Bartosz Ujazdowski - 202322

Algorytmy

Selekcja

```
        int adapt[N];
        int sumOfAdaptaion = 0;
        for (int i = 0; i < N; i++)
        {
            adapt[i] = adaptation(population[i]);
            sumOfAdaptaion += adapt[i];
        }
        double probability[N];
        for(int i = 0; i < N; i++)
        {
            probability[i] = (double)adapt[i] / (double)sumOfAdaptaion * 100.0; ①
        }
        std::vector<population_t> next_population;
        for (int i = 0; i < N; i++)                        ②
        {
            double r = (rand() % 10000) / 100.0;
            double sum = 0.0;
            for (int j = 0; j < N; j++)
            {
                sum += probability[j];
                if ( sum >= r )
                {
                    next_population.push_back(population[j]);
                    break;
                }
            }
        }
```

① Prawdopodobieństwo selekcji

② Algorytm ruletki

Krzyżowanie

```
        for (int i = 0; i < N; i += 2)
        {
            int index = rand() % next_population.size();
            population_t p1 = next_population[index];
            next_population.erase(next_population.begin() + index);
            index = rand() % next_population.size();
            population_t p2 = next_population[index];
            next_population.erase(next_population.begin() + index);
            if ( rand() % 101 > crossing_probability)
            {
                population[i] = p1;
                population[i + 1] = p2;
            } else
            {
                int locus = (rand() % 8) + 1;
                population[i] = cross(p1, p2, locus);
                population[i + 1] = cross(p2, p1, locus);
            }
        }
```

Funkcja Cross

```
population_t cross( population_t p1, population_t p2, int locus )
{
    p1.x = p1.x << 8 - locus;
    p1.x = p1.x >> 8 - locus;
    p2.x = p2.x >> locus;
    p2.x = p2.x << locus;
    p1.x = p1.x | p2.x;
    return p1;
}
```

Mutacja

```
        for ( int i = 0; i < N; i++ )
        {
            if (rand() % 101 <= mutation_probability)
            {
                mutation(population[i], (rand() % 8) + 1);
            }
        }
```

Funkcja mutation

```
population_t mutation(const population_t & population, int locus)
{
    population_t p = population;
    p.x = p.x >> locus - 1;
    p.x = p.x << locus - 1;
    p.x = p.x << 8 - locus;
    p.x = p.x >> 8 - locus;
    if (p.x == 0)
    {
        p.x = 1;
        p.x = p.x << locus -1;
    }
    p.x = p.x ^ population.x;
    return p;
}
```

Algorytm Genetyczny

W ćwiczeniu zajmiemy się optymalizacją funkcji 2(x^2 + 1) na przedziale <0,127> za pomocą algorytmu genetycznego. Naszymi operatorami będą krzyżowanie i mutacja. Krzyżowanie będzie realizowane przez zamianę części wokół losowo wybranego lokusa. Mutacja zmienia losowo wartość bitowej reprezentacji chromosomu(jego genotypu). Zmiennymi którymi będziemy manipulować by dostroić algorytm będą:

- Prawdopodobieństwo krzyżowania
- Prawdopodobieństwo mutacji
- Wielkość populacji
- Ilość iteracji

Wyniki

Wielkość populacji	10	10	10	20	20	20
Prawdopodobieństwo krzyżowania	70%	80%	90%	70%	80%	90%
Prawdopodobieństwo mutacji	5%	10%	15%	5%	10%	15%
Ilość iteracji	20	50	100	20	50	100
Wynik	81	107	114	111	123	127