Серия «Математика»
2011. Т. 4, № 4. С. 66—81

Онлайн-доступ к журналу:
http://isu.ru/izvestia

ИЗВЕСТИЯ
Иркутского
государственного
университета

УДК 518.517

Эволюционные алгоритмы в задаче минимизации булевых функций *

Б. П. Ильин
Байкальский государственный университет экономики и права

Аннотация. В статье дан обзор эволюционных алгоритмов и получены настройки параметров для решения задачи минимизации полиномиального представления булевых функций.

Ключевые слова: булевы функции; оптимизация; эволюционные алгоритмы; полиномиальное представление булевых функций.

1. Постановка задачи оптимизации

К задачам поиска оптимума сводятся многие из проблем прикладной математики. Когда для изучения какого-нибудь сложного явления конструируется математическая модель, к оптимизации прибегают для того, чтобы определить такую структуру и такие параметры последней, которые обеспечивали бы наилучшее согласование с реальностью. Другой традиционной областью применения оптимизации являются процедуры принятия решений, так как большинство из них нацелено именно на то, чтобы сделать «оптимальный» выбор. Помимо оптимизационных задач, представляющих самостоятельный интерес, на практике часто возникают задачи, которые «встроены» в некоторые вычислительные процессы, где они играют хотя и существенную, но все же вспомогательную роль. Это настолько типичная ситуация, что при описании процесса в целом далеко не всегда указывают на наличие подобных задач; к примеру, сказав, что процесс предполагает определение точки, в которой какая-то функция достигает своего критического значения, могут и не добавить, что эта точка будет найдена решением оптимизационной задачи [1].

* Работа выполнена при финансовой поддержке РФФИ, грант 09–01–00476.

Одной из задач логического синтеза является задача минимизации полиномиального представления булевых функций. Данная задача заключается в том, что одна и та же булева функция может быть представлена в виде разных полиномиальных представлений, но требуется найти минимальное представление, т. е. функцию минимальной сложности. В данном случае сложность булевой функции определяется как число слагаемых в полиноме.

В настоящее время для решения этой задачи существуют точные алгоритмы, которые основываются на полном переборе функций меньшего числа переменных. Данные алгоритмы позволяют получать точное решение задачи минимизации полиномиального представления булевых функций только лишь для функций с числом аргументов не более шести. Для функций с большим числом переменных недостаточно ресурсов современных компьютеров. В силу ограничений накладываемых на возможности современных вычислительных систем данная задача не может быть решена исключительно с использованием точных алгоритмов.

Такими методами решения данного типа задач могут выступать эволюционные алгоритмы, которые являют собой достаточно мощное средство, для нахождения точных или приближенных решений различных оптимизационных задач. Однако краеугольным камнем данного класса алгоритмов является подбор их параметров. В связи с этим была выдвинута задача отыскания таких параметров алгоритмов, при котором сложность любой булевой функции 6-ти переменных можно будет найти за минимальное время, при этом результат будет либо соответствовать точному значению, либо отличаться на единицу.

Решение задачи поиска параметров эволюционных алгоритмов заключалось в следующем:

1) Создан тестовый набор из 100 функций 6 переменных, так как длина такой функции равна 64 битам, то начальная популяция была сгенерирована в следующем виде:
   - первые 10 функций с максимальной сложностью;
   - следующие 5 функций получены случайным образом, т. е. представляют собой нулевой вектор, в котором случайным образом значения нескольких бит (не более пяти) изменены на противоположные;
   - следующие 80 функций разбиты на десятки таким образом, что ближе к концу списка число единиц в каждом десятке увеличивается на 10. Позиции единиц в этих векторах также случайны;
   - последние 5 функций получены путем установки в случайных позициях нулевых значений бит. Сама функция изначально задается единичным набором.

2) Сгенерированный массив функций оценивается с помощью алгоритма, позволяющего точно оценить сложность функции 6-ти переменных с использованием библиотек [6].

3) Параметры, с ограниченным количеством значений (тип кроссовера, тип селекции и т. д.) принимают все возможные варианты значений. Параметры, с неограниченным количеством значений (число итераций, размер популяции и т. д.) принимают значения с заданным шагом. На каждой итерации находится с помощью эволюционных алгоритмов значение сложности каждой из 100 функций.

4) Для каждого набора параметров определяется количество совпадающих значений, полученных с помощью точного и приближенного алгоритмов и количество значений, отличающихся от точного на единицу.

5) Наборы параметров алгоритмов, на которых значение сложности булевых функций находится точное, либо отличающееся на единицу подвергаются дальнейшему анализу.

2. Генетический алгоритм

В конце 1960-х гг. американским исследователем Джоном Холландом была предложена идея генетических алгоритмов, в основе которых лежит идея эволюции живых организмов, предложенная Чарльзом Дарвиным. Согласно ей, группы организмов постепенно развиваются благодаря естественному отбору, который заключается в рекомбинации генотипов, мутации и их комбинации. Первый ввел в обиход термин «генетический алгоритм» Д. Багли в 1967 г.

Генетический алгоритм (англ. genetic algorithm) можно определить как эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путем последовательного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию.

Для решения задачи минимизации полиномиального представления булевых функций используются точные алгоритмы. Однако при увеличении числа переменных нахождение сложностей булевых функций невозможно в силу ограниченности вычислительных возможностей современных вычислительных систем. Для преодоления этих ограничений существует возможность использования эволюционных алгоритмов, в частности генетический алгоритм. Все представленные в этой статье алгоритмы обладают набором параметров, которые определяют

ход их выполнения. Генетический алгоритм не является исключением. Для него были определены следующие параметры:

- булеву функцию, для которой определяется ее сложность;
- количество итераций, которое должен пройти алгоритм;
- количество мутаций, которое произойдет за время выполнения алгоритма;
- количество генов, которые претерпевают мутацию;
- размер популяции;
- тип генерирования начальной популяции;
- тип кроссовера;
- тип закона распределения при осуществлении выборки особей.

Как и в классическом ГА, в виде функций было реализовано несколько генетических операторов. Алгоритм состоит из следующих стадий:

1) Инициализация начальной популяции — на первом шаге генерируется популяция претендентов на решение, которая даст в результате выполнения операторов селекции и скрещивания новое потомство.

2) Следующим этапом происходит оценивание претендентов. В качестве функции приспособленности выступает сложность булевых функций. Для достижения этой реализована функция AbsMin(), которая возвращает с помощью сокращенной библиотеки [6] точное значение сложности любой функции 5-ти переменных

3) Оператор селекции Selection() осуществляет выбор представителей для применения генетических операторов. Для этого используются следующие законы распределения вероятностей:

   а) равномерный — вероятность выбора того или иного представителя одинакова. Позволяет выбирать представителей с одинаковой вероятностью, вне зависимости от их функции приспособленности;

   б) показательный — случайная величина X имеет показательное или экспоненциальное распределение с параметром $\lambda < 0$. Позволяет отбирать представителей таким образом, что вероятность выбора того или иного представителя уменьшается с увеличением значения функции приспособленности;

   в) усеченный нормальный — распределение, получаемое из классического нормального, при ограничении интервала возможных значений. Позволяет отбирать представителей таким образом, что вероятность выбора того или иного представителя уменьшается с увеличением значения функции приспособленности;

4) Оператор скрещивания — кроссовер. Определяет случайным образом одну или две точки скрещивания, в результате чего получает двух новых представителей, с генами от первого и второго родителя. По числу точек скрещивания реализованы:
   - Одноточечный кроссовер (функция Crossover_1point());
   - Двухточечный кроссовер (функция Crossover_2points()).

5) В зависимости от количества мутаций, которое задается, как параметр ГА следующим этапом может выступать оператор мутации (функция mutation()). Подвергаться мутации может как один бит конкретного представителя, так и сразу несколько — это значение, передается как параметр функции.

6) Сортировка (функция Sort()) — сортирует представителей по возрастанию функции приспособленности, в качестве которой выступает сложность булевой функции.

7) Завершающим этапом каждой итерации является формирование новой популяции. В новую родительскую популяцию извлекается половина представителей родительской популяции и половина популяции потомков. Тем самым, качественные решения задачи — потомки, не выпадают в ходе дальнейшего выполнения алгоритма. После выполнения всех этих этапов алгоритм возвращается к п. 2 и продолжается то число раз, которое пользователь задал при запуске данной программы.

Алгоритм 1. Function Gen_alg($f(x_1, \ldots, x_6)$, param1,. . . ,param13)
{Var /*определение переменных*/
arr_p /* массив потомков*/
Gen_ind(); // генерирование начальной популяции
AbsMin(); // оценка приспособленности хромосом
for i = 0 to n do {
Selection(); // выбор хромосом для скрещивания
if(param[9]==1) {Crossover_1point();} //одноточечный кроссовер
if(param[9]==2) {Crossover_2points();} //двухточечный кроссовер
if(i%mut_interval==0) {mutation(); //мутация хромосом }}
AbsMin(); // оценка приспособленности хромосом
Return res;}

Для генетического алгоритма была выбрана следующая стратегия — производился перебор итераций и размера популяции для всех параметров до тех пор, пока для всех 100 функий не будут найдены либо точные значения функции, либо значения, отличающиеся на единицу (Рис. 1).

С помощью данного метода были проведены испытания ГА с различными параметрами и были получены следующие результаты:

– Как видно из рисунка 1 при увеличении популяции претендентов на решение число итераций, для сходимости алгоритма к требуемому уровню точности, уменьшается, а также на это влияет использование усеченного нормального закона распределения для селекции претендентов. Для 10 представителей усеченный нормальный закон сходится уже на 400 итерациях, в то время как показательный лишь на 600. Стоит отметить, что с увеличением числа представителей эти отличия нивелируются.

– При небольшом числе итераций наиболее качественные значения алгоритма получены с использованием двухточечного кроссовера, с использованием первого типа генерации начальной совокупности, и достаточно большом размере этой популяции (Рис. 2). На рисунке 2 видно, что при существенном увеличении числа мутирующих генов (до 20) и выборе двухточечного кроссовера ситуация несколько меняется: для 10 представителей алгоритму с усеченным нормальным законом требуется уже более полутора тысяч итераций, в то время как равномерному порядка семисот. Но так же, как и в предыдущем случае, при увеличении числа представителей, разница в эффективности этих законов несущественна.

– Оптимальным числом мутаций — является треть от общего числа итераций, число мутирующих генов — до половины от общей длины функции.

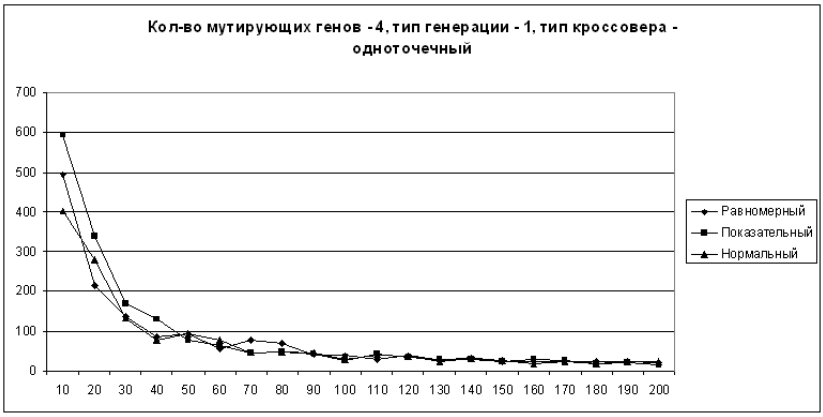

Рис. 1. График зависимости числа итераций от размера популяции для генетического алгоритма при использовании одноточечного кроссовера.

– При увеличении числа мутирующих генов, возрастает число итераций вне зависимости от остальных параметров.

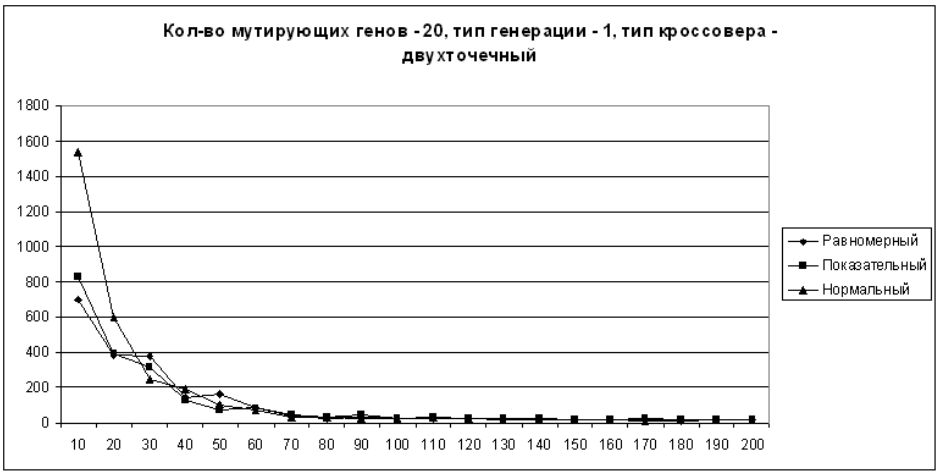

Рис. 2. График зависимости числа итераций от размера популяции для генетического алгоритма при использовании двухточечного кроссовера.

3. Алгоритм пчел

Алгоритм пчел (в англоязычных статьях так же встречаются названия Artificial Bee Colony Algorithm и Bees Algorithm) является довольно молодым алгоритмом для нахождения глобальных экстремумов (максимумов или минимумов) сложных многомерных функций (будем называть эту функцию целевой функцией) [7].

Идея алгоритма взята, как уже можно догадаться из названия, у пчел, а именно из их поведения при поиске мест, где можно раздобыть как можно больше нектара. Сначала из улья вылетают в случайно направлении какое-то количество пчел-разведчиков, которые пытаются отыскать участки, где есть нектар. Через какое-то время пчелы возвращаются в улей и сообщают остальным, где и сколько они нашли нектара.

После этого на найденные участки отправляются другие пчелы, причем, чем больше на данном участке предполагается найти нектара, тем больше пчел летит в этом направлении. А разведчики опять улетают искать другие участки, после чего процесс повторяется. В окрестность n лучших участков отправляется N пчел, а в окрестность m выбранных участков отправляется M пчел, причем на каждый из лучших участков должно приходиться больше пчел, чем на каждый из выбранных участков. Можно сделать так, что чем больше значение целевой функции, тем большее количество пчел будет отправляться на соответствующий участок, а можно N и M сделать фиксированными величинами.

Нужно обратить внимание, что пчелы отправляется не в точности на то место, где пчелы-разведчики нашли перспективные и лучшие

места, а в их окрестность, более точно координаты определяются случайным образом. Кроме того, окрестность, которая определяет область, в которую может быть послана пчела, можно уменьшать по мере увеличения номера итерации, чтобы постепенно решение сходилось к самому "кончику"экстремума. Но если область уменьшать слишком быстро, то решение может застрять в локальном экстремуме. После того как пчелы были отправлены на лучшие и выбранные участки, можно отправить тех же пчел-разведчиков на другие случайные точки. После всех этих операций снова находятся n лучших и m выбранных участков, на этот раз среди всех пчел из роя, а не только среди разведчиков, и запоминается самое лучшее место на функции, значение больше которого пока не было найдено, это и будет промежуточным решением.

Затем алгоритм повторяется до тех пор, пока не сработает какой-нибудь из критериев останова. Критериев останова может быть несколько. Например, если мы знаем значение целевой функции в глобальном экстремуме, то можем повторять алгоритм до тех пор, пока функция не достигнет некоторого значения, близкого к желаемому. Если значение функции в экстремуме неизвестно, то можем повторять шаги алгоритма до тех пор, пока на протяжении какого-то достаточно большого количества итераций найденное решение не будет улучшаться. В качестве положения пчел-медоносов и пчел-скаутов в пространстве берутся претенденты на решение задачи, а перемещение пчел в пространстве означает изменение претендента на некоторое расстояние Хэмминга. Реализация алгоритма пчел выглядит следующим образом:

1) Инициализация начальной популяции претендентов. Для этого используется функция Gen_ind() аналогичная функции из генетического алгоритма;

2) Следующим этапом происходит оценивание претендентов. В качестве функции приспособленности также выступает сложность булевых функций. Оценивание производится с помощью функции AbsMin();

3) В некоторой окрестности вокруг лучших претендентов случайным образом генерируются новые претенденты.

4) Полученный массив также проходит процедуру оценивания.

5) На этом шаге происходит отбор лучших претендентов.

6) Отобранные лучшие претенденты помещаются в массив, оставшиеся позиции заполняются случайным образом.

Эта процедура продолжается до тех пор, пока не будет выполнен критерий останова. В данном случае — не будет достигнуто максимальное число итераций.

Алгоритм 2. Function BeesAlgorithm($f(x_1, \ldots, x_6)$, param1,. . . ,param6)
{Var /*определение переменных*/
Gen_ind(ScoutBees); // генерирование начальной популяции
AbsMin(ScoutBees); // оценка приспособленности пчел-скаутов
for i = 0 to n do {
for j = 0 to BeesNumber do
{ Bees[j] = ScoutBees[j%ScoutNumber];
mutation(Bees[j],BeesLength,Distance); // "рассеяние"пчел-медоносов вокруг пчел-скаутов}
AbsMin(Bees); // оценка приспособленности пчел-медоносов
GetBestBees(ScoutBees, Bees); //выбор лучших пчел }
Return res;}

Стратегия перебора итераций и размера популяции до тех пор, пока не будут найдены все точные значений функций либо значения, отличающиеся на единицу, оказалась неэффективной, так как для нахождения качественных параметров алгоритмов требовалось слишком много времени работы программы. Поэтому была выбрана стратегия перебора размера популяции, итераций и других параметров с некоторым шагом. Этот подход позволил сэкономить время и проанализировать качество работы каждого из алгоритмов.

Для алгоритма пчел были получены следующие результаты:

– при числе претендентов 20 (Рис. 3) только третий тип генерации первоначальной совокупности позволяет получить относительно приемлемые решения;

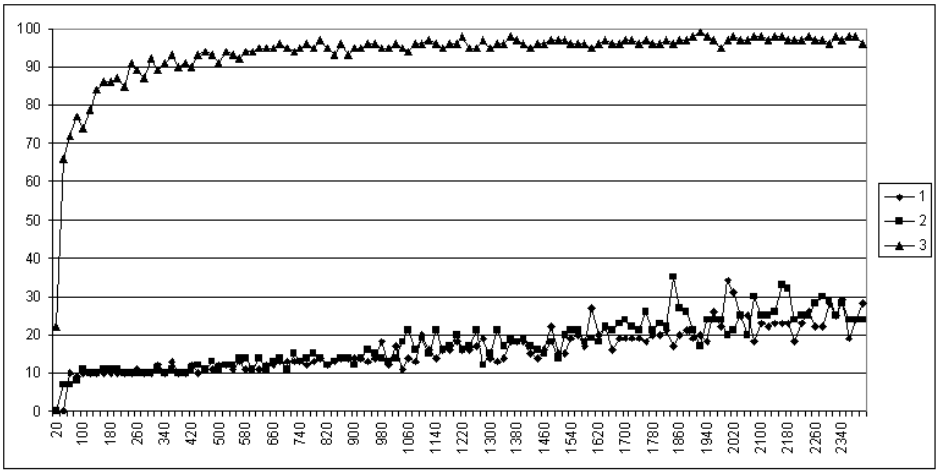

Рис. 3. График зависимости качества решений от числа итераций для алгоритма пчел при числе претендентов — 20.

– при числе претендентов 150 (Рис. 4) отчетливо видно, что первый и второй типы генерации позволяют получить лучший результат,

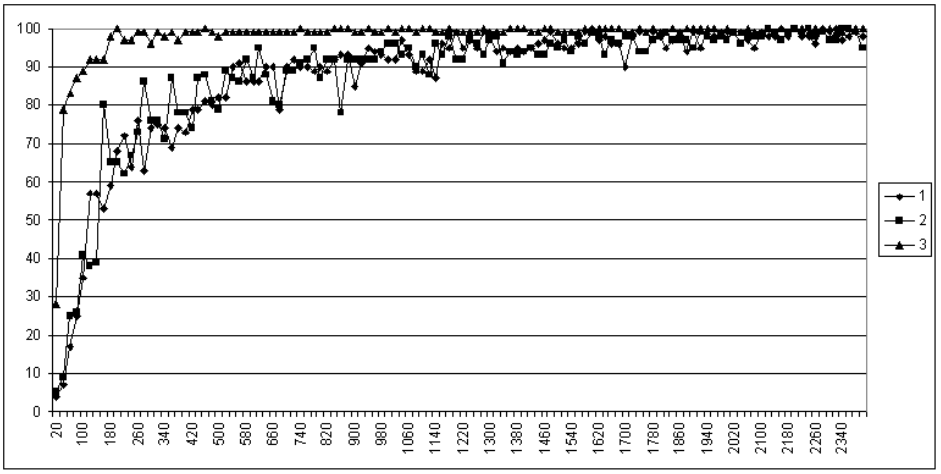

Рис. 4. График зависимости качества решений от числа итераций для алгоритма пчел при числе претендентов — 150.

чем при 20, однако, третий тип позволяет получать почти точные значения сложностей искомых функций;

- также возрастает качество решений, при увеличении расстояния «рассеяния» претендентов. На рис. 3 показан график зависимости качества получаемых решений от числа итераций, на котором параметром «рассеяния» является 1 бит;
- к увеличению качества работы алгоритма можно отнести выбор третьего типа генерации начальной совокупности пчел.

4. Алгоритм мух

Алгоритм мух был предложен в [4]. Основная идея этого алгоритма заключается в поведении мухи дрозофилы:

- муха находится в поисках пищи;
- передвижение мухи описывается случайным блужданием;
- муха ощущает запах потенциального местоположения пищи;
- после оценивания, если муху данное местоположение не удовлетворяет, то она перемещается в другое место;
- в процессе поиска муха обменивается информацией с другими мухами о наличии пищи и партнеров.

В качестве положения мухи в пространстве берутся претенденты на решение задачи, а перемещение мухи в пространстве означает изменение претендента на некоторое расстояние Хэмминга. Реализация алгоритма мух состоит из следующих этапов:

1) Инициализация начальной популяции претендентов. Для этого используется функция Gen_ind() аналогичная функции из генетического алгоритма;

2) Оценка приспособленности претендентов;

3) Выбор нескольких случайных точек в пространстве;

4) Сравнение этих точек с начальными положениями и выбор лучшего;

5) Перебор всех близлежащих точек, отличающихся от выбранной, максимально на некоторое число бит;

6) Перемещение в лучшую точку. Если эта точка не найдена — случайное блуждание.

Повторение этих шагов, пока не выполнен критерий останова.

Алгоритм 3. Function FlyAlgorithm($f(x_1 \dots x_6)$, param1,. . . ,param3)
{Var /*определение переменных*/
Gen_ind(Flies); // генерирование начальной популяции
AbsMin(Flies); // оценка приспособленности мух
for i = 0 to n do {
for j = 0 to FlyNumber do
{ Gen_ind(Flies[j]); //имитация «посещения» мухой случайных точек
AbsMin(Flies[j]);
GetMinFly(Flies[j]); // нахождение минимальной точки в некоторой окрестности}}
Return res; }

Для алгоритма мух были получены следующие результаты:
- при числе претендентов 10 (Рис. 5) ситуация схожа с алгоритмом пчел — третий тип генерации обеспечивает высокое качество работы алгоритма уже для небольшого числа итераций;
- при увеличении числа претендентов до 30 (Рис. 6) разница между типами генерации также нивелируется;
- существенно влияет количество бит, на которое претендент может "смещаться"в пространстве;

5. Алгоритм обезьян

На основе наблюдений за поведением обезьян, перемещающихся по деревьям в поисках еды, А. Мачерино и О. Сереф был предложен обезьяний алгоритм, предназначенный для решения задач глобальной оптимизации. Ветви деревьев представляются соседними выполнимыми

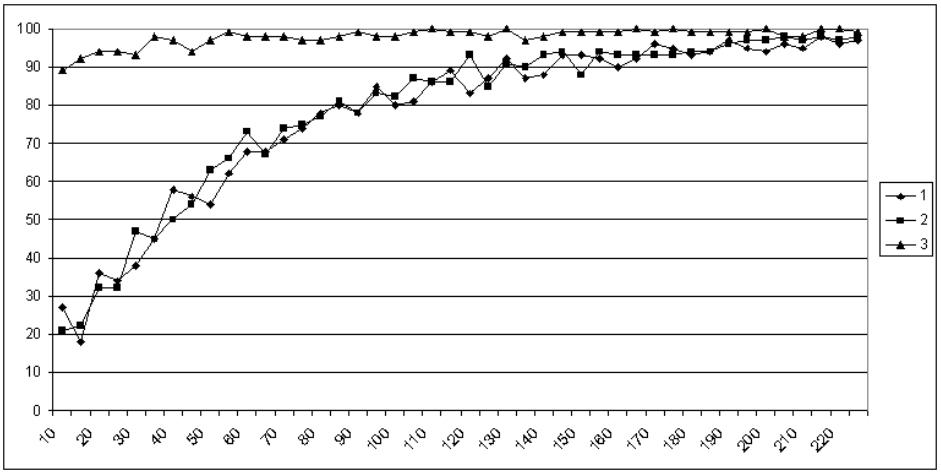

Рис. 5. График зависимости качества решений от числа итераций для алгоритма мух. Число претендентов — 10.

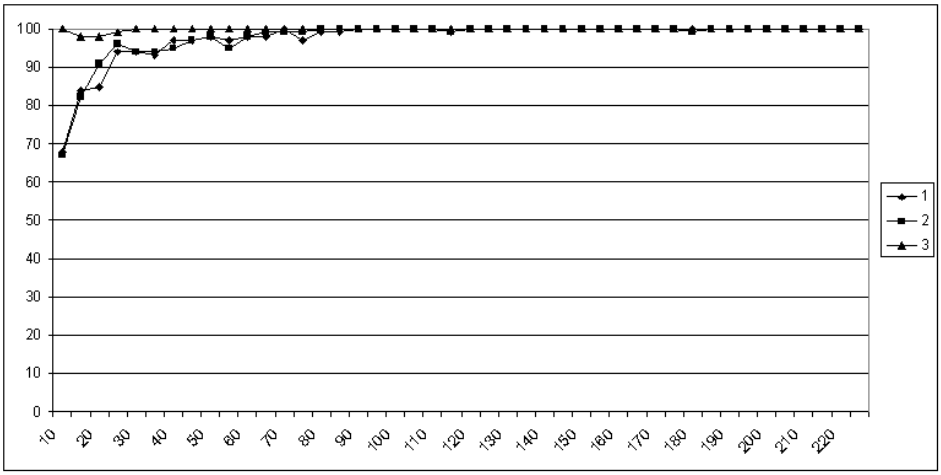

Рис. 6. График зависимости качества решений от числа итераций для алгоритма мух. Число претендентов — 30.

решениями определенной проблемы глобальной оптимизации. Обезьяна может перемещаться вверх-вниз по дереву и помечать пройденные ветки в попытках отыскания лучших решений.

Обезьяний алгоритм основывается на наборе параметров, которые определяют сходимость алгоритма. Высота деревьев — общее число веток, которые обезьяна может посетить от корня до вершины. Число путей дерева представляются количеством раз, за которое обезьяна может совершить восхождение по этому же дереву. Два других параметра связаны с памятью эвристики обезьяньего алгоритма. Чтобы избежать

локального минимума, найденные лучшие решения на каждом из деревьев заносятся в память. Фактически, каждый раз обезьяна прекращает подъем на дерево из-за того, что она исчерпала дозволенное количество путей восхождения и начинает подъем на другое дерево с другой начальной позиции. Лучшие решения запоминаются в памяти, поэтому обезьяна может выбрать одно или несколько решений в качестве начала нового дерева.

В качестве положения обезьян в пространстве берутся претенденты на решение задачи, а перемещение обезьяны в пространстве означает изменение претендента на некоторое расстояние Хэмминга. Для задачи минимизации полиномиального представления булевых функций алгоритм обезьян может иметь следующий вид:

1) инициализация начальной популяции претендентов. Для этого используется функция Gen_ind() аналогичная функции из генетического алгоритма;

2) оценка приспособленности;

3) получение двух соседних точек для восхождения, путем изменения нескольких бит на противоположные и их оценка;

4) переход в эту точку, если приспособленность точки лучше. Эта процедура повторяется до тех пор, пока не будут найдены субоптимальные решения;

5) выбор места для "прыжка— изменение нескольких бит в претенденте. Переход в эту точку после нескольких попыток, если нашлось лучшее решение;

6) выбор новой точки для восхождения.

Повторяется до выполнения критерия останова.

Алгоритм 4.

```
Function MonkeyAlgorithm(f(x_1 ... x_6), param1,...,param5)
{Var /*определение переменных*/
Gen_ind(Monkeys); // генерирование начальной популяции
AbsMin(Monkeys); // оценка приспособленности мух
for i = 0 to n do {
for j = 0 to MonkeysNumber do
{ ClimbProcess();
WatchJumpProcess():
SomersaultProcess();
AbsMin(Monkeys);} }
Return res; }
```

Для алгоритма обезьян были получены следующие результаты:

– также, как и в предыдущих двух случаях, третий тип генерации первоначальной совокупности при числе претендентов 10 позволяет при числе итераций больше 50 (Рис. 7) получать практически точные значения;

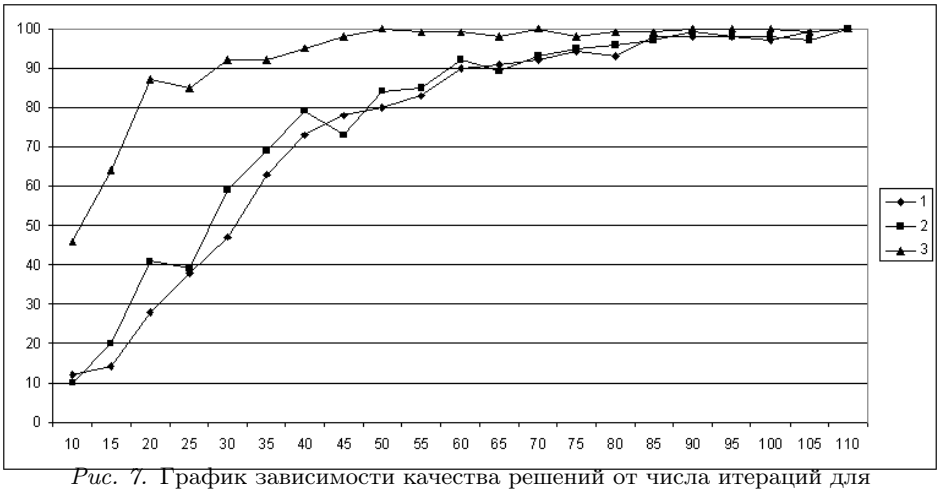

Рис. 7. График зависимости качества решений от числа итераций для алгоритма обезьян. Число претендентов — 10.

– при увеличении числа претендентов до 100 сходимость алгоритма происходит быстрее. При числе итераций свыше 70 (Рис. 8) все три типа генерации демонстрируют практически одинаковую эффективность;
– тип генерации, а также число бит для «прыжка» обезьяны, число «прыжков» и число бит, для новых подъемов обезьяны существенного влияния не оказывает.

6. Результаты вычислительных экспериментов

Все представленные алгоритмы были реализованы с использованием языка C++. По проведенным экспериментальным запускам алгоритмов можно получены следующие результаты — наиболее подходящими для поиска минимального полиномиального представления булевых функций 6-ти переменных можно использовать следующие алгоритмы:

– генетический алгоритм. Параметры алгоритма: 100–120 итераций, 30–35 мутаций, 4 мутирующих гена, 150–160 — размер популяции, тип генерации — 1, тип кроссовера — двухточечный, равномерный закон распределения, среднее время выполнения — 47 секунд;
– алгоритм пчел. Параметры алгоритма: 150–160 итераций, 150–160 пчел-скаутов, 300–320 пчел-медоносов, 75–85 лучших пчел, 4 бита

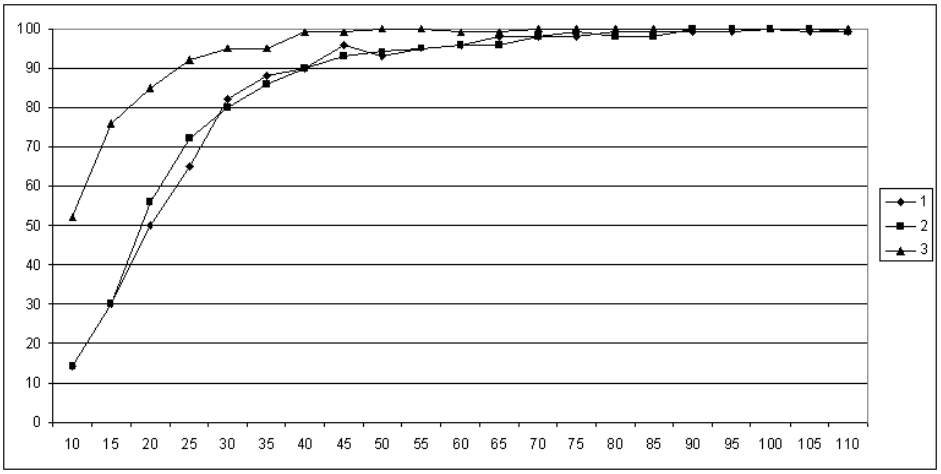

Рис. 8. График зависимости качества решений от числа итераций для алгоритма обезьян. Число претендентов — 100.

расстояния рассеяния пчел, тип генерации начальной совокупности — 3, среднее время выполнения — 68 секунд;

- алгоритм мух. Параметры алгоритма: число мух — 30–40, число итераций — 50–60, тип генерации начальной совокупности — 3, число шагов — 6, область видимости мухи - 3 бита, среднее время выполнения — 175 секунд;
- алгоритм обезьян. Параметры алгоритма: число обезьян — 100–120, число итераций — 200–220, число бит для изменения — 4, число бит для «прыжка» — 7, число возможных «прыжков» - 10–12, среднее время выполнения — 89 секунд.

Результаты, полученные в ходе исследования, позволили осуществить адаптацию реализованных алгоритмов для минимизации полиномиального представления булевых функций 7 переменных. На сегодня для большей их части невозможно получить точное значение сложности, можно получить лишь оценочные значения. Тестовые запуски адаптированных алгоритмов для функций 7 переменных с максимальной сложностью 24, которая была получена оценочным путем, получили сложность 29.

Список литературы

1. Гилл Ф. Практическая оптимизация : пер. с англ. / Ф. Гилл, У. Мюррей, М. Райт. – М. : Мир, 1985.
2. Ильин Б. П. Введение в генетические алгоритмы (ГА) / Б. П. Ильин // Применение математических методов и информационных технологий в экономике. – Иркутск : Изд-во БГУЭП, 2008.– С. 63–71.

3. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский ; пер. с польск. И. Д. Рудинского. – М. : Горячая линия - Телеком, 2007. – 452 с.: ил.
4. Abidin Z. Z. A Survey: Animal-Inspired Metaheuristic Algorithms / Z. Z. Abidin, M. R. Arshad, U. K. Ngah. – Nibong Tebal : School of Electrical and Electronic Engineering, 2009.
5. Bonabeau E. Swarm Intelligence: From Natural to Artificial Systems / E. Bonabeau, M. Dorigo, G. Theraulaz. – Oxford : Oxford University Press, 1999. – 320 c.
6. Gaidukov A. Algorithm to derive minimum ESOPs for 6-variable functions / A. Gaidukov // Proceedings of the 5th International Workshop on Boolean Problems 2002, Freiberg, Germany, Sept. 19–20, 2002. – P. 141–148.
7. The Bees Algorithm - A Novel Tool for Complex Optimisation Problems / D. T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri , S. Rahim , M. Zaidi. – Cardiff : Cardiff University, 2006. – P. 454–459.

B.P. Ilyin

Evolutionary optimization algorithms for minimization problem of boolean functions

Abstract. This paper contains the review of the evolutionary optimization algorithms for minimization of boolean functions.

Keywords: Boolean functions, optimization, evolutionary algorithms, polynomial representation of Boolean functions

Ильин Борис Петрович, старший преподаватель, Кафедра информатики и кибернетики, Байкальский Государственный Университет Экономики и Права, 664003, Иркутск, ул. Ленина, 11 тел.: (3952)242846 (rays.irk@gmail.com)

Ilyin Boris, Baikal State University of Economics and Law, 1, Lenin St., Irkutsk, 664003 assistant, Phone: (3952)242846 (rays.irk@gmail.com)