Гриценко О.И.
ИУ5-81

РЕАЛИЗАЦИЯ МНОЖЕСТВА АЛГОРИТМОВ СКРЕЩИВАНИЯ, СЕЛЕКЦИИ, МУТАЦИИ И РЕДУКЦИИ И ИХ ЭКСПЕРИМЕНТАЛЬНОЕ СРАВНЕНИЕ ДЛЯ РЕШЕНИЯ КОНКРЕТНЫХ ЗАДАЧ

Введение

Основной механизм эволюции - это естественный отбор. Он представляет собой процесс, приводящий к выживанию и преимущественному размножению наиболее приспособленных к данным условиям среды особей, которые, следовательно, приносят больше потомства, чем плохо приспособленные. А так как потомки наследуют генетическую информацию своих родителей, то потомки более сильных индивидуумов так же будут более приспособленными. Таким образом увеличивается количество особей, обладающих наиболее полезными для выживания в данных условиях признаками, а значит, приспособленность вида в целом будет с течением времени возрастать.

Именно на таких генетических процессах биологических организмов основаны генетические алгоритмы, предназначенные для решения задач функциональной оптимизации. Эти задачи относятся к наиболее распространенному и важному для практики классу задач. Каждому из нас не раз приходилось решать задачи подобного рода и на работе, и в повседневной жизни либо для оптимального распределения своего времени между разнообразными делами, либо для достижения максимальной скорости или производительности определённого процесса.

Генетический алгоритм (ГА) представляет собой эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. [4]

Например, ГА могут использоваться, чтобы проектировать структуры моста, для поиска максимального отношения прочности/веса, или определять наименее расточительное размещение для нарезки форм из ткани. Они могут также использоваться для интерактивного управления процессом, например на химическом заводе, или балансировании загрузки на многопроцессорном компьютере. [5]

Постановка задачи

В данной статье будем решать задачу поиска кратчайшего пути для информационного пакета (сообщения) в компьютерной сети, состоящей из 10 узлов, который можно представить в виде матрицы, изображённой на рис.1.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	14	5	26	13	15	26	30	27	5
14	0	23	26	9	20	21	15	8	20
5	23	0	2	11	25	14	20	12	20
26	26	2	0	23	25	3	23	22	13
13	9	11	23	0	12	24	1000	24	25
15	20	25	25	12	0	26	5	24	9
26	21	14	3	24	26	0	15	17	22
30	15	20	23	1000	5	15	0	9	25
27	8	12	22	24	24	17	9	0	16
5	20	20	13	25	9	22	25	16	0

Рис.1. Матрица с весами путей (дуг) между узлами сети

Матрица, представленная на рис.1, является симметричной, то есть, например, путь из первого узла ко второму имеет вес, равный весу пути из второго узла в первый. Для упрощения расчетов граф сети лучше сделать полносвязанным (каждая пара вершина имеет связь). При этом для обозначения отсутствия связи выбраны очень большие значения веса дуги (1000), а для петли (дуги, соединяющей вершину с самой собой) – значение веса равное 0.

Решать данную задачу будем с помощью генетических алгоритмов. Рассмотрим систему для решения подобных задач, разработанную в Web-среде (PHP).

Ниже будут рассмотрены общие принципы работы генетических алгоритмов, различные способы организации алгоритмов селекции, скрещивания, мутации и редукции, сравнение эффективности и результатов работы некоторых из них на примере программы, реализованной на языке PHP.

Общие принципы генетических алгоритмов

При решении задачи оптимизации с помощью генетического алгоритма переменные, характеризующие решение, представлены в виде ген в хромосоме. Каждая хромосома представляет собой один из вариантов решения задачи. Набор хромосом называется популяцией. Процессом решения задачи является переход от одного поколения к другому. Причём поколение – это переход от одной популяции к другой путём различных модификаций. Новые решения позиционируются в популяции в соответствии с их положением на поверхности целевой функции.

ГА состоит из шагов, изображённых схемой на рис.2.

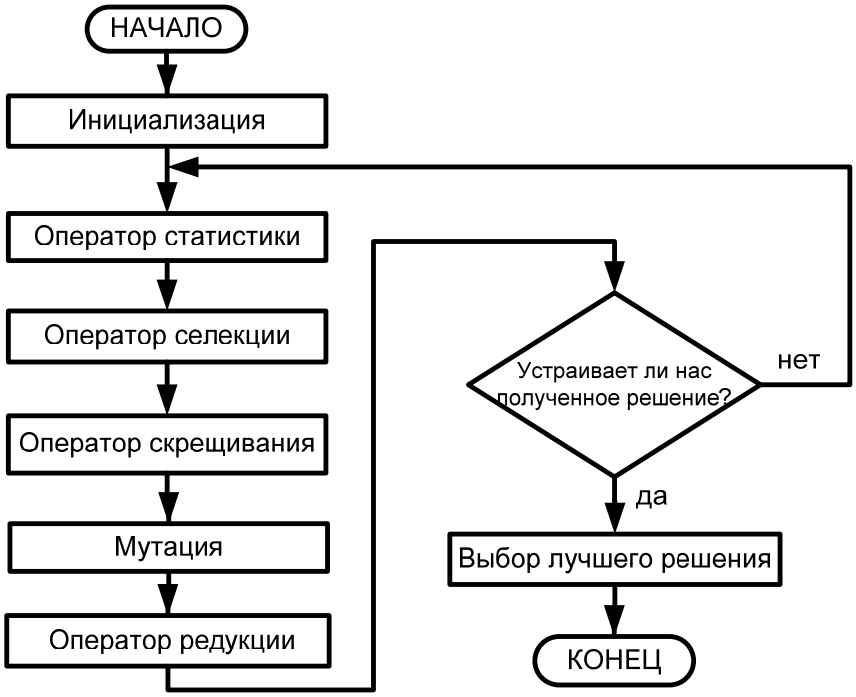

Рис.2. Основные шаги ГА

Рассмотрим каждый шаг ГА в отдельности.

Инициализация, т.е. формирование исходной популяции, заключается в случайном выборе заданного количества хромосом (особей), представляемых двоичными последовательностями фиксированной длины или псевдокодами.

Оператор статистика собирает сведения о текущей популяции, выделяет максимальное, минимальное и среднее значения целевой функции. По полученным значениям можно судить о результатах работы ГА и эффективности того или иного способа реализации его операторов.

Оператор селекция выбирает те хромосомы, которые будут участвовать в создании потомков для следующей популяции, т.е. для очередного поколения. Такой выбор производится согласно принципу естественного отбора, по которому наибольшие шансы на

участие в создании новых особей имеют хромосомы с наилучшими значениями целевой функции.

В **операторе скрещивание** выбранные хромосомы обмениваются своими частями, в результате чего может быть получен новый набор хромосом – потомки.

Мутация – появление дополнительной информации как в родителе, так и в потомке. Разделяют два вида мутаций

- случайная (случайное изменение генов на новые значения в одном или нескольких местах);
- направленная (делается направлено для решения задачи). [1]

Оператор редукция уничтожает особи с наихудшим качеством.

Реализация решения поставленной задачи с помощью ГА

В качестве примера решения поставленной в начале статьи задачи рассмотрим следующую программу.

Сначала пользователю предоставляется возможность выбрать, эффективность алгоритмов каких операторов он хочет сравнить. При запуске программы на экране появляется предложение выбрать один из трёх алгоритмов:

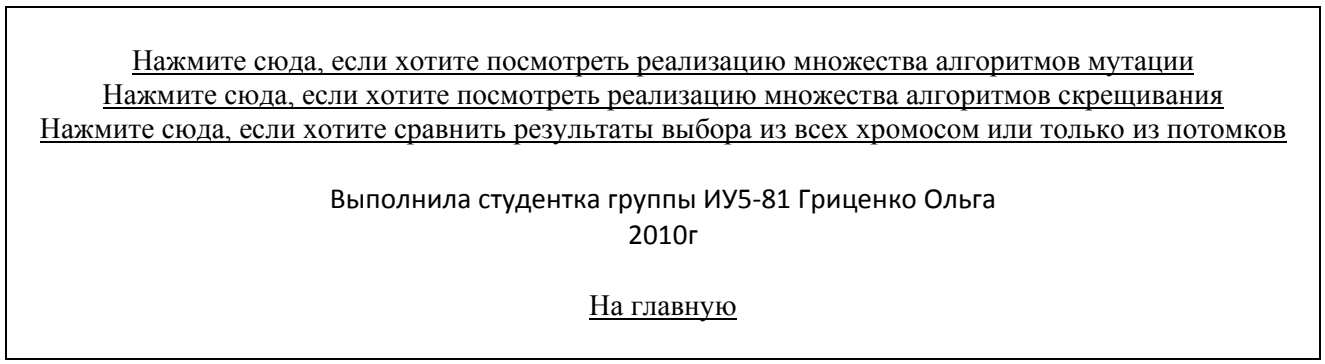

Рис. 3. Главное меню программы

Далее пользователю предоставляется возможность выбрать начальный и конечный узлы сети, между которыми необходимо найти кратчайший путь. Общее количество узлов в сети фиксировано по условию задачи и равно 10.

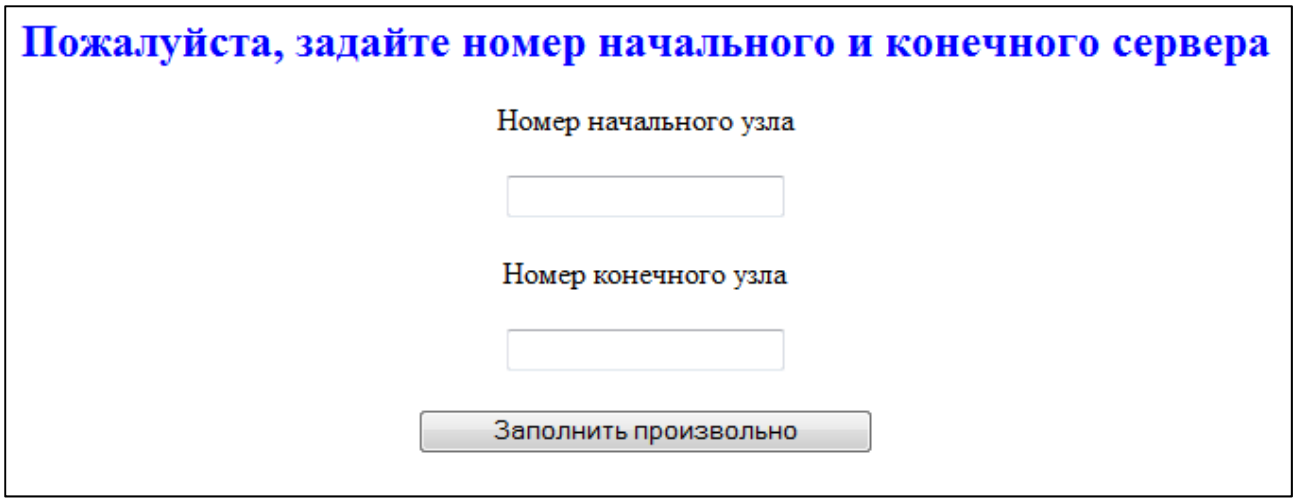

Рис. 4. Выбор начального и конечного узлов сети

После этого матрица весов заполняется произвольно с учётом симметричности и того, что для петли значение веса равно 0. Таким образом, сначала заполняется произвольными значениями часть матрицы, расположенная выше диагонали. Затем эти значения отражаются на оставшуюся половину. После этого выбираются произвольные значения весов (которые не являются петлями) и им присваивается значение 1000. Это значит, что не существует пути между этими двумя вершинами.

```
for($h=0; $h<2; $h++)
{
    $a = rand(0,9);
    $b = rand(0,9);
    if(!($a==$b))  // проверка на то, что это не петля
    {
        $cell[$a][$b] = 1000;
    }
}
```

Как видно из приведённого участка кода, можно дважды попасть на одну и ту же дугу графа, что значит, что только между двумя его вершинами не будет пути. То, что мы присваиваем 1000 таким дугам позволяет нам облегчить реализацию алгоритма, так как граф остаётся полносвязным, но при этом хромосомы с такими значениями весов быстро отсеются.

Далее составляется первая популяция. Тут следует заметить одно допущение – в качестве генов хромосомы используются числа в десятичной системе счисления. Каждое из этих чисел означает номер вершины, через которую сообщение проходит, если идёт по данному пути. Первому и последнему гену присваиваются значения, введённые пользователем. При этом первоначально гены в хромосомах не повторяются, хотя и выбираются произвольно.

Далее из полученных хромосом выбираются те, которые будут участвовать в кроссинговере. Для этого вычисляется значение целевой функции для каждой хромосомы как сумма весов всех путей, встречающихся на пути от исходного к конечному узлу, что иллюстрируется следующим фрагментом кода

```
for($i=0; $i<$num; $i++)
{
    for($j=0; $j<10; $j++)
    {
         // определение номера первой вершины
         $y = $chrom[$i][$j];
         // определение номера второй вершины
         $x = $chrom[$i][$j+1];
         // определение веса пути из первой вершины во вторую
         // с помощью таблицы cell
         $f[$i] = $f[$i] + $cell[$y-1][$x-1];
    }
}
```

Здесь
i – номер хромосомы,
j – номер гена в i-ой хромосоме,
f – качество i-ой хромосомы,
chrom – массив хромосом,
cell – таблица со значениями весов.

Затем определяется вероятность участия хромосомы в кроссинговере по следующей формуле

$$Pi = \left(1 - \frac{fi}{\sum_0^{n-1} fi}\right) * 100\%, \text{ где}$$

fi – качество i-ой хромосомы;
n – размер популяции;
i – номер хромосомы.

Для решения поставленной задачи нам необходимо найти минимальный путь, поэтому мы полученное значение вычитаем из единицы.

Если полученное значение вероятности превышает 90%, то в конец массива chrom дописывается качество данной хромосомы, то есть она помечается как хромосома, которая может принимать участие в кроссинговере.

Далее происходит выбор пар хромосом и скрещивание этих пар.

Замечание. При скрещивании следует учитывать, что первый и последний гены (начальный и конечный узлы) должны оставаться без изменений.

После скрещивания формируется новая популяция (она может состоять как из всех хромосом, так и только из потомков). Затем осуществляются мутация и редукция.

Оператор редукции определяет качество для каждой хромосомы, определяет 10 наилучших результатов и оставляет хромосомы, качество которых равно одному из выбранных значений. Остальные хромосомы отбрасываются. В результате у нас получится 10 или чуть больше хромосом (так как могут встретиться хромосомы, качество которых одинаково).

```
function reduction(&$chrom, $cell)
{
    // определяем количество хромосом, которое получилось
    //после скрещивания

    $num = count($chrom);

// определение качества каждой хромосомы

    for($i=0; $i<$num; $i++)
    {
        for($j=0; $j<10; $j++)
        {
            $y = $chrom[$i][$j];
            $x = $chrom[$i][$j+1];
            $f[$i] = $f[$i] + $cell[$y-1][$x-1];
        }
        $ff[$i] = $f[$i];
    }
    sort($ff);      // сортировка массива для определения
    array_splice($ff,10);    //лучших 10 значений
    $i = 0;

// оставляем только те хромосомы, качество которых входит в
//десять лучших значений

    while($chrom[$i])
    {
        if($f[$i]!=$ff[0])
        {

// если качество хромосомы не входит в десять лучших значений
            if(!(array_search($f[$i], $ff)))
            {
            // то удаляем эту хромосому их массива chrom
                array_splice($chrom,$i,1);
                array_splice($f,$i,1);
            }
            else
            {
            // иначе переходим к следующей хромосоме
                $i++;
            }
        }
        else
        {
            $i++;
        }
    }
```

Различные алгоритмы перечисленных алгоритмов подробно будут описаны ниже, поэтому сейчас мы не будем на них останавливаться.

Виды алгоритмов скрещивания

Существует большое количество способов, которыми можно организовать скрещивание хромосом. Самый простой из них - **случайный выбор родительской пары.** При таком подходе оба родителя выбираются из всей популяции случайным образом, причём каждая особь может участвовать в скрещивании несколько раз. К преимуществам данного подхода можно отнести его простоту и универсальность при решении любого рода задач. Однако этот способ обладает значительным недостатком, который заключается в зависимости от численности популяции – с ростом числа особей снижается эффективность алгоритма, реализующего данный подход.

Второй способ реализации алгоритма скрещивания называется **селективным**. При данном подходе для создания родительской пары выбираются только те хромосомы, качество которых не меньше определённого значения. К преимуществам данного способа относится более быстрая сходимость реализующего его алгоритма. Его недостатком является невозможность применения в тех задачах, где ставится задача определения нескольких экстремумов, поскольку для таких задач алгоритм, как правило, быстро сходится к одному из решений. [3] Чтобы исправить этот недостаток, можно использовать подходящий механизм отбора, который будет замедлять слишком быструю сходимость алгоритма.

В селективном наборе можно выделить ещё две разновидности алгоритмов скрещивания:

1. **одна особь может участвовать в кроссинговере несколько раз;**
2. **каждая особь может участвовать в кроссинговере только один раз.**

Процедура, осуществляющая селективное скрещивание, при котором одна особь может участвовать в кроссинговере несколько раз, имеет такой вид:

```
function cross_rand($chrom, &$children, $how)
{
     echo "<H2><FONT COLOR = BLUE>Скрещивание</FONT></H2><BR>";

     $num = count($chrom);    // определяем количество хромосом
     $v=0;
     for($i=0;$i<$num;$i++)
     {

     // если хромосома выбрана для скрещивания (в массиве есть
     // 11-ый элемент)

          if($chrom[$i][10])
          {

// то переписываем эту хромосому во вспомогательный  массив, с
//которым будем работать оператор скрещивания

               for($j=0; $j<11; $j++)
               {
                    $newchrom[$v][$j] = $chrom[$i][$j];
```

```
                }
                $f[$i]=$chrom[$i][10];
                $v++;
            }
        }
        $newnum = count($newchrom);
        $amount = $newnum-1;
        $j=0;
        for($i=0; $i<4; $i++)
        {
            $first = 0;
            $second = 0;
            $many = 0;

// выбираем хромосомы для скрещивания

            while($first == $second AND $many<10)
            {
                $first = rand(0, $amount);
                $second = rand(0, $amount);
                $many++;
            }

        // если не удалось выбрать 2 различные хромосомы

            if($many==10)
            {
                $amount2 = $amount - 1;
                $first = rand(0, $amount2);
                $second = $first + 1;
            }

            // Копируем выбранные хромосомы для того, чтобы скрестить
копии

            for($k=0; $k<10; $k++)
            {
                $temp1[$k] = $newchrom[$first][$k];
                $temp2[$k] = $newchrom[$second][$k];
            }

            // Выбираем способ скрещивания

            if($how==1)    // одноточечное скрещивание
            {
                crosspair1point($temp1, $temp2);
            }
            if($how==2)    // двухточечное скрещивание
            {
                crosspair2points($temp1, $temp2);
            }

            // Сохраняем потомков в отдельный массив
```

```
            for($k=0; $k<10; $k++)
            {
                 $children[$j][$k] = $temp1[$k];
                 $children[$j+1][$k] = $temp2[$k];
            }
            $j = $j + 2;

            // Очищаем вспомогательные массивы

            unset($temp1);
            unset($temp2);
      }
unset($first);
unset($second);
unset($num);
unset($amount);
}
```

Процедура, осуществляющая селективное скрещивание двух случайно выбранных хромосом, при котором каждая особь может участвовать в кроссинговере только один раз, отличается следующим фрагментом кода:

```
      // Удаляем из первоначального массива хромосомы, которые уже скрещивались, чтобы каждая скрещивалась только один раз

                array_splice($newchrom, $first, 1);
                array_splice($newchrom, $second, 1);

      // Уменьшаем количество хромосом на 2

                $amount = $amount - 2;
```

В данных процедурах

newchrom – копия первоначального массива хромосом, которую мы делаем, чтобы можно было работать с этими хромосомами, не портя исходный набор хромосом;

children – массив, состоящий из потомков.

Как можно было заметить из комментариев к двум представленным выше процедурам, алгоритмы скрещивания можно также разделить на

1. одноточечные (представлено на рис.5)

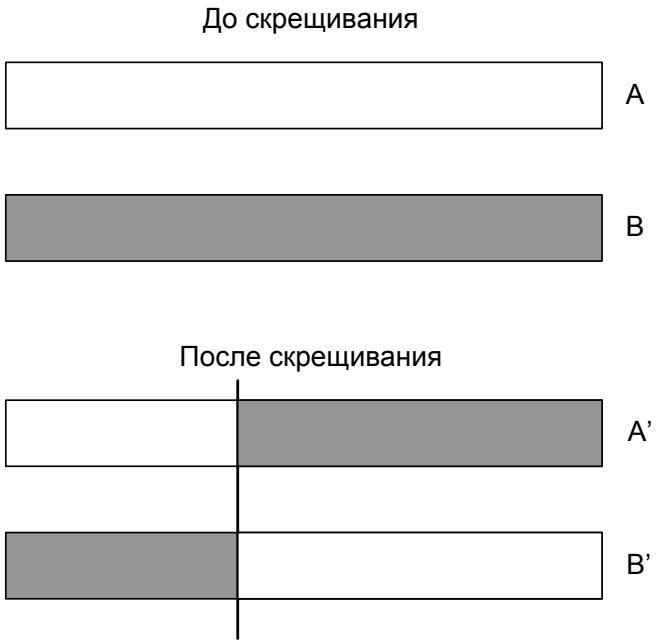

Рис.5. Пример одноточечного скрещивания

2. многоточечные (представлено на рис.6)

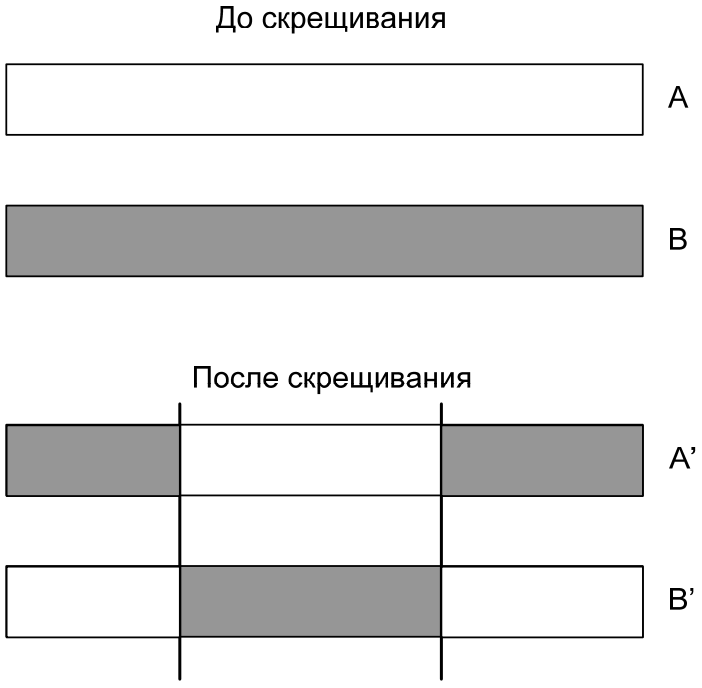

Рис.6. Пример двухточечного скрещивания

Процедура, которая осуществляет одноточечное скрещивание хромосом, имеет такой вид:

```
function crosspair1point(&$part1, &$part2)
{
     $point = 0;

// выбираем произвольную точку скрещивания

     $point = rand(2,8);

     // записываем во вспомогательные массивы гены исходных
//массивов, расположенные до точки скрещивания

     for($t=1,$h=0; $t<$point; $t++,$h++)
     {
          $p11[$h] = $part1[$t];
          $p21[$h] = $part2[$t];
     }
// записываем во вспомогательные массивы гены исходных //массивов,
расположенные после точки скрещивания

     for($t=$point,$h=0; $t<9; $t++,$h++)
     {
          $p12[$h] = $part1[$t];
          $p22[$h] = $part2[$t];
     }
// запоминаем начальную и конечную вершины, так как они не должны
//меняться

     $f[0]=$part1[0];
     $s[0]=$part1[9];

// соединяем вспомогательные массивы так, как показано на рис.4,
//получая две новые хромосомы

     $part1 = array_merge($f, $p21, $p12, $s);
     $part2 = array_merge($f, $p11, $p22, $s);
}
```

Аналогично осуществляется двухточечное скрещивание. В этом случае необходимо выбрать две несовпадающие точки скрещивания.

Существует ещё два способа формирования родительской пары - **инбридинг** и **аутбридинг**. Оба эти метода основаны на понятии схожести особей. Под инбридингом понимается такой метод, когда первый родитель выбирается случайно, а вторым с большей вероятностью будет максимально близкая к нему особь. Аутбридинг же, наоборот, формирует брачные пары из максимально далеких особей. К преимуществам данных подходов можно отнести возможность их применения для решения многоэкстремальных задач. Однако их влияние на поведение генетического алгоритма не одинаково. Инбридинг

характеризуется тем, что при данном способе поиск концентрируется в локальных узлах. Это приводит к разбиению популяции на отдельные локальные группы вокруг подозрительных на экстремум участков. Аутбридинг предупреждает сходимость алгоритма к уже найденным решениям, работая с новыми, неисследованными областями.

В рассматриваемой программе сравниваются четыре алгоритма мутации:

1. **Скрещиваются две случайно выбранные особи поколения, и одна особь может скрещиваться несколько раз. При этом скрещивание одноточечное.**
2. **Скрещиваются две случайно выбранные особи поколения, и одна особь может скрещиваться несколько раз. При этом скрещивание двухточечное.**
3. **Скрещиваются две случайно выбранные особи поколения, и одна особь может скрещиваться только один раз. При этом скрещивание одноточечное.**
4. **Скрещиваются две случайно выбранные особи поколения, и одна особь может скрещиваться только один раз. При этом скрещивание двухточечное.**

Показатели, характеризующие эффективность алгоритмов представлены на рис.7:

Результаты

	Каждая особь может скрещиваться несколько раз						**Каждая особь может скрещиваться только один раз раз**					
	Одноточечное скрещивание			**Двухточечное скрещивание**			**Одноточечное скрещивание**			**Двухточечное скрещивание**		
Номер испытания	**Max**	**Avr**	**Min**	**Max**	**Avr**	**Min**	**Max**	**Avr**	**Min**	**Max**	**Avr**	**Min**
до начала испытаний	161	128.5	96	161	128.5	96	161	128.5	96	161	128.5	96
1	112	92.5	73	113	95	77	125	95	65	122	109	96
2	99	90	81	109	95.5	82	114	89.5	65	118	102.5	87
3	83	58.5	34	90	72	54	102	81.5	61	108	92.5	77

График сравнения результатов

На главную

Рис.7. Результаты работы разных алгоритмов скрещивания

Нажимаем на «График сравнения результатов» и получаем

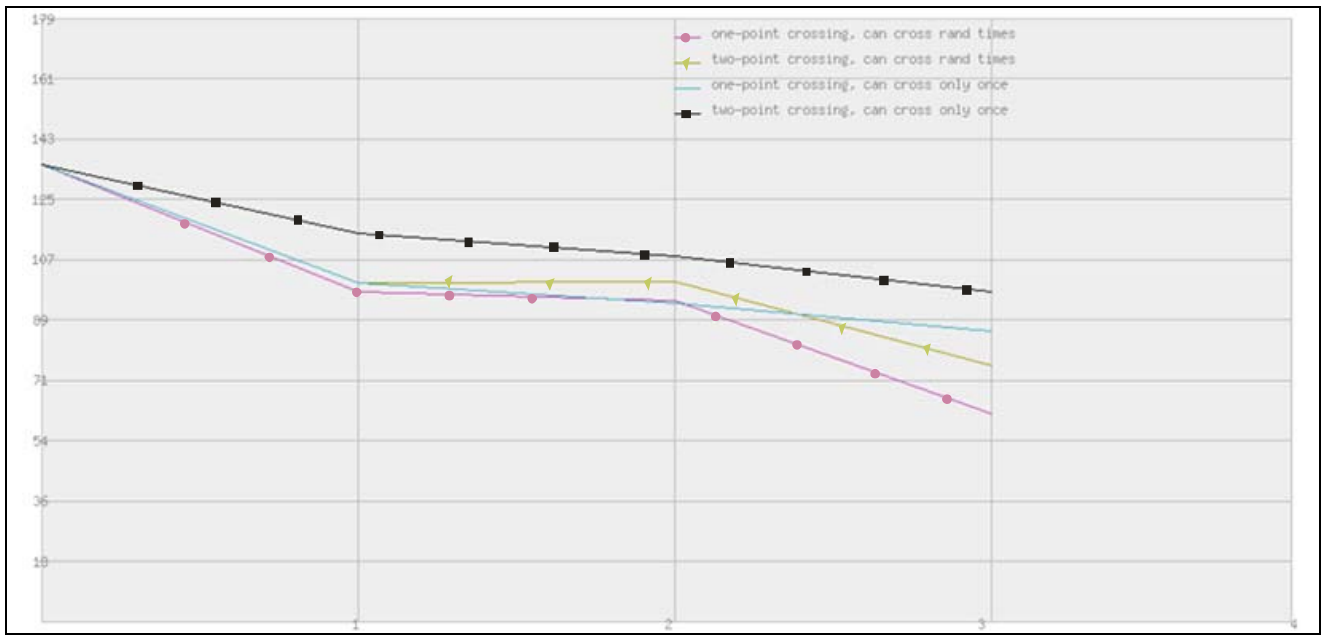

Рис. 8. График сравнения результатов работы различных алгоритмов скрещивания

На графике, представленном на рис.8, показаны изменения средних значений качества хромосом для четырёх различных алгоритмов скрещивания по итерациям. И даже при 4 итерациях видно, что скрещивание, при котором каждая хромосома может участвовать в кроссинговере несколько раз, даёт наименьшие средние показатели, то есть лучшее решение. Тот же вывод следует и из таблицы на рис.7, где представлены не только средние, но и максимальные с минимальными значениями качества хромосом для каждого алгоритма на каждой итерации. Это происходит потому, что хромосомы с лучшими показателями могут участвовать в скрещивании несколько раз, а значит, показатели потомков тоже будут лучше. При этом лучшие показатели у одноточечного скрещивания.

Виды алгоритмов мутации

С помощью мутации создаются такие цепочки генов, которые не входили в первое поколение, но были необходимы для получения правильного решения. В тоже время она создает бесполезные и «вредные» цепочки, что затрудняет поиск решения. Необходимо правильно пообобрать настройки мутации, чтобы она и помогала и мало мешала поиску решения. [2]

Выделяют несколько видов алгоритмов мутаций:

1. Инверсия гена;
2. Замена гена на случайный;
3. Замена гена на соседний.

В начале каждой из трёх процедур, реализующих перечисленные алгоритмы, хромосомам присваиваются произвольные вероятности мутации. И только если эта вероятность больше 80%, хромосома мутирует. Рассмотрим отдельно каждый алгоритм мутации.

1. Инверсия гена

При инверсии гена выбирается случайный ген с первого по восьмой, так как нулевой и девятый гены – начальный и конечный узлы сети, которые мутировать не должны. При этом значение выбранной вершины не должно равняться 10, так как 10-10=0, а путь между двумя разными вершинами не может быть равным 0.

Процедура, которая осуществляет инверсию гена, имеет такой вид:

```
function mutation_inversion(&$chrom, $num)
{
     $possible = 80;     // с такой вероятностью хромосома мутирует

     // Определим вероятность мутации каждой хромосомы

     for($i=0; $i<$num; $i++)
     {
          $p[$i] = rand(1,1000001)/10000;
     }

     for($i=0; $i<$num; $i++)
     {
          $q=$i+1;

          // сравниваем вероятность мутации i-ой хромосомы с
//заданным значением

          if($p[$i]>=$possible)
          {
               // Инверсия гена
               $point = rand(1,8);
               // выбираем ген, значение которого не равно 10
               while($chrom[$i][$point]==10)
               {
                    $point = rand(1,8); // ген, который мутирует
               }
               $u = $point + 1;
               $chrom[$i][$point] = 10 - $chrom[$i][$point];
               $message .= "<br>Инверсия $u гена в $q хромосоме";
          }
     }
unset($p);
unset($point);
unset($u);
unset($q);
     return $message;
}
```

2. Замена гена на случайный

При замене гена на случайный две случайно выбранные вершины обмениваются своими значениями. Процедура, которая осуществляет замену гена на случайный, имеет такой вид:

```
function mutation_change_random(&$chrom, $num)
{
     $possible = 80;     // с такой вероятностью хромосома мутирует

     // Определим вероятность мутации каждой хромосомы

     for($i=0; $i<$num; $i++)
     {
          $p[$i] = rand(1,1000001)/10000;
     }

     for($i=0; $i<$num; $i++)
     {
          $q=$i+1;

          // сравниваем вероятность мутации i-ой хромосомы с
//заданным значением

          if($p[$i]>=$possible)
          {
               // Замена гена на случайный

               $point = rand(1,8); // ген, который мутирует
               $u = $point + 1;
               $newpoint = rand(0,9); //ген, на который заменяется
               $newu = $newpoint + 1;
               $chrom[$i][$point] = $chrom[$i][$newpoint];
               $message .= "<br>Мутация $u гена в $q хромосоме на
$newu ген";
          }
     }
unset($p);
unset($point);
unset($newpoint);
unset($newu);
unset($u);
unset($q);

     return $message;

}
```

3. Замена гена на соседний

При замене гена на соседний произвольно выбранная вершина обменивается своим значением со следующей за ней вершиной. Таким образом, две соседние вершины получают одинаковые значения, а значит расстояние между ними равно 0. Это способствует более быстрой сходимости алгоритма.

Процедура, которая осуществляет замену гена на случайный, имеет такой вид:

```
function mutation_change_next(&$chrom, $num)
{
     $possible = 80;     // с такой вероятностью хромосома мутирует

     // Определим вероятность мутации каждой хромосомы

     for($i=0; $i<$num; $i++)
     {
          $p[$i] = rand(1,1000001)/10000;
     }

     for($i=0; $i<$num; $i++)
     {
          $q=$i+1;

          // сравниваем вероятность мутации i-ой хромосомы с
//заданным значением

          if($p[$i]>=$possible)
          {
               // Замена гена на соседний

               $point = rand(1,8);
               $u = $point+1;
               $chrom[$i][$point] = $chrom[$i][$point+1];
               $message .= "<br>Мутация $u гена в $q хромосоме";
          }
     }
unset($p);
unset($point);
unset($u);
unset($q);

     return $message;

}
```

Сравним результаты работы этих трёх алгоритмов. Показатели, характеризующие эффективность алгоритмов представлены на рис.9:

Результаты

	Мутация - инверсия вершины			Мутация вершины - замена на случайную			Мутация вершины - замена на соседнюю		
Номер испытания	**Максимальное значение**	**Среднее значение**	**Минимальное значение**	**Максимальное значение**	**Среднее значение**	**Минимальное значение**	**Максимальное значение**	**Среднее значение**	**Минимальное значение**
до начала испытаний	170	140.5	111	170	140.5	111	170	140.5	111
1	132	113.5	95	127	111	95	126	110.5	95
2	119	96.5	74	125	105	85	110	85.5	61
3	95	83	71	100	85.5	71	78	65	52

График сравнения результатов

На главную

Рис.9. Результаты работы разных алгоритмов мутаций

Нажимаем на «График сравнения результатов» и получаем:

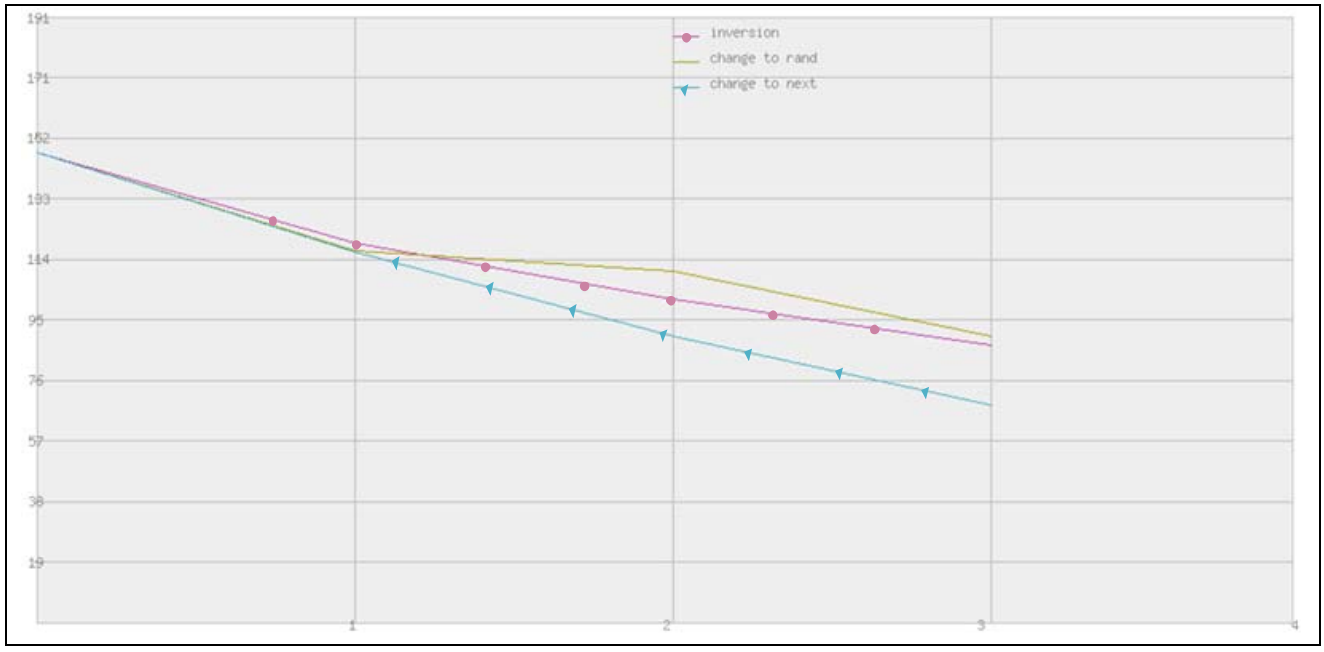

Рис. 10. График сравнения результатов работы различных алгоритмов мутаций

На графике, представленном на рис.10, показаны изменения средних значений качества хромосом для трёх различных алгоритмов мутаций по итерациям. И даже при 4 итерациях видно, что мутация с заменой гена на соседний, как и предполагалось, даёт наименьшие средние показатели, то есть лучшее решение. Тот же вывод следует и из таблицы на рис.9, где представлены не только средние, но и максимальные с минимальными значениями качества хромосом для каждого алгоритма на каждой итерации.

Механизм отбора

Существует несколько механизмов отбора хромосом, наиболее эффективными из которых являются **элитный** и **отбор с вытеснением**. При элитном отборе новая популяция содержит только лучшие особи как из родителей, так и из потомков. Недостатком данного способа может являться слишком быстрая сходимость. Однако алгоритм, реализованный на сочетании элитного отбора с определёнными методами скрещивания, например, с аутбридингом, является достаточно эффективным и широко применяемым.

При втором методе на результат отбора влияет не столько степень приспособленности хромосомы, сколько наличие в формируемой популяции особи с аналогичным хромосомным набором. А уже из всех особей с одинаковыми генотипами выбираются те, чья приспособленность выше. Использование данного механизма отбора позволяет достигнуть две цели, а именно: сохранение в популяции лучших решений и поддержание достаточного генетического разнообразия. Данный метод эффективен для решения многоэкстремальных задач.

Существует ещё один вариант отбора – когда в новую популяцию входят только потомки, получившиеся при скрещивании. Но такой метод не очень эффективен, так как в результате скрещивания мы можем получить хромосомы с худшим качеством, чем было у родителей. То есть данный подход не всегда обеспечивает сходимость алгоритма, реализованного на его основе.

В рассматриваемой программе сравниваются два варианта отбора:

1. **отбор идёт из всех хромосом (потомки+родители);**
2. **отбор идёт только из потомков.**

Сравним результаты работы этих трёх алгоритмов. Показатели, характеризующие эффективность алгоритмов представлены на рис.11:

Результаты

	Выбор лучших хромосом из всех возможных			Выбор лучших хромосом только из потомков		
Номер испытания	**Максимальное значение**	**Среднее значение**	**Минимальное значение**	**Максимальное значение**	**Среднее значение**	**Минимальное значение**
до начала испытаний	1167	626.5	86	1167	626.5	86
1	142	114	86	205	156	107
2	134	102.5	71	149	113.5	78
3	89	65.5	42	118	89.5	61

График сравнения результатов

На главную

Рис.11. Результаты работы разных методов отбора

Нажимаем на «График сравнения результатов» и получаем

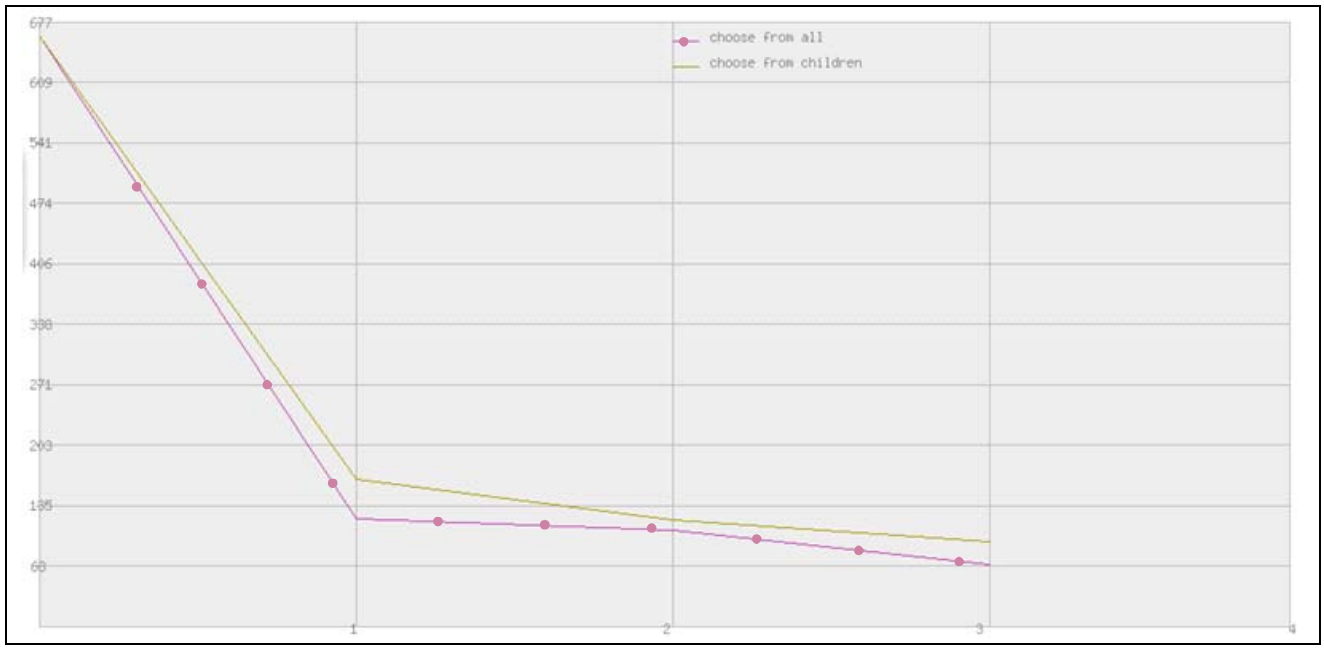

Рис. 12. График сравнения результатов работы различных методов отбора

На графике, представленном на рис.12, показаны изменения средних значений качества хромосом для двух различных методов отбора по итерациям. В данном примере

видно, что метод отбора из всех хромосом, как мы и предполагали, даёт лучшее решение. Тот же вывод следует и из таблицы на рис.11, где представлены не только средние, но и максимальные с минимальными значениями качества хромосом для каждого алгоритма на каждой итерации.

Список использованной литературы

1. Лекции по дисциплине «Интеллектуальные системы» за 2010 год;
2. «Пример реализации и исследование эффективности ГА», Филиппович А.Ю., Серебряков А.М.;
3. Т.В.Панченко Генетические алгоритмы, «Астраханский университет», 2007;
4. http://ru.wikipedia.org;
5. www.algolist.manual.ru.