|  |
| --- |
| **Population p = new Population(50, true);**  int **generations** = -1;  Individual **fittest = p**.getFittest();  double **max** = **fittest**.getFunc();  **do {**  generations++;  Individual **individual** = p.getFittest();  **if (max** < **individual**.getFunc()**) {**  **fittest** = **individual**;  **max** = **individual**.getFunc();  System.out.print("Generation: " + **generations** + "\t");  System.out.println("Fittest: " + **fittest**);  **}**  p = GeneticAlgorithm.**evolve**(p);  **} while (generations < 50);**  System.out.println("Generations used: " + **generations**);  System.out.println("Best solution:");  System.out.println(**fittest**); |

Рисунок 1.2 – Ітераційний механізм зміни поколінь

|  |
| --- |
| public static Population **evolve** (Population **population**) {  Population **nextGeneration** = new Population();  **for** (int i = 0; i < **population**.size(); i++) {  Individual **child**;  **do {**  Individual **parent1** = **randomSelection**(**population**);  Individual **parent2** = **inbreedingSelection**(**population**, **parent1**);  **child** = **parent1**.**crossover**(**parent2**);  **child**.**mutate**();  } **while**(**nextGeneration**.**contains**(**child**));  **nextGeneration**.**saveIndividual**(i, **child**);  }  return **nextGeneration**;  } |

Рисунок 1.3 – Еволюційний механізм з витисненням

|  |
| --- |
| private static Individual **inbreedingSelection**(  Population **population**,  Individual **firstParent**)  {  Individual **closest** = **population**.getIndividual(0);  double **min** = Double.MAX\_VALUE;  Individual **individual**;  **for** (int i = 1; i < **genes.SIZE**.size(); i++) {  **individual** = **population**.getIndividual(i);  if (**individual**.**equals**(**closest**))  continue;  double **dist** = **firstParent**.**getDistance**(**individual**);  **if** (**dist** < min) {  **min** = **dist**;  **closest** = **individual**;  }  }  return **closest**;  } |

Рисунок 1.4 – Реалізація стратегії вибору другого батька «Інбридинг»

|  |
| --- |
| public Individual **crossover**(Individual **other**) {  int **start** = **Random**.nextInt(**genes.SIZE**);  int **length** = **Random**.nextInt(**genes.SIZE**);  long **mask** = 0;  **for** (int **i** = **start**; **i** < **start** + **length**; **i**++) {  **mask** |= 1 << **i** % **genes.SIZE**;  }  long **newGenes** = 0;  **newGenes** |= **mask & this.genes**;  **newGenes** |= **~mask & other.genes**;  return **new** Individual(**newGenes**);  }  public void **mutate**() {  for (int **i** = 0; **i** < **genes.SIZE**; **i**++) {  if (**Random**.nextDouble() <= **MUTATION**\_**RATE**) {  **if** (((1 << **i**) & **genes**) == 0)  **genes** = **genes** | (1L << **i**);  **else**  **genes** = **genes** ^ (1L << **i**);  }  }  } |

Рисунок 1.5 – Механізми двоточкового схрещування та мутації

Результат роботи програми приведено на рисунку 1.6.

|  |
| --- |
| Generation: 3 Fittest: x1 = 0.517, x2 = 4.356, func = 36.708  Generation: 26 Fittest: x1 = 0.577, x2 = 4.539, func = 48.312  Generation: 29 Fittest: x1 = 0.915, x2 = 4.511, func = 50.608  Generation: 30 Fittest: x1 = 0.594, x2 = 4.595, func = 53.998  Generation: 31 Fittest: x1 = 0.594, x2 = 4.595, func = 54.887  Generation: 35 Fittest: x1 = 0.594, x2 = 4.595, func = 54.887  Generation: 37 Fittest: x1 = 0.632, x2 = 4.595, func = 57.030  Generations used: 50  Best solution:  x1 = 0.632, x2 = 4.595, func = 57.030 |

Рисунок 1.6 – Результат роботи програми

ВИСНОВКИ

В ході лабораторної роботи було:

* розроблено ітераційний механізм еволюції популяцій;
* реалізовано механізм добору батьків «інбридинг»;
* реалізовано механізми двоточкового схрещування та мутації;
* проведено оптимізацію задачі пошуку максимального значення функції за допомогою реалізованого генетичного алгоритму.