УДК 004.051 **Информационные технологии**

Безгачев Ф. В., *старший преподаватель кафедры информационно-правовых дисциплин и специальной техники Сибирский юридический институт МВД России, Россия, г.Красноярск*

Галушин П. В., *кандидат технических наук, доцент кафедры информационно-правовых дисциплин и специальной техники Сибирский юридический институт МВД России, Россия, г.Красноярск*

Рудакова Е. Н., *преподаватель кафедры административного права и административной деятельности ОВД Сибирский юридический институт МВД России, Россия, г.Красноярск*

ЭФФЕКТИВНАЯ РЕАЛИЗАЦИЯ ИНИЦИАЛИЗАЦИИ И МУТАЦИИ В ГЕНЕТИЧЕСКОМ АЛГОРИТМЕ ПСЕВДО-БУЛЕВОЙ ОПТИМИЗАЦИИ

Аннотация: В данной статье предлагается реализация алгоритмов инициализации начальной популяции и мутации для генетического алгоритма псевдо-булевой оптимизации, превосходящая по быстродействию прямолинейные реализации. Использование этих реализаций способно уменьшить временные и энергетические затраты на решение задач псевдо-булевой оптимизации. Предложенный алгоритм генерации случайных чисел, имеющих распределение Бернулли с параметром p=0,5, может быть использован и в иных методах псевдо-булевой оптимизации, в которых требуется порождение случайных булевых векторов.

Ключевые слова: псевдо-булевая оптимизация, генетический алгоритм, псевдо-случайные числа, распределение Бернулли, мутация.

Annotation: This article proposes the implementation of initialization algorithms for the initial population and mutations for the genetic algorithm of pseudo-Boolean optimization, which is faster than naive implementations. Usage of

these implementations can reduce the time and energy costs of solving pseudo-Boolean optimization problems. The proposed algorithm for generating random numbers with a Bernoulli distribution with parameter p = 0.5 can be used in other pseudo-Boolean optimization methods that require the generation of random Boolean vectors.

Keywords: pseudo-boolean optimization, genetic algorithm, pseudo-random numbers, Bernoulli distribution, mutation.

Задачи оптимизации постоянно возникают в практической деятельности человека. Действительно, если существует возможность выбирать параметры технической или экономической системы, то разумно сделать это самым лучшим (в смысле некоторого критерия) образом. Всё более актуальными становятся сложные задачи оптимизации, то есть задачи, в которых критерий не обладает свойствами, позволяющими эффективно применять классические методы оптимизации: гладкостью, выпуклостью.

Для решения сложных задач оптимизации, как правило, используются стохастические методы. Многие методы оптимизации, будучи сами по себе регулярными, требуют задания начального приближения к решению, которое выбирается случайным образом.

Достаточно широкое распространение в последнее время получили методы псевдо-булевой оптимизации, то есть методы оптимизации вещественных функций булевых переменных, например, генетические алгоритмы [1]. Они применяются и для решения задач оптимизации с дискретными и вещественными переменными с использованием бинаризации. Вещественные переменные дискретизируются с заданной точностью, а дискретные представляются в виде двоичной записи целых чисел.

Отметим, что в работах, посвященных эволюционным методам псевдо-булевой оптимизации, редко обсуждаются вопросы их эффективной программной реализации. Поэтому многие реализации этих методов оказываются субоптимальными. Рассмотрим возможности более эффективной

реализации инициализации и мутации в генетическом алгоритме псевдо-булевой оптимизации.

Особи начальной популяции в генетическом алгоритме получают, как правило, присваивая каждой булевой переменной случайным образом с одинаковой вероятностью ноль или единицу. То есть рассматриваемые булевы переменные имеют распределение Бернулли [2] с параметром 0,5.

В общем случае случайные числа с распределением Бернулли с параметром p получают с использованием следующего алгоритма:

1. Порождается целое случайное число R равномерно-распределённое на интервале $[0,2^L\text{-}1]$. Соответствующие процедуры реализованы в стандартных библиотеках большинства современных языков программирования.
2. Если $R < p \cdot 2^L$, то результат равен единице, иначе – нулю.

Можно показать, что биты двоичного представления целого числа имеют распределение Бернулли с параметром 0,5 [3, с. 39]. Поэтому описанный алгоритм, являясь универсальным, оказывается крайне неэффективным в интересующем нас случае p равно 0,5: для получения одного случайного бита используются L случайных бит.

Описанное выше свойство битов равномерно-распределённых целых чисел можно использовать для более эффективной инициализации битов. Будем каждое случайное число R использовать для получения не одного, а L бит. То есть для получения случайного бита будем брать очередной бит числа R (начиная, например, с самого младшего двоичного разряда), а новое число R будем порождать только после исчерпания всех L битов.

Для подтверждения эффективности описанного метода порождения равномерно распределённых битов были проведены численные эксперименты. Для каждого количества битов порождалось 10000 случайных векторов. Тестовая программа была написана на языке программирования C++, использовался компилятор GCC [4] версии 7.2.0 с уровнем оптимизации –O2. Для тестирования использовался ПК с процессором AMD Phenom™ X4 955 и 8

ГБ оперативной памяти. В таблице 1 приведено время (в секундах) порождения случайных битов для общего метода и предложенного в данной работе.

Таблица 1 - Время порождения случайных битов

Количество бит	10	20	30	40	50	60	70	80	90	100
Общий метод	2	5	8	10	13	16	18	21	24	27
Предложенный специализированный метод для p=0,5	1	2	3	5	6	7	8	10	11	12

Из таблицы видно, что быстродействие описанного метода примерно в два раза выше, чем общего метода порождения случайных чисел с распределением Бернулли.

Перейдём теперь к рассмотрению алгоритма мутации. Напомним, что оператор мутации применяется к каждому решению на каждом поколении генетического алгоритма для увеличения разнообразия решения и избегания скатывания в локальный минимум. Обозначим вероятность точечной мутации (то есть мутации одного гена) как p_m, введём также обозначение для дополняющего события (ген не изменяется в ходе мутации): $q_m = 1 - p_m$.

Прямолинейная реализация оператора мутации языке программирования C++ выглядит примерно следующим образом:

```
void mutation(Gen * p, size_t n, double p_m)
{
    for(int i = 0; i < n; ++ i)
        if( rnd() < p_m )
            p[i] = !p[i];
}
```

Здесь p – указатель на блок памяти, в котором размещено решение, n – его длина, а rnd – функция, генерирующая случайные вещественные числа, равномерно распределённые на интервале [0, 1).

Имеется несколько путей усовершенствования данного кода. Во-первых, можно избавиться от условной инструкции в теле цикла (Как известно, условные инструкции могут замедлять работу современных микропроцессоров с конвейерной архитектурой [5, с. 843]). Значение p[i] после выполнения цикла зависит от двух булевых величин p[i] до начала выполнения тела цикла и значения выражения rnd() < p_m. Составим таблицу возможных ситуаций:

Таблица 2 - Возможные ситуации

p[i] старое	rnd() < p_m	p[i] новое
0	0	0
1	0	1
0	1	1
1	1	0

Нетрудно понять, что новое p[i] равно нулю тогда и только тогда, когда оба аргумента имеют одинаковое значение, такая функция называется «исключающим ИЛИ». С учётом этого факта мы можем переписать функцию мутации следующим образом:

```
void mutation(bool * p, size_t n, double p_m)
{
    for(int i = 0; i < n; ++ i)
        p[i] ^= ( rnd() < p_m );
}
```

Эта реализация выполняет столько же шагов цикла, что и исходная прямолинейная, но лучше подходит для современных процессоров с конвейерной архитектурой, так как не включает условных инструкций.

Однако существует и другое направление усовершенствования данного алгоритма, основанное на том факте, что вероятность мутации, как правило, очень мала. Это означает, что в большинство генов не изменится в ходе процесса мутации и большинство случайных чисел генерируется «вхолостую».

Вместо того, чтобы случайным образом определять будет ли инвертирован данный ген, можно случайным образом определять, сколько генов подряд будет пропущено перед тем как ген изменится.

Вероятность того, что будет пропущено ровно $k - 1$ генов, а k-ый будет изменён, равна, по теореме умножения из теории вероятностей:

$$P_k = q_m^{k-1} p_m = (1 - p_m)^{k-1} p_m$$

Данное дискретное распределение называется «геометрическим распределением». Существует достаточно эффективный способ получения случайных чисел с данным распределением из равномерно распределенных случайных чисел, основанный на достаточно простом преобразовании, не включающем какие-либо условные переходы [6, с. 161].

Используя данный подход, мы можем написать следующую реализацию оператора мутации:

```
size_t geometry(double p)
{
	return size_t(log(1 – rnd() ) / log(1 – p) + 1);
}

void mutation(bool * p, size_t n, double p_m)
{
	for(int i = geometry(p) - 1; i < n; i += geometry(p) - 1)
		p[i] = !p[i];
}
```

Проанализировав данный программный код, мы можем заметить, что данная реализация выполняет меньше шагов в цикле, чем прямолинейная реализация, описанная выше, так как за один шаг может продвинуться более чем на один индекс. Кроме того, тело цикла не содержит условных инструкций, то есть данная реализация не уступает предыдущей (на основе «исключающего или») в плане пригодности для архитектуры современных процессоров. Ещё одно её преимущество – использование меньшего количества случайных чисел,

порождение которых может быть затратной операцией. Использование в этом алгоритме функции логарифм не приводит к снижению быстродействия, так как эта функция (наряду со многими другими) реализована в большинстве современных процессоров общего назначения на аппаратном уровне.

Заметим также, что реализация на основе геометрического распределения хуже подходит для распараллеливания, так как в ней величина шага определяется генератором псевдо-случайных чисел. С другой стороны, порождение псевдо-случайных чисел плохо поддаётся распараллеливанию. Поэтому в качестве направления дальнейших исследований можно указать сравнение быстродействия распараллеленных вариантов двух предложенных реализаций на многоядерных процессорах.

Таким образом, в данной работе была показана эффективность специализированного метода порождения равномерно распределённых булевых переменных по сравнению с общим методом порождения случайных чисел с распределением Бернулли. Также были предложены две реализации оператора мутации генетического алгоритма, лучше подходящие для компьютеров с современной архитектурой.

Эти результаты могут быть использованы для повышения быстродействия программного обеспечения, реализующего и/или использующего методы оптимизации псевдо-булевой оптимизации, в частности – генетический алгоритм.

Библиографический список:

1. Goldberg, D. E. Genetic algorithms in search, optimization, and machine learning. Reading, MA: Addison-Wesley, 1989.

2. Johnson N.L., Kotz S., Kemp A. Univariate Discrete Distributions (2nd Edition). – Wiley, 1993.

3. Булинский А.В., Ширяев А.Н. Теория случайных процессов. – М. : ФИЗМАТЛИТ, 2003.

4. GCC, the GNU Compiler Collection [Электронный ресурс]. – Режим доступа: gcc.gnu.org (дата обращения: 02.12.2019).

5. Ахо А.В., Лам М.С., Сети Р., Ульман Д.Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд. : пер. с англ. – М.: ООО «И. Д. Вильямс», 2008.

6. Кнут Д.Э. Искусство программирования, том 2. Получисленные методы, 3-е изд. : пер. с англ. – М. : ООО «И. Д. Вильямс», 2007. – 832 с. : ил..