Лабораторная работа №4

Генетическое программирование

Общие сведения

В генетическом программировании (ГП) в качестве особи выступает программа, представленная в определенном формате, которая решает некоторую задачу. Часто это выполняется с использованием обучающих данных и индуктивного вывода. ГП очень близко к машинному обучению и поэтому в качестве фитнесс-функции как правило выступают функции ошибки. ГП работает с генетическим материалом переменной длины, что требует нестандартной формы представления генома и соответствующих генетических операторов.

Программы составляются из переменных, констант и функций, которые связаны некоторыми синтаксическими правилами. Поэтому определяется терминальное множество, содержащее константы и переменные, и функциональное множество, которое состоит, прежде всего, из операторов и необходимых элементарных функций (exp(x), sin(x) и т.п.). Следует отметить, что терминалы и функции играют различную роль. Терминалы обеспечивают входные значения в систему (программу), в то время как функции используются при обработке значений внутри системы. Термины «функции» и «терминалы» взяты из древовидного представления, и соответствуют узлам древовидных (или графоподобных) структур.

Терминальное множество

Терминальное множество включает в себя:

1) внешние входы в программу;

2) используемые в программе константы;

3) функции, которые не имеют аргументов.

Слово «терминал» используется, так как перечисленные выше объекты соответствуют терминальным (конечным, висячим) вершинам в древовидных структурах. Терминал дает некоторое (численное) значение, у него нет входных аргументов, и он имеет нулевую «арность».

Следует отметить тесную связь внешних входов с обучающими выборками, которые часто используются в ГП. При этом каждая переменная (признак, фактор и т.п.) обучающей выборки соответствует своему терминалу.

Функциональное множество

Функциональное множество состоит из операторов (взятых у языков программирования) и различных функций. Оно может быть достаточно широким и включать типовые конструкции различных языков программирования, такие как:

1) булевы функции И, ИЛИ, НЕ и т.п.;
2) арифметические функции сложения, вычитания, умножения, деления;
3) трансцендентные функции (тригонометрические, логарифмические);
4) функции присваивания значения переменным (a:=2);
5) условные операторы (if then, else: case или switch операторы ветвления);
6) операторы переходов (go to, jump, call- вызов функции);
7) операторы цикла (while do, repeat until, for do);
8) подпрограммы и функции.

С одной стороны, терминальное и функциональное множество должны быть достаточно большими для представления потенциального решения. Но другой стороны не следует сильно без необходимости расширять функциональное множество, поскольку при этом резко возрастает

пространство поиска решений. Набор функций существенно зависит от решаемой задачи.

Структуры для представления программ

Терминалы и функции должны быть объединены по определенным правилам в некоторые структуры, которые могут представлять программы и использоваться при их выполнении. Выбор структуры существенно влияет на порядок выполнения программы, распределение и использование локальной или глобальной памяти.

В настоящее время наиболее распространенными структурами для представления особей (потенциальных решений проблемы) являются:

1) древовидное представление;

2) линейная структура;

3) графоподобная структура.

Древовидное представление

В качестве примера рассмотрим арифметическую формулу, которую удобно представлять деревом. Рассмотрим арифметическую формулу $\frac{d}{e}-a*(b+c)$ (в обычном представлении). Отметим, что дерево (генотип) рис.4.1 также представляет эту формулу (фенотип) $\frac{d}{e}-a*(b+c)$.

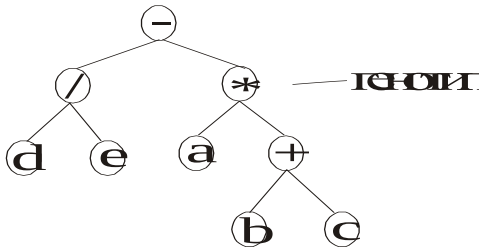


Рис.4.1 Древовидное представление формулы d/e-a*(b+c).

При этом листья дерева соответствуют терминалам, а внутренние узлы – функциям.

Древовидная форма представления генотипа оказывается для данного класса задач более эффективной, и позволяет работать с программами или выражениями различной длины. Важным аспектом является также использование памяти при выполнении программы. Древовидная структура позволяет использовать только локальную память в процессе выполнения. Локальность памяти встроена в саму древовидную структуру. Значения переменных доступны для функции только в дереве, корню которого соответствует функция. Например, значения переменных d, e являются локальными относительно узла ‘/’.

Линейные структуры

Рассмотрим один из возможных вариантов линейной структуры ГП, ориентированного на программирование на языке Си. При этом каждая особь (программа) представлена последовательностью переменной длины операторов Си. Ниже представлен пример такой программы.

```
void int u
double v[8]
{
…
v[0]=v[5]+75;
v[7]=v[0]-69;
if (v[1]>0)
  if (v[4]>21)
    v[4]=v[2]*v[1];
v[2]=v[5]+v[4];
v[6]=v[4]-4;
v[1]=sin(v[6]);
if(v[0]>v[1])
  v[5]=v[7]+115;
if(v[1]<=v[6])
v[1]=sin(v[7]);
}
```

Здесь функциональное множество состоит из арифметических операций, условных операторов if и вызовов функций.

Переменные и константы вместе образуют множество терминальных символов. При этом подходе любой оператор кодируется 4-х мерным вектором, компонентами которого является тип оператора (одна компонента) и адреса (указатели) переменных и констант.

Например, оператор $v_i=v_j+c$ представляется вектором (+,i,j,c). Каждая компонента использует 1 байт памяти, следовательно, число переменных (и констант) ограниченно сверху 256.

Графоподобные структуры

Рассмотрим типичную блок-схему программы, которая представлена на рис.4.2.

Здесь каждая особь – программа представляется в виде графа. Вершина графа представляет линейный участок программы. Она имеет 2 части – собственно линейную программу и узел ветвления. Из рисунка видно, что блок-схема линейного графа более естественна, чем линейная (древовидная) форма представления программы.

Это объясняется тем, что многие программы содержат операторы ветвления, после которых может быть вызов другой части кода программы, не входящей в линейную последовательность операторов. Линейная программа выполняется в том случае, когда достигается данный узел графа при интерпретации программы на конкретных данных. После выполнения линейной части выбирается узел-последователь в соответствии с функцией ветвления этого узла. В течение интерпретации узлы только одного пути графа от входов до выхода будут выполняться в зависимости от текущих значений переменных условий операторов ветвления.

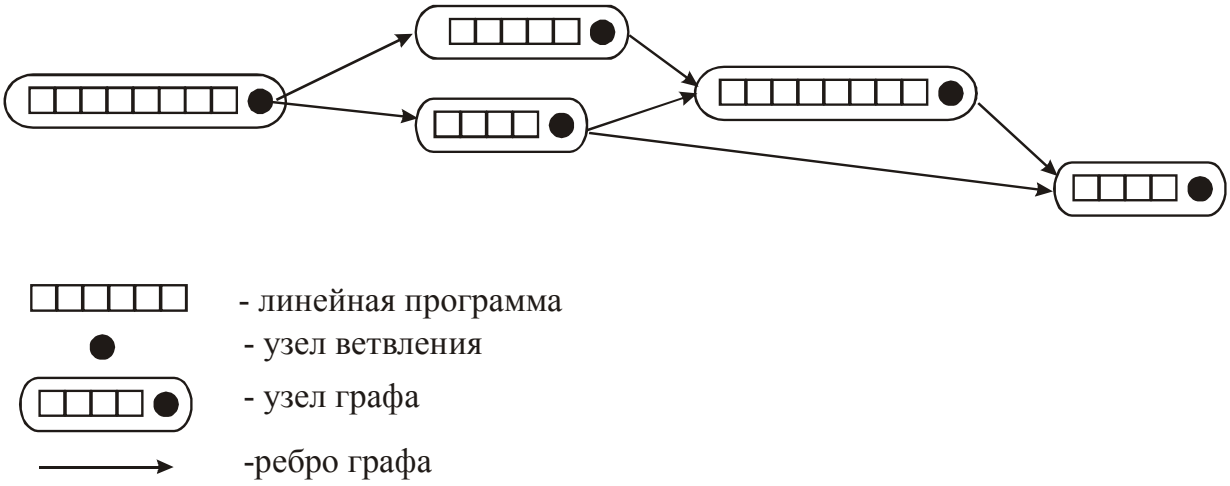


Рис.4.2. Представление программы в виде графа

Реализация линейных подструктур использует список операторов Си (или другого языка программирования) переменной длины, которые оперируют с переменными или константами и полученные значения присваиваются переменным. После выполнения программы, вычисленные значения запоминаются в “выходные” переменные.

Функция ветвления также является оператором Си, который оперирует с теми же переменными, что и линейная программа, но этот оператор только читает значения этих переменных.

На рис.4.3 представлен детально узел графа, соответствующий некоторой части линейной программы.

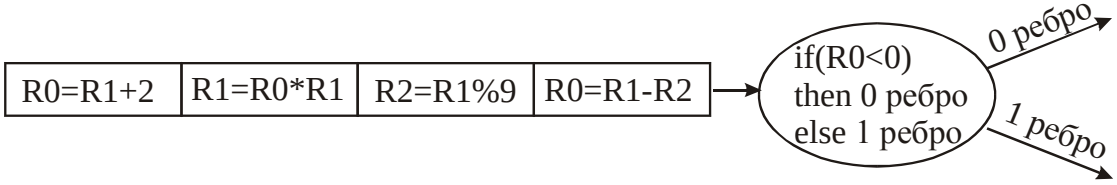


Рис.4.3. Подпрограмма, соответствующая узлу графа.

Данная модель, как и предыдущая, использует при выполнении программы глобальную память. Графоподобные структуры позволяют расширить класс задач, которые могут быть решены с помощью ГП, так как являются более общими по сравнению с первыми двумя и допускают эффективную реализацию.

Известен другой способ применения структур графов в ГП, называемый PADO. Здесь каждая программа представляется ориентированным графом, содержащим узлы, индексированную память для входных переменных и стек, как показано на рис.4.4. В произвольном ориентированном графе каждый узел может иметь N исходящих дуг.

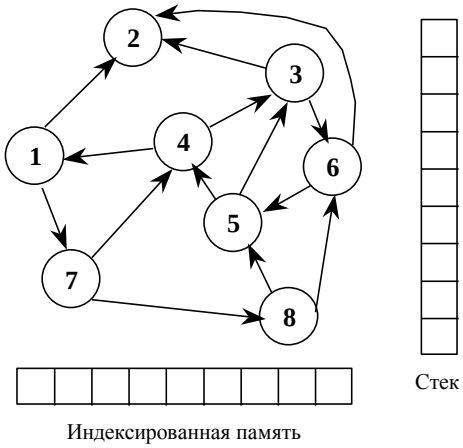


Рис.4.4. Представление программы ориентированным графом.

Эта структура не просто показывает потоки информации и управление ходом выполнения программы. Узел в графе соответствует сегменту программы и имеет две части: «действие» и «ветвление». Часть «действие» содержит константы и функцию, которая выполняется при достижении данного узла в процессе интерпретации программы. Данные передаются между узлами через стек. Часть «действие» получает данные из (верхушки) стека и после выполнения соответствующей функции передает преобразованные данные снова в стек. После этого выполняется

«ветвление», которое определяет ветвь следующего выполняемого узла на основании данных стека, памяти или специальных констант ветвления. Очевидно, что в процессе интерпретации необязательно посещаются все вершины графа. Каждая программа имеет две специальных вершины «старт» и «конец» и кроме этого может содержать некоторые специальные узлы типа «вызов подпрограммы». Поскольку граф может содержать обратные связи, то вершина «конец» при интерпретации на некоторых входных данных может быть недостижимой вследствие «зацикливания». Поэтому необходимо контролировать и ограничивать время выполнения программы.

Инициализация начальной популяции

Данный этап в ГП является не таким простым, как в классическом ГА, где генерация случайных двоичных строк не представляет особых проблем. Это связано с различной сложностью особей и их структурой. Естественно методы инициализации начальной популяции различны для разных форм представления программ. Одним из важнейших параметров в ГП является максимально возможный размер (сложность) программы.

Инициализация древовидных структур

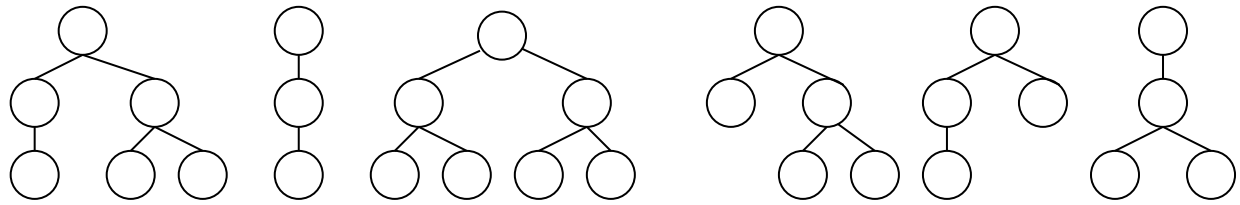
Для деревьев в качестве меры сложности используется максимальная глубина дерева или общее число узлов в дереве. Глубиной узла называется минимальное число узлов, которые необходимо пройти от корня дерева к этому узлу. Максимальной глубиной дерева Dm называется максимально возможная глубина в дереве для терминального символа (листа). Если

арность каждого узла равна двум, то общее число узлов не превышает 2^{B_b}, которое также используется в качестве меры сложности.

Инициализация древовидных структур выполняется путем случайного выбора функциональных и терминальных символов при заданной максимальной глубине дерева. Применяются два основных метода: а) полная (full) и б) растущая (grow) инициализация.

В полном методе при генерации дерева, пока не достигнута максимальная глубина, допускается выбор только функциональных символов, а на последнем уровне (максимальной глубины) выбираются только терминальные символы.

В растущей инициализации генерируются нерегулярные деревья с различной глубиной листьев вследствие случайного на каждом шаге выбора функционального или терминального символа. Здесь при выборе терминального символа рост дерева прекращается по текущей ветви и поэтому дерево имеет нерегулярную структуру.



а) б)

Рис.4.5. Деревья, генерируемые при инициализации разными методами.

При использовании первого метода начальная популяция содержит однородное множество структур, что способствует вырождению генетического материала (и преждевременной сходимости в локальных экстремумах). Поэтому на практике часто эти два метода используют одновременно следующим образом. Начальная популяция генерируется так, чтобы в нее входили деревья с разной максимальной длиной примерно

поровну (для нашего примера D_m=1, D_m=2, D_m=3. D_m=4). Для каждой глубины первая половина деревьев генерируется полным методом, а вторая – растущей инициализацией.

Инициализация линейных структур

В этом случае процесс инициализации выполняется совершенно по-другому. Прежде всего, особь – программа разбивается на следующие четыре части:

1) Заголовок;
2) тело;
3) «подвал»;
4) выход (возврат).

Из них только тело программы генерируется с помощью эволюции, остальные части программы используются стандартные и заготавливаются заранее. Алгоритм инициализации можно сформулировать следующим образом:

1) Выбор случайной длины из заданного диапазона;
2) Копирование заготовленного заголовка;
3) Инициализация и пополнение собственно операторов в программу пока не достигнута длина, определенная в пункте 1. Операторы выбираются случайно, сначала тип, затем переменная или константа из заданного диапазона;
4) Копирование в конец программы заготовленного «подвал»;
5) Копирование в конец программы заготовленных операторов выхода.

Кроссинговер в генетическом программировании

В начальной популяции особи (потенциальные решения), как правило, имеют низкие (плохие) значения фитнесс-функции. В процессе эволюции с помощью генетических операторов популяция развивается и значения

фитнесс-функции улучшаются. В ГП используются те же, что и в ГА, генетические операторы кроссинговера, мутации и репродукции. Отметим, что в терминах машинного обучения они соответствуют операторам поиска решения. Далее мы рассмотрим выполнение генетических операторов на ранее определенных структурах.

Выполнение кроссиновера на древовидных структурах

Для древообразной формы представления используются следующие три основные оператора кроссинговера (ОК):

а) узловой ОК;

б) кроссинговер поддеревьев;

в) смешанный.

В узловом операторе кроссинговера выбираются два родителя (два дерева) и узлы в этих деревьях. Первый родитель называется доминантом, второй – рецессивом. Узлы в деревьях могут быть разного типа. Поэтому сначала необходимо убедиться, что выбранные узлы у родителей являются взаимозаменяемыми. Если узел во втором родителе не соответствует типу узла первого родителя, то случайным образом выбирается другой узел во втором родителе, который опять подлежит проверке на предмет совместимости. Далее производится обмен узлов.

В кроссинговере поддеревьев родители обмениваются не узлами, а определяемыми ими поддеревьями. Оператор выполняется следующим образом:

1. Выбираются родители (один – доминантный, другой – рецессивный). Далее необходимо убедиться, что выбранные узлы взаимозаменяемы, т.е. принадлежат одному типу. Иначе, как и в предыдущем случае в рецессивном дереве выбирается другой узел с последующей проверкой.
2. Затем производится обмен поддеревьев, которые определены этими узлами.

3. Вычисляется размер ожидаемых потомков. Если ожидаемый размер (сложность потомка) не превышает заданный порог, такой обмен ветвями запоминается.

Этот тип ОК является основным. При этом под размером (под)дерева понимается, как и ранее, либо его высота, либо число его вершин .

При смешанном операторе кроссинговера для некоторых узлов выполняется узловой ОК, а для других - кроссиновер поддеревьев.

Кроссинговер на линейных структурах

Скрещивание на линейных структурах выполняется достаточно просто. Здесь у родителей выполняется обмен линейными сегментами, как это показано на рис.4.6.

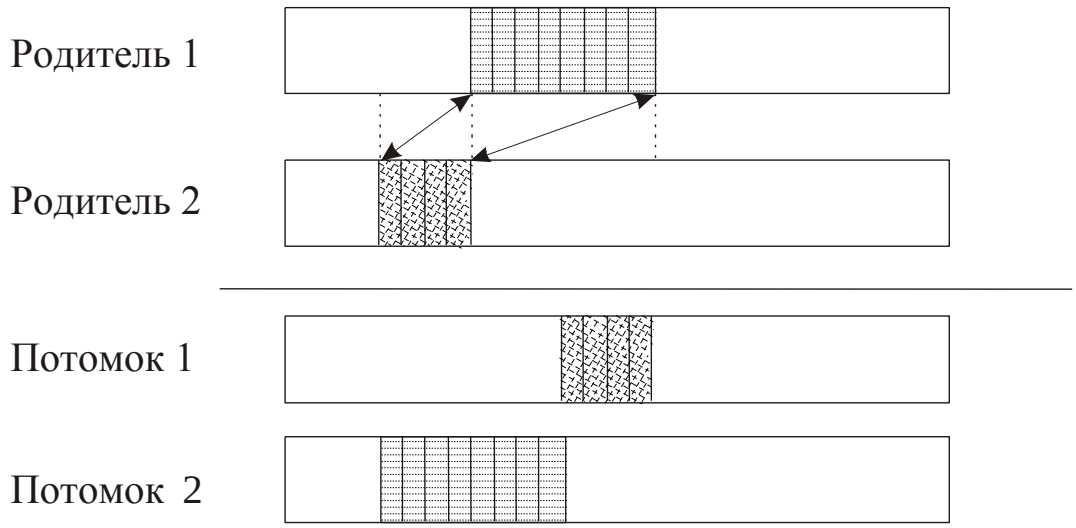


Рис.4.6. Кроссинговер на линейных структурах.

При этом в каждом из двух родителей выбирается сегмент, начиная со случайной позиции и имеющий случайную длину. Далее производится обмен выбранных сегментов программ. Если размер хотя бы одного из потомков

превышает некоторый порог, то результаты ОК аннулируются. При этом точки скрещивания выбираются только между операторами.

Выполнение кроссинговера для графоподобных структур

Кроссинговер комбинирует генетический материал из двух родительских программ путем обмена некоторых частей программ. ОК для этой модели может быть реализован двумя способами.

Первый способ похож на кроссиновер поддеревьев, который определен на древообразных структурах путем обмена поддеревьев. Здесь обмен производится подграфами, как это показано на рис.4.7.

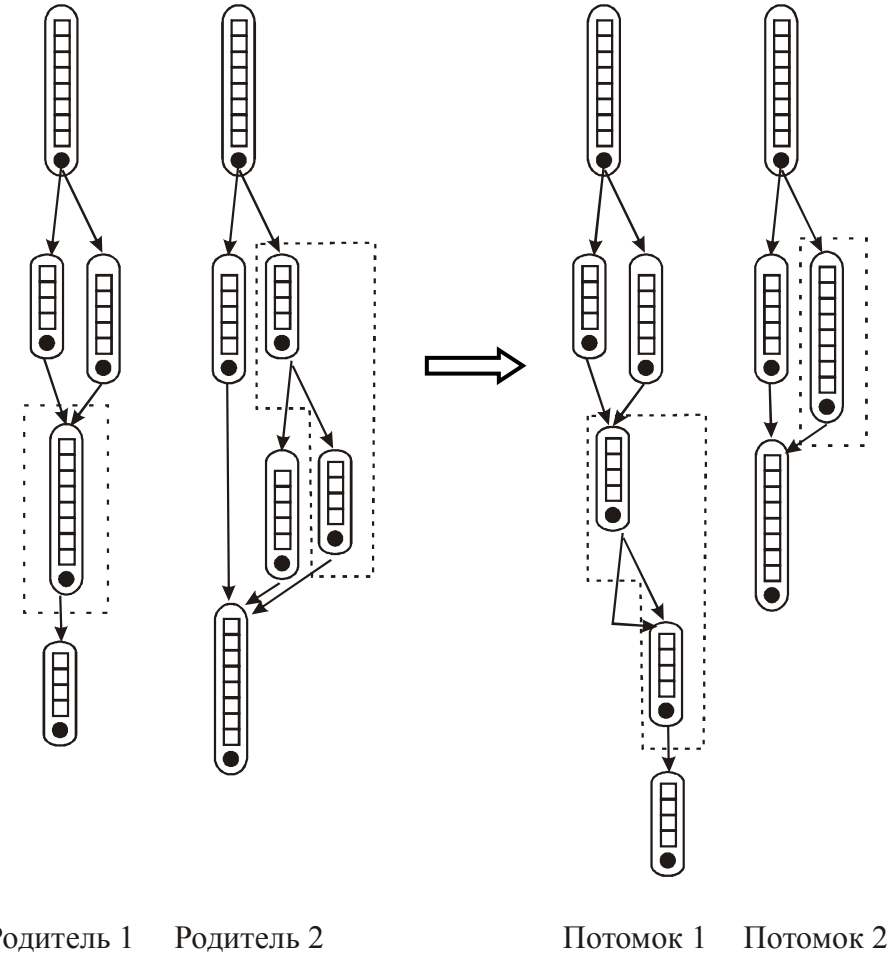


Рис.4.7. Кроссинговер на графоподобных структурах.

При этом в каждой особи - родителе выбирается случайным образом множество смежных узлов и производится обмен 2-х подграфов между этими подграфами.

Второй тип кроссинговера выполняет линейный ОК. Здесь для каждого родителя выбирается линейный сегмент (в одном узле графа), начинающийся со случайной позиции данного сегмента и имеющий случайную длину. Далее, как обычно, производится обмен этими линейными сегментами. Если размер хотя бы одного потомка превышает некоторый порог, то результаты ОК аннулируются и выполняется обмен равными сегментами меньшей длины. Обычно линейный ОК выполняется в вероятностью $P_l = 0.1$. Пример выполнения этого ОК представлен на рис.4.8.

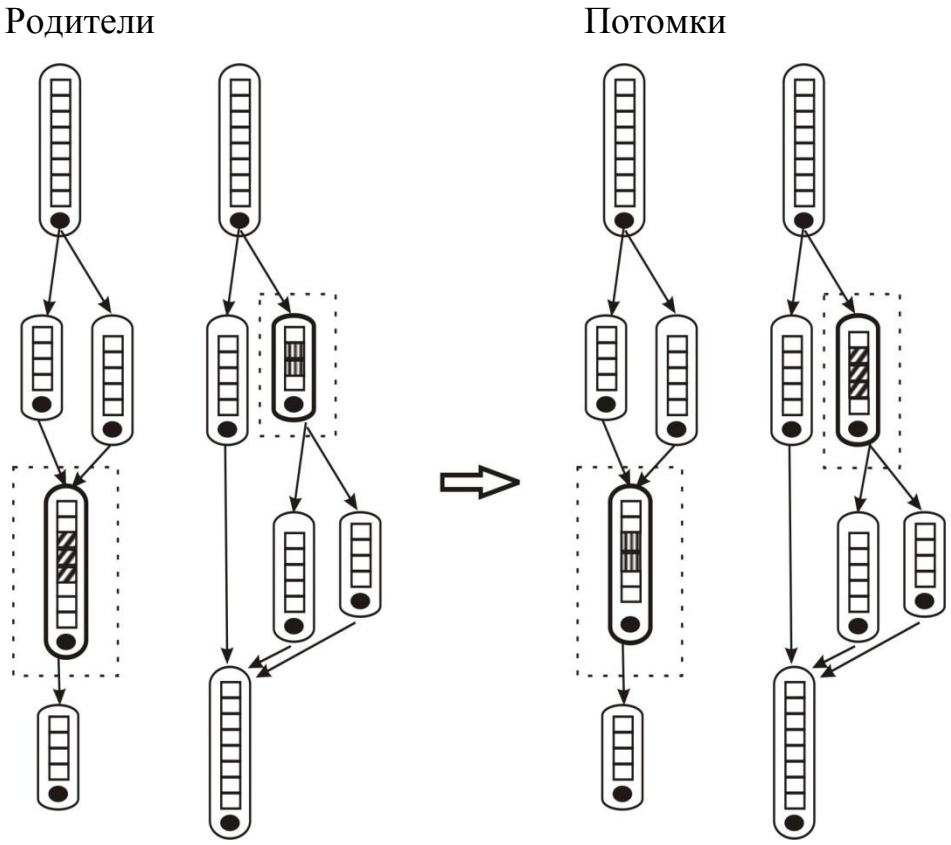


Рис.4.8. Линейный кроссинговер на графах.

Как правило, в процессе эволюции используются ОК обоих типов. В целом ОК выполняется следующим образом:

1) выбор точки скрещивания P_1, P_2 в обоих родителях.
2) выбор с заданной вероятностью типа ОК (для 1-го типа с вероятностью P_G, для 2-го с вероятностью $1-P_G$). Если выбран 1-й тип то переход на п.3, иначе на п.4.
3) если размер потомка не превышает порог, то выполнить ОК 1-го типа, переход на п.5.
4) Если размер потомка не превышает порог, то выполнить ОК 2-го типа.
5) Конец.

Мутация в генетическом программировании

После выполнения кроссиговера с заданной малой вероятностью выполняется мутация для выбранной одной особи-программы.

Выполнение мутации на древовидных структурах

Для деревьев используются следующие операторы мутации (ОМ):

а) узловая;

б) усекающая;

в) растущая.

Узловая мутация выполняется следующим образом:

- выбрать случайным образом узел, подлежащий мутации, определить его тип;
- случайно выбрать из соответствующего множества вариантов узлов отличный от рассматриваемого узел;
- поменять исходный узел на выбранный.

Усекающая мутация производится так:

- определяется или выбирается узел;
- случайным образом выбирается терминальный символ из заданного множества;
- обрезается ветвь узла мутации;

- вместо обрезанной ветви помещается выбранный терминальный символ.

Растущая мутация выполняется следующим образом:

- определяется узел мутации;
- если узел нетерминальный то необходимо отсечь ветви исходящие из него, иначе выбрать другой узел.
- вычислить размер (сложность) остатка дерева;
- вместо отсеченного дерева вырастить случайным образом новое дерево так, чтобы размер нового построенного дерева не превышал заданный порог.

Это очень мощный оператор, который обладает большими возможностями.

Выполнение мутации на линейных структурах

Здесь мутация выполняется совершенно другим способом. Прежде всего, из линейного сегмента (тела программы) случайно выбирается оператор (команда) и в нем производятся изменения одного из следующих видов:

1) имя переменной заменяется на другое случайно выбранное из заданного множества;
2) оператор может быть также изменен случайным образом на некоторый другой из функционального множества;
3) может быть случайно изменено значение константы на некоторое другое значение из заданного диапазона.

Выполнение мутации на графоподобных структурах

В этом случае оператор мутации случайным образом выбирает множество узлов и изменяет либо узел линейной программы, либо узел ветвления, либо число исходящих из условного оператора ветвей. То есть

ОМ не строит новые линейные последовательности. Измененная ОМ программа помещается назад в популяцию.

Фитнесс-функция в генетическом программировании

В отличие от генетических алгоритмов, где часто при поиске экстремумов в качестве фитнесс-функции используется исходная целевая функция, в ГП фитнесс-функция обычно определяет меру близости между реальными y_i и требуемыми d_i выходными значениями. Поэтому в качестве фитнесс-функции часто используется абсолютная или квадратичная ошибка.

В этом случае фитнесс-функция использует обучающее множество данных, на котором выполняется обучение системы. С помощью фитнесс-функции в процессе обучения реализуется обратная связь, которая показывает насколько хорошо данная особь-программа реализует необходимую функцию на обучающем множестве.

Естественно разработано множество типов фитнесс-функций вид которых существенно зависит от исследуемой проблемы. В некоторых случаях, кроме близости решений учитываются и другие критерии, например, длина или время выполнения программы. В этом случае говорят о многокритериальных фитнесс-функциях.

Интроны

Программы, построенные с помощью методов ГП, имеют тенденцию к накоплению интронов – ненужных и непригодных участков кода.

Например:

$(NOT(NOT\ x))$,

(AND (ORXX)),

(+ (-XX)),

(MOVE_LEFT MOVE_RIGHT),

(IF (2=1) . . .),

A:=A.

Таких фрагментов в программе возникает достаточно много (их количество может достигать 60%), и обнаружение и удаление интронов представляет серьезную проблему в ГП. Разработаны специальные методы для их устранения.

Общий алгоритм генетического программирования

Таким образом, для решения задачи с помощью ГП необходимо выполнить описанные выше предварительные этапы:

1) Определить терминальное множество;
2) Определить функциональное множество;
3) Определить фитнесс-функцию;
4) Определить значения параметров, такие как мощность популяции, максимальный размер особи, вероятности кроссинговера и мутации, способ отбора родителей, критерий окончания эволюции (например, максимальное число поколений) и т.п.

После этого можно разрабатывать непосредственно сам эволюционный алгоритм, реализующий ГП для конкретной задачи.

Например, решение задачи на основе ГП можно представить следующей последовательностью действий.

1) установка параметров эволюции;
2) инициализация начальной популяции;
3) t:=0;
4) оценка особей, входящих в популяцию;
5) t:=t+1;
6) отбор родителей;
7) создание потомков выбранных пар родителей – выполнение оператора кроссинговера;
8) мутация новых особей;

9) расширение популяции новыми порожденными особями;

10) сокращение расширенной популяции до исходного размера;

11) если критерий останова алгоритма выполнен, то выбор лучшей особи в конечной популяции – результат работы алгоритма. Иначе переход на шаг 4.

Порядок выполнения лабораторной работы

1. При домашней подготовке:

- изучить теоретический материал;
- ознакомиться с типами структур для представления программы;
- рассмотреть способы инициализации, начальной популяции;
- рассмотреть способы выполнения операторов кроссинговера и мутации;
- выполнить индивидуальное задание на любом языке высокого уровня с необходимыми комментариями и выводами.

2. Во время занятия:

- продемонстрировать результаты выполнения работы;
- получить допуск к защите лабораторной работы.

3. Защитить отчет по лабораторной работе.

Задание

1. Разработать эволюционный алгоритм, реализующий ГП для нахождения заданной по варианту функции (таб. 4.1).

- Структура для представления программы – древовидное представление.
- Терминальное множество: переменные x_1, x_2, x_3, …, x_n, и константы в соответствии с заданием по варианту.
- Функциональное множество: +, -, *, /, abs(), sin(), cos(), exp(), возведение в степень,

– Фитнесс-функция – мера близости между реальными значениями выхода и требуемыми.

2. Представить графически найденное решение на каждой итерации.
3. Сравнить найденное решение с представленным в условии задачи.

Таблица 4.1. Индивидуальные задания.

№ вв.	Вид функции	Кол-во пер-ых N	Промежуток исследования
1	f1(x)=sum(x(i)^2), i=1:n;	10	-5.12<=x(i)<=5.12.
2	f1a(x)=sum(i·x(i)^2), i=1:n;	9	-5.12<=x(i)<=5.12.
3	f1b(x)=sum(sum(x(j)^2), j=1:i), i=1:n;	8	-5.536<=x(i)<=65.536
4	f2(x)=sum(100·(x(i+1)-x(i)^2)^2+(1-x(i))^2), i=1:n-1;	7	-2.048<=x(i)<=2.048.
5	f6(x)=10·n+sum(x(i)^2-10·cos(2·pi·x(i))); i=1:n;	9	-5.12<=x(i)<=5.12.
6	f7(x)=sum(-x(i)·sin(sqrt(abs(x(i))))); i=1:n;	10	-500<=x(i)<=500.
7	f8(x)=sum(x(i)^2/4000)-prod(cos(x(i)/sqrt(i)))+1; i=1:n;	5	-600<=x(i)<= 600.
8	f9(x)=sum(abs(x(i))^(i+1)); i=1:n;	8	-1<=x(i)<=1.
9	f10(x)=-a·exp(-b·sqrt(1/n·sum(x(i)^2)))-exp(1/n·sum(cos(c·x(i))))+a+exp(1); a=20; b=0.2; c=2·pi; i=1:n;	4	-2.768<=x(i)<=32.768
10	f11(x)=-sum(c(i)·(exp(-1/pi·sum((x-A(i))^2))·cos(pi·sum((x-A(i))^2)))), i=1:m, m=5; A(i),C(i)<>0, m=5	4	0<=x(i)<=10.
11	f12(x)=-sum(sin(x(i))·(sin(i·x(i)^2/pi))^(2·m)), i=1:n, m=10;	5	0<=x(i)<=pi.
12	fBran(x1,x2)=a·(x2-b·x1^2+c·x1-d)^2+e·(1-f)·cos(x1)+e;	2	-5<=x1<=10, 0<=x2<=15.

	a=1, b=5.1/(4·pi^2), c=5/pi, d=6, e=10, f=1/(8·pi);		
13	fEaso(x1,x2)=-cos(x1)·cos(x2)·exp(-((x1-pi)^2+(x2-pi)^2));	2	-100<=x(i)<=100, i=1:2.
14	fGold(x1,x2)=[1+(x1+x2+1)^2·(19-14·x1+3·x1^2-14·x2+6·x1·x2+3·x2^2)]·[30+(2·x1-3·x2)^2·(18-32·x1+12·x1^2+48·x2-36·x1·x2+27·x2^2)]	2	-2<=x(i)<=2, i=1:2.
15	fSixh(x1,x2)=(4-2.1·x1^2+x1^4/3)·x1^2+x1·x2+(-4+4·x2^2)·x2^2;	2	-3<=x1<=3, -2<=x2<=2.

Содержание отчета.

1. Титульный лист.
2. Индивидуальное задание по варианту.
3. Краткие теоретические сведения.
4. Программа и результаты выполнения индивидуального задания с комментариями и выводами.
5. Письменный ответ на контрольный вопрос по варианту (номер контрольного вопроса совпадает с номером варианта).

Контрольные вопросы

1. Чем отличаются терминальное и функциональное множества?
2. Какие структуры используются для представления программ в ГП?
3. Опишите древовидное представление.
4. Опишите линейное представление программы.
5. Опишите представление программы в виде графа.
6. Какие знаете методы инициализации древовидных структур?
7. Как производится инициализация линейных структур?
8. Какие виды кроссинговера вы знаете для древовидных структур?

9. Как выполняется кроссинговер на линейных структурах?
10. Какие виды кроссинговера вы знаете для графоподобных структур?
11. Какие виды мутации вы знаете для древовидных структур?
12. Как производится мутация на линейных структурах?
13. Как можно определить фитнесс-функцию в ГП?
14. Что такое интроны?
15. Приведите общий алгоритм ГП.