**Мета роботи**

Ознайомитися з основними теоретичними відомостями, вивчити еволюційні оператори схрещування та мутації, що використовуються при розв’язуванні задач комбінаторної оптимізації.

**Завдання**

Розробити за допомогою пакету Matlab програмне забезпечення для

вирішення задачі комівояжера.

В - 7





**Виконання завдання**

Для виконання лабораторної роботи використовувалася C#.

Код програми

Код програми та допоміжних бібліотек, різних видів схрещувань, мутацій і видбірів можна знайти у репозиторії, а у звіті наведені реалізації класів для даної лабораторної, без базових:

**./GeneticSharp/Domain/Crossovers/CycleCrossover.cs**

public class CycleCrossover : CrossoverBase

{

public CycleCrossover()

: base(2, 2)

{

IsOrdered = true;

}

protected override IList<IChromosome> PerformCross(IList<IChromosome> parents)

{

var parent1 = parents [0];

var parent2 = parents [1];

var cycles = new List<List<int>>();

var offspring1 = parent1.CreateNew();

var offspring2 = parent2.CreateNew();

var parent1Genes = parent1.GetGenes();

var parent2Genes = parent2.GetGenes();

// Search for the cycles.

for (int i = 0; i < parent1.Length; i++)

{

if (!cycles.SelectMany(p => p).Contains(i))

{

var cycle = new List<int>();

CreateCycle(parent1Genes, parent2Genes, i, cycle);

cycles.Add(cycle);

}

}

// Copy the cycles to the offpring.

for (int i = 0; i < cycles.Count; i++)

{

var cycle = cycles[i];

int geneCycleIndex = 0;

if (i % 2 == 0)

{

// Copy cycle index pair: values from Parent 1 and copied to Child 1, and values from Parent 2 will be copied to Child 2.

for (int j = 0; j < cycle.Count; j++)

{

geneCycleIndex = cycle[j];

offspring1.ReplaceGene(geneCycleIndex, parent1Genes[geneCycleIndex]);

offspring2.ReplaceGene(geneCycleIndex, parent2Genes[geneCycleIndex]);

}

}

else

{

// Copy cycle index odd: values from Parent 1 will be copied to Child 2, and values from Parent 1 will be copied to Child 1.

for (int j = 0; j < cycle.Count; j++)

{

geneCycleIndex = cycle[j];

offspring2.ReplaceGene(geneCycleIndex, parent1Genes[geneCycleIndex]);

offspring1.ReplaceGene(geneCycleIndex, parent2Genes[geneCycleIndex]);

}

}

}

return new List<IChromosome>() { offspring1, offspring2 };

}

private void CreateCycle(Gene[] parent1Genes, Gene[] parent2Genes, int geneIndex, List<int> cycle)

{

if(!cycle.Contains(geneIndex))

{

var parent2Gene = parent2Genes[geneIndex];

cycle.Add(geneIndex);

var newGeneIndex = parent1Genes.Select((g, i) => new { Value = g.Value, Index = i }).First(g => g.Value.Equals(parent2Gene.Value));

if (geneIndex != newGeneIndex.Index)

{

CreateCycle(parent1Genes, parent2Genes, newGeneIndex.Index, cycle);

}

}

}

}

**./Labworks.Framework/OnePointMutation.cs**

public class OnePointMutation : MutationBase

{

public OnePointMutation()

{

IsOrdered = true;

}

protected override void PerformMutate(IChromosome chromosome, float probability)

{

if (RandomizationProvider.Current.GetDouble() <= probability)

{

var indexPrimary = RandomizationProvider.Current.GetInt(0, chromosome.Length - 1);

var indexSecondary = indexPrimary + 1;

var genePrimary = chromosome.GetGene(indexPrimary);

var geneSecondary = chromosome.GetGene(indexSecondary);

chromosome.ReplaceGene(indexPrimary, genePrimary);

chromosome.ReplaceGene(indexSecondary, geneSecondary);

}

}

}

**Результати виконання завдання**

Результати для кількості міст: 16 (для 32, та 64 результати наведені в табл. 1.):

Рис. 1. Найкращий результати 1-ого покоління

Рис. 2. Найкращий результати 5-ого покоління

Рис. 3. Найкращий результати 25-ого покоління

Рис. 4. Найкращий результати 100-ого покоління

Рис. 5. Найкращий результати 200-ого покоління (останнього)

Табл. 1. Порівняння залежності кількості міст і популяції

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Кількість міст** | ***16*** | | | ***32*** | | | ***64*** | | |
| **Популяція** | 16 | 32 | 64 | 16 | 32 | 64 | 16 | 32 | 64 |
| **Час виконання,c** | 8.67 | 12.1 | 15.44 | 12.97 | 14.77 | 17.84 | 18.11 | 19.12 | 21.5 |
| **Мін. довжина** | 3138 | 3336 | 3451 | 6136 | 6526 | 6421 | 10451 | 10961 | 10925 |
| **Кількість**  **поколінь** | 201 | 238 | 251 | 261 | 222 | 269 | 258 | 289 | 256 |

**Висновки**

Виконавши лабораторну роботу я вивчив еволюційні оператори схрещування та мутації, що використовуються при розв’язуванні задач комбінаторної оптимізації. Реалізував за допомогою пакету С# програмне забезпечення для вирішення задачі комівояжера з циклічним схрещуванням і одноточковою мутацією. В результаті програма коректно працює для кількість міст до 16, 32, 64. Знайдені шляхи оптимальні.

Під час тестування виявив, що одноточечний оператор обміну є менш ефективним, ніж випадковий (швидше отримав оптимальний шлях).