Филиппович А.Ю., Серебряков А.М.

ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ. ПРИМЕР РЕАЛИЗАЦИИ И ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ

1. Постановка задачи.

В рамках проекта «Автоматизированная система научных исследований динамики ассоциативно-вербальной модели языкового сознания русских как индикатора образа России в новейшей истории и современности» при поддержке гранта РГНФ № 06-04-03803в создается подсистема «Анализа ассоциативно-вербальной сети (АВС)». В нее входит модуль «Поиска ассоциативных цепочек», который предназначен для исследования связей и поиска путей между ассоциациями с помощью различных методов. АВС представляет собой взвешенный граф, вершинами которого являются слова естественного языка (стимулы и реакции), дугами –различные связи, которые возникают между ними, а весами дуг - частота проявления связи. На первом этапе исследования топологии сети можно отказаться от анализа типов связей и их направленности, упростив тем самым математические и программные методы. Для простоты расчетов также будем обозначать конкретные стимулы и реакции с помощью числовых идентификаторов.

В данной статье будем рассматривать задачу поиска кратчайшего пути между двумя вершинами АВС. Например, для графа на **Рис 1.1.** кратчайшим путем из вершины 1 в вершину 5 является путь через вершину 4. Рядом с дугами графа указаны стоимости прохождения по этой дуге.

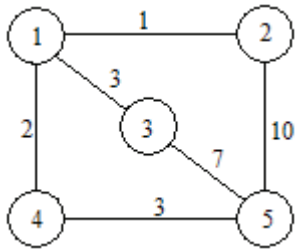

Рис 1.1. Взвешенный ненаправленный граф.

Решать данную задачу будем с помощью *генетических алгоритмов*. Рассмотрим систему для решения подобных задач, разработанную в среде Delphi (**Рис 1.2.**)

Ниже будут рассмотрены примеры на языке Object Pascal, с помощью которых можно будет реализовать систему, использующие генетические алгоритмы. Будут проведены исследования по оптимизации ГА, насколько зависит эффективность ГА от той или иной реализации *генетических операторов*, от разных значений вероятностей из применения. Так же рассмотрим насколько эффективно решение данной задачи с помощью ГА при тех или иных начальных условиях. Но сначала опишем ограничения, которые накладываются на данную задачу рассматриваемой системой.

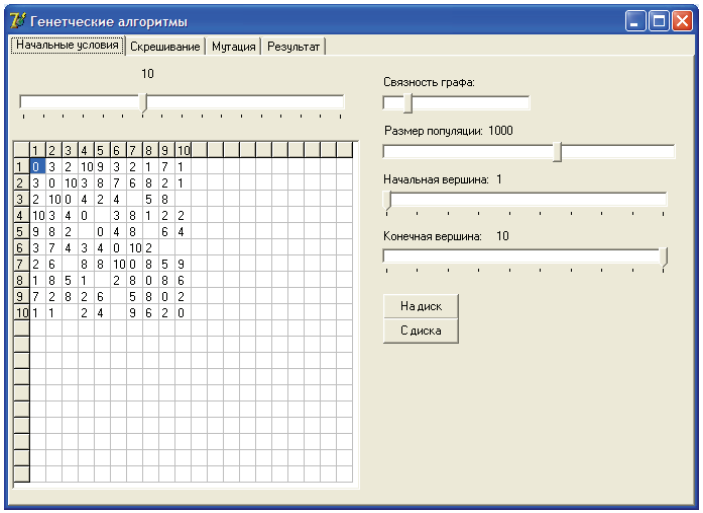

Рис 1.2. Общий вид окна системы.

Количество вершин графа может варьироваться от четырех до двадцати. Размер *популяции* может меняться от одной особи до ста тысяч. Поиск может производиться между двумя любыми вершинами. Длина хромосомы постоянна, количество генов в ней меньше количества вершин в графе на два, т.к. вершины, между которыми ищется кратчайший путь, не меняются во время поиска. Гены в хромосоме имеют десятичный формат для более простого понимания кода программы, и каждый

из них представляет собой номер вершины. Длины дуг (или стоимость прохождения по дуге) могут принимать значения от 1 до 10, при этом простой на вершине не стоит ничего. Между двумя вершинами может вообще не быть пути, условимся тогда что в этом случае расстояние между вершинами равно 1000001. Такое большое число гарантирует что хромосома, имеющая в своем составе такой путь, будет отсеяна во время *селекции*. В рассматриваемой системе мы может регулировать *связность графа*, т.е. процент «бесконечных вершин».

В данной задаче генетический алгоритм будет стараться минимизировать *целевую функцию*. Целевая функция в данной задаче – это сумма длин всех дуг пути, представляемого хромосомой.

2. Скрещивание.

В данной системе количество *точек скрещивания* можно варьировать. См. **Рис 2.1.**

Рис 2.1. Многоточечное скрещивание.

Процедура, которая осуществляет скрещивание двух хромосом, может иметь такой вид:

```
type Hromosom = record
      Gens: array of byte;
      f: real;
      end;

type MatrType = array of array of real;
```

```
var
  Find1, Find2: integer;
  Matrix: MatrType;

procedure CroosPair(RandFlag: boolean; pointsCount,
                      len: integer; h1,h2:hromosom;
                      var rez1,rez2:hromosom);
type rec=record
      num:integer;
      f:boolean;
     end;
var
 arr: array of rec;
 i, j, p, tlen: integer;
 flag: boolean;
 rand, points: integer;
begin
 tlen:=len-3;
 for i:=1 to (len-2) do
  begin
   arr[i].num:=i;
   arr[i].f:=false;
  end;
 if RandFlag then points:=random(pointscount)+1
  else points:=pointsCount;
 for i:=1 to points do
  begin
   p:=0;
   j:=1;
   rand:=(trunc(random(tlen))+1);
   while (j<=rand) do
    begin
     inc(p);
     while (arr[p].f) do inc(p);
     inc(j);
    end;
   dec(tlen);
   arr[p].f:=true;
  end;
 flag:=true;
 for i:=1 to (len-2) do
  begin
   if flag then
    begin
     rez1.Gens[i]:=h2.Gens[i];
     rez2.Gens[i]:=h1.Gens[i];
    end
   else
    begin
     rez1.Gens[i]:=h1.Gens[i];
```

```
      rez2.Gens[i]:=h2.Gens[i];
     end;
    if arr[i].f then
     if flag then flag:=false else flag:=true;
   end;
  rez1.f:=Matrix[Find1,rez1.Gens[1]];
  rez2.f:=Matrix[Find1,rez2.Gens[1]];
  for i:=1 to (len-3) do
   begin
    rez1.f:=rez1.f+Matrix[rez1.Gens[i],rez1.Gens[i+1]];
    rez2.f:=rez2.f+Matrix[rez2.Gens[i],rez2.Gens[i+1]];
   end;
    rez1.f:=rez1.f+Matrix[rez1.Gens[len-2],Find2];
    rez2.f:=rez2.f+Matrix[rez2.Gens[len-2],Find2];
end;
```

Данный листинг представляет собой процедуру для скрещивания двух хромосом в любом количестве точек. Первым параметром является флаг, который показывает будет ли выбрано количество точек скрещивания случайно или оно будет задано вторым параметром. Третий параметр – длина хромосомы. Следующие два - хромосомы, которые требуется скрестить, а последние два – результат скрещивания. Глобальные данные: Find1 и Find2 вершины путь между которыми ищется; Matrix – симметричная матрица чисел, которая является матрицей весов дуг графа, поиск в котором ведется.

Arr – это вспомогательный массив для записи мест где будет происходить скрещивание. Все места скрещивания выбираются случайным образом и распределены равномерно. После выбора точек скрещивания, происходит обмен генами между хромосомами и в конце рассчитываются новые значения целевых функции.

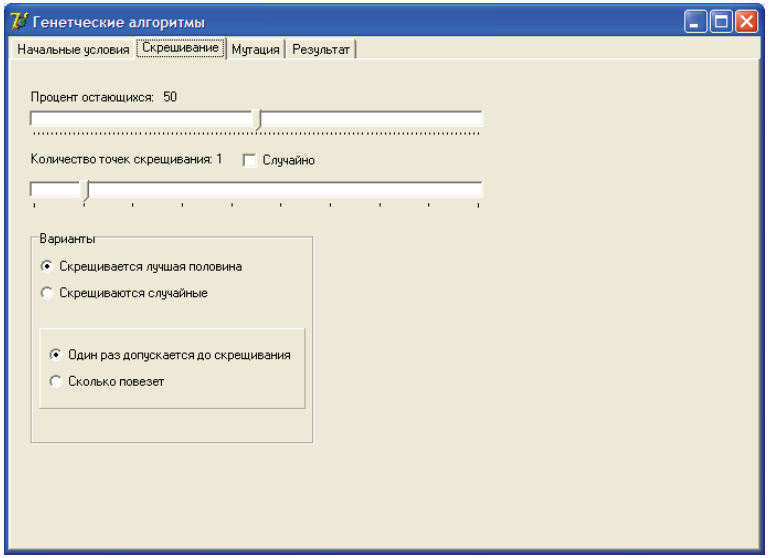

Рис. 2.2. Настройка параметров скрещивания.

В данной системе доступны четыре разных способа отбора особей для скрещивания:

1. Скрещиваются две случайно выбранные особи поколения, и одна особь может скрещиваться несколько раз.
2. Скрещиваются две случайно выбранные особи поколения, и одна особь может скрещиваться только один раз.
3. Скрещиваются две особи из лучшей половины поколения, и одна особь может скрещиваться несколько раз.
4. Скрещиваются две особи из лучшей половины поколения, и одна особь может скрещиваться только два раза.

Рассмотрим каждый из этих вариантов и попытаемся выявить наиболее подходящий. Для того чтобы в эксперимент не вмешивались посторонние факторы, не будем производить мутации и сохранение наиболее приспособленных особей из предыдущих поколений зададим равным 1%. Чтобы эксперимент смог завершиться, зададим достаточно большой размер популяции – 1000 особей, количество генов – 10, и сделаем граф полно-связным. Также не будем учитывать случаи попадания в *локальный экстремум*, т.к. они пока нас не интересуют, и потом, при добавлении других операторов, появляться не будут. Количество точек скрещивания будем задавать два раза: сначала равное единице, а затем

случайное. Это позволит определить влияет ли этот параметр на эффективность того или иного варианта. Оценивать эффективность того или иного метода будем по количеству новых поколений, которое придется создать для того, чтобы получить результат.

Первый вариант при одноточечном скрещивании фактически показал результат в 327 итерации, при случайном количестве точек скрещивания – 207, но при этом очень часто «проваливался» в локальный экстремум. Вариант 2, кроме того, замедлил примерно на 40%. процесс генерации нового поколения из-за контроля на отсутствие повторяемости, показал следующие результаты: 211 в первом и 190 во втором случаях. Третий вариант скрещивания дал результат в 62 итерации в первом случае и 51 во втором. Четвертый вариант оказался самым эффективным с результатом в 50 новых поколений с одноточечным скрещиванием, со случайным количеством точек скрещивания получилось в среднем 45 поколений.

Самым выгодным оказался вариант номер 4. Также эксперимент показал, что количество точек скрещивания качественно не влияет на выбор варианта в данных условиях.

Рассмотрим теперь эксперименты со случайным количеством точек скрещивания хромосом. Очень важной характеристикой процесса является то, что из-за случайности процесса решение часто не бывает найдено и попадает в локальный экстремум. Но если необходимая комбинация генов совпала, то решение находится быстрей из-за более сильной динамики процесса.

Теперь, при тех же начальных условиях эксперимента, выберем вариант 4, и будем варьировать количество точек скрещивания. В результате получим зависимость представленную на графике **Рис 2.3.**.

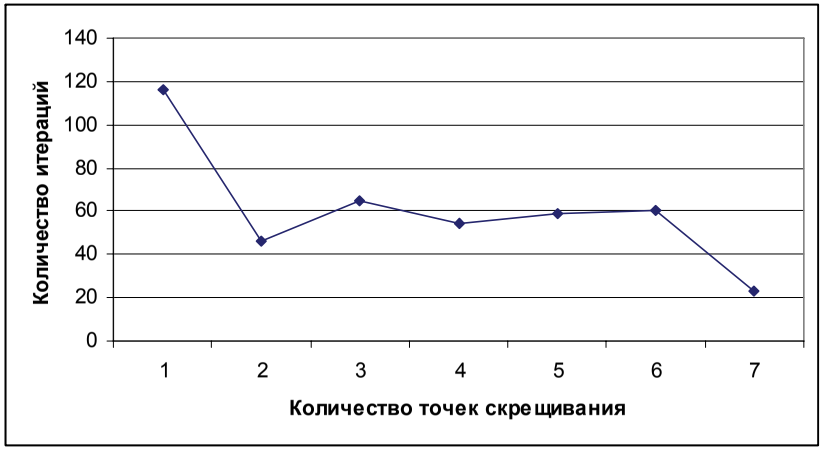

Рис 2.3. Зависимость количества итерации от количества точек скрещивания.

Данное исследование показало довольно интересные результаты. Во-первых, скрещивание работает более эффективно если количество точек скрещивания четное, т.е. хромосома, как минимум, сохранят свой первый и последний ген. Так же из графика видно, что чем больше количество точек скрещивания, то тем меньше количество итераций. Это происходит из-за того, что процесс приобретает все большую случайность. Про вариант с 7-ю точками стоит сказать отдельно. Для того чтобы добиться верного решения от него требуется упорство, т.к. процесс постоянно показывает более длинный путь, чем нужно. Но те редкие случаи, что были, дают возможность говорить о низком значении количества итераций. В данном случае происходящий процесс наименее похож на скрещивание, а носит чрезвычайно случайных характер, поэтому в случаях с меньшим количеством точек скрещивания «попадания» в локальный экстремум не наблюдалось.

По итогам этого раздела будем считать, что наилучшим вариантом для скрещивания является вариант 4, а количество точек скрещивания нужно принимать равным двум.

3. Мутация.

Мутация – второй по важности генетический оператор. 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. В рассматриваемой системе можно выбрать из четырех типов мутации:

1. Инверсия вершины;
2. Замена вершины на случайную;
3. Замена вершины на соседнюю;
4. Случайный способ из представленных выше.

Также задается вероятность, с которой к той или иной особи будет применена мутация. Внутри хромосомы мутирует случайная вершина.

Процедура мутации поколения может иметь такой вид:

```
type Population = array of Hromosom;

procedure Mutation(var ppl: population; amount, len,
  typem: integer; p: real);
var i, j, r: integer;
begin
  for i := 1 to amount do
    begin
      j := random(len - 2) + 1;
      if (random(1000001) / 10000) <= p then
        begin
          if typem = 4 then typem := random(3) + 1;
          if typem = 1 then
            ppl[i].Gens[j] := len + 1 - ppl[i].Gens[j]
          else if typem = 2 then
            ppl[i].Gens[j] := random(len) + 1
          else

            if j = 1 then
              ppl[i].Gens[j] := ppl[i].Gens[j + 1]
            else if j = (len - 2) then
              ppl[i].Gens[j] := ppl[i].Gens[j - 1]
            else
              begin
                r := random(2) + 1;
                if r = 1 then
                  ppl[i].Gens[j] := ppl[i].Gens[j + 1]
                else
```

```
                    ppl[i].Gens[j] := ppl[i].Gens[j - 1];
                end;
            ppl[i].f := Matrix[Find1, ppl[i].Gens[1]];
            for j := 1 to (len - 3) do
              ppl[i].f := ppl[i].f +
                Matrix[ppl[i].Gens[j], ppl[i].Gens[j +
                1]];
            ppl[i].f := ppl[i].f + Matrix[ppl[i].Gens[len
              - 2], Find2];
          end;
      end;
  end;
```

Ppl – популяция, в которой необходимо провести мутацию, amount – количество особей в популяции, len – количество генов в хромосоме, typem – тип мутации согласно списку выше, p – вероятность мутации хромосомы в процентах.

Попробуем оценить какая самая оптимальная вероятность мутации. Для этого необходимо так подобрать параметры задачи, чтобы при поиске решения популяция вырождалась к неверному решению, имеющему более длинный путь, чем возможно было бы. Для этого необходимо задать достаточно небольшой размер популяции, чтобы необходимые сочетания генов встречались реже. Выборку самых лучших особей, проводить не будем, т.к. она значительно сокращает время поиска решения.

Выберем полносвязанный граф с десятью вершинами, размер популяции - 100, количество генов в особи – 10. Будем оценивать вероятность мутации по количеству создаваемых поколений для получения решения и по количеству неверных решений, если для каждого исследования требовалось получить десять верных решений.

В результате получилась следующая зависимость:

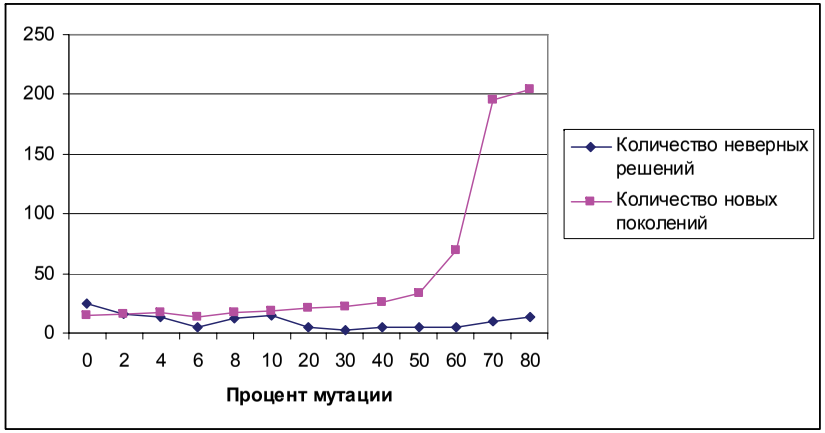

Рис. 3.1. Исследование вероятности мутации.

При данных начальных условиях получили результаты такие, что неизменно при увеличении вероятности мутации количество поколений, которые необходимо создать для получения решения неизменно растет. При вероятности мутации в 6% число неверных решений наиболее низкое. Значит, эта вероятность мутации оптимальна в данных условиях. При вероятности мутации в 30% и более количество неверных решений уменьшается, это происходит из-за того, что такая вероятность уже почти всегда обеспечивает необходимое сочетание генов, но время на получение решения уже сильно увеличивается.

Теперь рассмотрим эффективность того или иного способа мутации, из представленных в системе. При инверсии вершин, т.е. получении нового гена по номеру путем вычитания текущего номера из максимального, количество неверных решений при верных тридцати было равно 15. При мутации на случайную вершину – 10, такой результат лишь подчеркивает то, что мутация - процесс случайный. Чем более равномерное распределение, тем эффективнее мутация. При замене вершины в пути на следующую или предыдущую количество неверных решений получилось равным семи, значит, при выбрасывании вершины из пути, появляются правильные пути чаще. Но при уменьшении связности графа это преимущество нивелируется.

4. Элитизм.

Элитизм в генетических алгоритмах – выбор или приведение решение к заведомо лучшему результату. Элитизм «помогает» алгоритму быстрее прийти к верному решению. В рассматриваемой системе он представлен механизмом сохранения лучших особей из предыдущего поколения. Некоторое количество (задается процентным соотношением относительно общего числа особей в популяции) особей, целевая функция которых минимальна, входят в следующее поколение без изменений. Этот метод позволяет выделять наилучшие варианты путей. Проведем исследование - при каком проценте сохранения лучших путей решение находится максимально быстро (**Рис. 4.1.**):

Рис. 4.1. Зависимость быстроты поиска решения от количества оставшихся вершин.

По графику видно, что элитизм начинает действовать с 30% сохранения особей и затем его влияние не изменяется практически до конца. Начиная с 80% количества новых особей уже не хватает, и решение ищется долго. Также при выборе количества сохраняемых вершин, не-

обходимо учитывать, что чем больше вершин без изменения переходят из поколения в поколение, то тем больше неверных решений может возникнуть.