2 Методы с одним состоянием

В градиентных методах оптимизации делается существенное допущение о том, что можно вычислить первую (или даже вторую) производную. Это очень *существенное* допущение. Оно разумно при оптимизации хорошо понятной математической функции известной формы. Однако в большинстве случаев вычислить градиент не удается, поскольку *неизвестен даже вид и свойства функции.* Можно только изменять значения входных параметров функции, вычислять ее значение и оценивать его.

Представьте, к примеру, что имеется сиулятор гуманоидного робота, и необходимо отыскать цикл оптимальных операций с привязкой ко времени, которые позволили бы роботу шагать вперед без падения. При этом имеется n различных операций, а потенциальными решениями являются строки произвольной длины, содержащие эти операции. Эту строку можно подать на вход симулятора и получить через некоторое время значение качества (насколько далеко вперед прошел робот, прежде чем упасть). Как найти хорошее решение?

В подобных случаях оптимизационная **задача** представлена черным ящиком (здесь симулятор робота). В ящике имеется слот для тестирования **потенциального решения** (***candidate solution***) (здесь, строки, содержащей операции для робота) задачи. После нажатия большой красной кнопки можно получить **качество** (***quality***) этого решения. При этом нет никаких предположений относительно характера поверхности функции качества. Более того, это потенциальное решение не обязательно является числовым вектором: оно может быть графом, деревом, набором правил или строкой операций для робота! Т.е. всем, что может иметь отношение к задаче.

Для оптимизации потенциального решения данной задачи необходимо наличие четырех составляющих:

- Доступность одного или нескольких начальных потенциальных решений. Известно как **процедура инициализации** (***initialization procedure***).
- Возможность оценки качества потенциального решения. Известно как **процедура оценивания** (***assessment procedure***).
- Копирование существующих решений.
- Улучшение потенциального решения, которое создает *рандомизированное слегка измененное* (*randomly slightly different*) потенциальное решение. Данная операция вместе с операцией копирования называются **процедурой модификации** (***modification procedure***).

В добавление к этим операциям метаэвристический алгоритм обычно также осуществляет **процедуру селекции** (***selection procedure***), которая определяет, какие потенциальные решения останутся, а какие необходимо отбросить в результате анализа пространства возможных решений задачи.

2.1 Локальный поиск

Начнем с рассмотрения простого метода — **локального поиска** (***Hill-Climbing***). Данный метод напоминает градиентный подъем, однако для его использования нет необходимости в определении величины градиента или даже его направления: необходимо только тщательно проверять новые потенциальные решения в окрестности уже существующего, и принимать их, если они лучше текущего. Это позволяет продвигаться к экстремуму, до тех пор, пока не будет достигнут локальный оптимум.

Отметим сильное сходство между локальным поиском и градиентным подъемом. Единственное различие заключается в том, что локальный поиск использует более общую операцию операцию

[0]Перевод раздела из книги Luke S. Essentials of Metaheuristics. A Set of Undergraduate Lecture Notes. Zeroth Edition. Online Version 0.11. September, 2010 (`http://cs.gmu.edu/~sean/book/metaheuristics/`). Перевел – Юрий Цой, 2010-2011 гг. Любые замечания, касающиеся перевода, просьба присылать по адресу yurytsoy@gmail.com
Данный текст доступен по адресу: `http://qai.narod.ru/GA/meta-heuristics_2.pdf`

Алгоритм 4 Локальный поиск

```
1: S ← некоторое начальное потенциальное решение {Процедура инициализации}

2: repeat
3:    R ← Tweak (Copy (S)) {Процедура модификации}
      {Процедуры оценивания и селекции}
4:    if Quality(R) > Quality(S) then
5:       S ← R
6:    end if
7: until S — идеальное решение или закончилось время поиска
8: return S
```

Tweak вместо стохастического (частично случайного) подхода к поиску лучшего потенциального решения. Иногда таким образом удается найти решения хуже, а иногда и лучше.

Можно сделать этот алгоритм более агрессивным: используя n вариантов изменений потенциального решения сразу, и оставляя только лучший из них. Модифицированный алгоритм называется **локальным поиском с наискорейшим подъемом** (***Steepest Ascent Hill-Climbing***), поскольку рассматривая множество потенциальных решений в окрестности текущего и выбирая наилучшее, мы тем самым находим градиент и продвигаемся вдоль него.

Алгоритм 5 Локальный поиск с наискорейшим подъемом

```
1: n ← требуемое количество модификаций для поиска градиента

2: S ← некоторое начальное потенциальное решение

3: repeat
4:    R ← Tweak (Copy (S))
5:    for n − 1 раз do
6:       W ← Tweak (Copy (S))
7:       if Quality(W) > Quality(R) then
8:          R ← W
9:       end if
10:   end for
11:   if Quality(R) > Quality(S) then
12:      S ← R
13:   end if
14: until S — идеальное решение или закончилось время поиска
15: return S
```

Популярным вариантом, который я называю локальным поиском с наискорейшим подъемом и заменой (***Steepest Ascent Hill-Climbing with Replacement***), является отсутствие сравнения R и S: вместо этого просто заменяем S на R. Естественно в этом случае возникает риск потери лучшего найденного решения, поэтому алгоритм можно дополнить, сохраняя лучшее найденное в текущем заупске решение в отдельной переменной $Best$. В конце запуска возвращаем $Best$. Практически во всех последующих алгоритмах мы также будем использовать подобный прием, так что привыкайте!

2.1.1 Назначение функции Tweak

Инициализация, функции Copy, Tweak и (в меньшей степени) функция оценивания приспособленности определяют **представление** (***representation***) потенциального решения. Вместе они задают способ кодирования решения и операции над ним.

Как может выглядеть решение? Это может быть вектор; список произвольной длины для объектов; неупорядоченный набор либо множество объектов; дерево; граф. Или любая комбинация вышеперечисленного, т.е. все что подходит для рассматриваемой задачи. И, если удастся создать четыре приведенных выше функции, способные обрабатывать выбранное представление данных, — то вы при деле.

Простым и часто используемым вариантом представления потенциальных решений, которое будет

Алгоритм 6 Локальный поиск с наискорейшим подъемом и заменой

$n \leftarrow$ требуемое количество модификаций для поиска градиента

$S \leftarrow$ некоторое начальное потенциальное решение
$Best \leftarrow S$

repeat
  R $\leftarrow$ Tweak (Copy (S))
  for $n-1$ раз **do**
    $W \leftarrow$ Tweak (Copy (S))
    if Quality(W) $>$ Quality(R) **then**
      $R \leftarrow W$
    end if
  end for
  $S \leftarrow R$
  if Quality(S) $>$ Quality($Best$) **then**
    $Best \leftarrow S$
  end if
until $Best$ — идеальное решение или закончилось время поиска
return $Best$

рассматриваться далее, совпадает с используемым в градиентном методе: **вещественный вектор фиксированной длины** (***fixed-length vector of real-valued numbers***). Легко создать такой случайный вектор путем последовательной генерации случайных чисел в заданном диапазоне. В случае включения границ min и max и использования вектора длины l, можно использовать следующий алгоритм:

Алгоритм 7 Генерация случайного вещественного вектора

$min \leftarrow$ минимальное допустимое значение элемента вектора
$max \leftarrow$ максимальное допустимое значение элемента вектора

$\overrightarrow{v} \leftarrow$ новый вектор $\langle v_1, v_2, \ldots v_l \rangle$
for $i = 1$ до l **do**
  $v_i \leftarrow$ случайное число равномерно распределенное в закрытом интервале от min до max
end for
return $\overrightarrow{v}$

Операция Tweak над вектором может (помимо прочего) выражаться в прибавлении небольшого случайного шума к каждому элементу. Для текущего определения данной операции предположим, что **на данный момент** шум не превышает некоторого малого уровня. Ниже приведен пример, как можно прибавить ограниченную, равномерно распределенную случайную составляющую к вектору. Для каждой позиции в векторе шум прибавляется в случае, если для данной вероятности p при подбрасывании монеты выпадет «орел». В большинстве случаев $p = 1$.

Появился параметр, который можно регулировать — r, задающий диапазон границ в Tweak. Если этот диапазон мал, то локальный поиск будет продвигаться к ближайшему локальному пику и не сможет перейти на соседний пик, поскольку границы не позволяют делать длинные переходы. Оказавшись на вершине пика, все последующие переходы будут вести к худшим позициям, поэтому алгоритм сойдется. Кроме этого, скорость продвижения к пику также ограничена рассматриваемым диапазоном. С другой стороны, если диапазон слишком велик, то локальный поиск будет много «метаться» из стороны в сторону. Существенно, что, оказавшись недалеко от пика, алгоритм будет плохо сходиться к самой вершине, т.к. большинство переходов приведут к «перепрыгиванию» пика.

Таким образом, малый диапазон приводит к медленному продвижению и сходимости к локальному оптимуму, а большой — к большим скачкам и плохой сходимости к пику. Заметим, что роль этого параметра сходна с ролью α для градиентного подъема. Данный параметр является одним из способов контроля соотношения **исследование — эксплуатация** (***Exploration versus Exploitation***) в рамках локального поиска. Алгоритмы оптимизации, использующие преимущественно локальные улучшения, *эксплуатируют* локальный градиент, а алгоритмы, которые чаще используют случай-

Алгоритм 8 Ограниченная равномерная конволюция

$\overrightarrow{v} \leftarrow$ вектор $\langle v_1, v_2, \ldots v_l \rangle$ для конволюции
$p \leftarrow$ вероятность прибавить шум к элементу вектора $\{$Часто $p = 1\}$
$r \leftarrow$ половина диапазона значений шума.
$min \leftarrow$ минимальное допустимое значение элемента вектора
$max \leftarrow$ максимальное допустимое значение элемента вектора

for $i = 1$ до l **do**
  if $p \geq$ случайное число равномерное распределенное в $[0.0, 1.0]$ **then**
    repeat
      $n \leftarrow$ случайное число равномерно распределенное в закрытом интервале от $-r$ до $+r$
    until $min \leq v_i + n \leq max$
  end if
  $v_i = v_i + n$
end for
return $\overrightarrow{v}$

ные перемещения в пространстве поиска, считают *исследующими*. В качестве рекомендации: *более* целесообразным является использование сильно эксплуатирующего алгоритма (он быстрее), однако чем «хуже» пространство поиска, тем большая возникает необходимость в исследующем алгоритме.

2.2 Алгоритмы глобальной оптимизации с одним состоянием

Алгоритмом глобальной оптимизации (***global optimization algorithm***) является такой алгоритм, который в случае достаточно продолжительной работы рано или поздно найдет глобальный оптимум. Практически всегда это достигается путем перебора всех возможных точек в пространстве поиска. Алгоритмы с одним состоянием, которые мы встречали ранее не могут гарантировать нахождение глобального оптимума. Это происходит в силу (временного) задания операции Tweak: «осуществление небольших, ограниченных и случайных изменений». Результатом Tweak не могут стать значительные изменения. Если алгоритм окажется в достаточно большом локальном оптимуме, то усилиями Tweak выйти из него будет невозможно. Поэтому те алгоритмы, которые были описаны ранее являются **алгоритмами локальной оптимизации** (***local optimization algorithms***).

Существует множество способов, чтобы создать алгоритм глобальной оптимизации. Начнем с простейшего — **Случайного поиска** (***Random Search***).

Алгоритм 9 Случайный поиск

$Best \leftarrow$ некоторое начальное решение

repeat
  $S \leftarrow$ случайное потенциальное решение
  if Качество(S) $>$ Качество($Best$) **then**
    $Best \leftarrow S$
  end if
until в $Best$ записано идеальное решение, или закончилось время поиска
return $Best$

Алгоритм случайного поиска является наиболее исследующим (и применим к глобальной оптимизации). С другой стороны, алгоритм локального поиска (алгоритм 4), использующий Tweak, делающую лишь небольшие изменения и никогда не совершающую крупных, можно считать наиболее эксплуатирующим (и являющимся локальным). Однако всегда существуют способы получить разумную эксплуатационную составляющую в глобальном алгоритме оптимизации. Примером является популярный метод, называемый **Локальный поиск со случайными перезапусками** (***Hill-Climbing with Random Restarts***), представляющий нечто среднее между двумя описанными алгоритмами. В нем в течение случайного количества шагов выполняется локальный поиск. По завершению производится перезапуск алгоритма со случайной начальной точкой и для другого случайного по продолжительности интервала времени. И так далее. Псевдокод:

Алгоритм 10 Локальный поиск со случайными перезапусками

```
T ← распределение возможных временных интервалов

S ← некоторое начальное случайное потенциальное решение
Best ← S

repeat
   time ← случайный момент времени в недалеком будущем
   repeat
      R ← Tweak(Copy(S))
      if Качество(R) > Качество(S) then
         S ← R
      end if
   until в S записано идеальное решение, или прошло время time, или закончилось общее время
   поиска
   if Качество(S) > Качество(Best) then
      Best ← S
   end if
   S ← некоторое случайное потенциальное решение
until в Best записано идеальное решение, или закончилось общее время поиска
return Best
```

Если случайно генерируемые временные интервалы будут слишком длительными, то алгоритм просто не будет отличаться от локального поиска. Аналогично, если интервалы очень короткие, то алгоритм по сути совпадет со случайным поиском (каждый раз будет осуществляться сброс на новую случайную начальную точка поиска). Интервалы средней продолжительности позволят более полно использовать оба подхода. Неплохо, правда?

Все зависит от ситуации. Рассмотрим рис .6. На первом рисунке, обозначенном как *Unimodal* (*унимодальный*), показан случай, когда локальный поиск приведет к решению, близкому к оптимальному, а случайный поиск будет являться очень неудачным выбором. Однако для рисунка с меткой *Noisy* (*зашумленный*), локальный поиск уже даст весьма плохие результаты, а вот результаты случайного поиска могут быть не хуже результатов оптимизации, сделанной человеком (который заранее не имеет никакой информации о функции). Разница в том, что для унимодального случая существует сильная зависимость между расстоянием (вдоль оси Ox) двух потенциальных решений и их относительным качеством: близкие решения будут как правило иметь схожее качество, а отдаленные соответственно не будут иметь выраженной связи. Для зашумленного случая подобная зависимость отсутствует: даже близкие решения могут иметь очень сильно отличающееся качество. Часто говорят, что для эффективной работы локального поиска необходимо свойство **гладкости** (***smoothness***).

Однако одного этого еще не достаточно. Рассмотрим функцию, помеченную *Needle in a Haystack* (*иголка в стоге сена*), для которой эффективным является только случайный поиск, а локальный поиск малоприменим. В чем различие между этой функцией и унимодальной? Ведь функция *Needle in a Haystack* является очень даже гладкой. Для нормльной работы локального поиска необходим **информативный градиент** (***informative gradient***), который должен вести в направлении наилучшего решения. Но можно придумать функцию с весьма *неинформативным* (*uninformative*) градиентом, при котором локальный поиск будет работать очень плохо! Для функции на рисунке *Deceptive* (*обманчивый*), локальному поиску не просто нелегко прийти к решению, но он вообще активно *уводится в противоположном направлении*.

Таким образом, существуют задачи, в которых жадный локальный поиск работает очень хорошо, а также задачи, где лучше использовать большие, практически случайные, изменения. В глобальных алгоритмах оптимизации приходится учитывать эти особенности, и мы уже сталкивались с аналогичной проблемой: **эксплуатация против исследования**. И снова, в качестве полезной эвристики, важно использовать сильно эксплуатирующий алгоритм (он быстрее), но чем «хуже» пространство поиска, тем больше вероятность, что в нем эффективнее будет работать исследующий алгоритм.

Ниже перечислены несколько способов создания алгоритма глобального поиска, а также подходы к настройке соотношения между исследованием и эксплуатацией в этих алгоритмах:

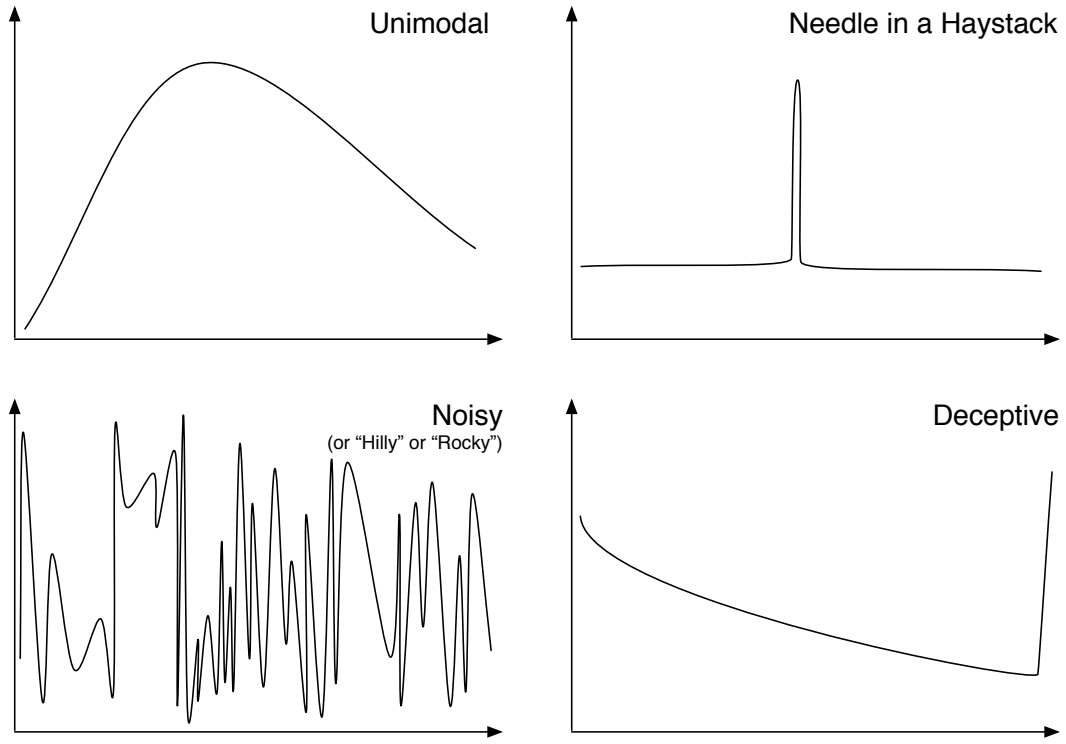

Рис. .6: 4 примера функции качества

- **Настройка процедуры модификации**. Операция Tweak время от времени должна совершать большие непредсказуемые изменения.

  Чем обеспечивается глобальный поиск. Если запустить данный алгоритм на достаточно большое время, то такая случайность его работы приведет к перебору всех возможных решений.

  Исследование против эксплуатации. Чем больше значительных случайных изменений, тем сильнее исследование.

- **Настройка процедуры селекции**. Измените алгоритм таким образом, чтобы хотя бы некоторое время поиск шел в сторону ухудшения.

  Чем обеспечивается глобальный поиск. При достаточно большой длительности запуска в результате многократных «спусков с холмов» можно когда-нибудь найти нужный глобальный экстремум.

  Исследование против эксплуатации. Чем чаще производится движение «вниз», тем сильнее исследование.

- **Проба новых решений**. Совершайте перезапуски с новой начальной точки.

  Чем обеспечивается глобальный поиск. Перепробовав достаточное количество начальных точек, алгоритм придет к окрестности глобального экстремума.

  Исследование против эксплуатации. Чем чаще перезапуск, тем сильнее исследование.

- **Использование большой выборки**. Одновременно рассматривается множество различных потенциальных решений.

  Чем обеспечивается глобальный поиск. При условии, что имеется достаточное количество потенциальных решений, одно из них окажется в области глобального экстремума.

  Исследование против эксплуатации. Чем больше различных потенциальных решений одновременно рассматривается, тем сильнее исследование.

Далее рассматриваются некоторые другие алгоритмы глобальной оптимизации. Основное внимание будет уделено так называемым алгоритмам оптимизации с одним состоянием, которые хранят информацию только об одном потенциальном решении. Вот так: никакой большой выборки.

2.3 Настройка процедуры модификации: $(1+1)$, $(1+\lambda)$ и $(1,\lambda)$

Данные три алгоритма со странными именами являются разновидностью уже рассмотренных процедур локального поиска, но различными операциями Tweak для обеспечения глобальной оптимизации. Они также представляют вырожденные случаи более общих эволюционных алгоритмов (μ,λ) и $(\mu+\lambda)$, рассматриваемых далее (в Разделе 3.1).

Общая задача простая: необходимо разработать операцию Tweak, которая *в основном* производит незначительные улучшения, однако *время от времени* совершает большие изменения, которые *могут* привести к любым результатам. Т.е. большую часть времени осуществляется локальный поиск, однако присутствует возможность случайного перехода в далекую от текущего положения точку. Поэтому имеется вероятность, пусть и небольшая, что локальный поиск окажется удачным и результат операции Tweak окажется в окрестности оптимума.

К примеру представим снова, что решения представлены вещественными векторами фиксированного размера. До этого улучшение таких векторов производилось с использованием алгоритма ограниченной однородной конволюции (Алгоритм 8). Ключевое слово здесь *ограниченный*: оно накладывает необходимость выбора между точностью для локального поиска и возможностью выхода из локального оптимума. Однако **Гауссовское** [1] (также **нормальное**, или колоколообразное) распределение $N(\mu,\sigma^2)$ позволяет сделать и то и другое: как правило оно возвращает малые значения, но иногда и большие. Несмотря на ограниченность, Гауссовское распределение в действительности может *когда-нибудь* вернуть и очень большое число. Данное распределение требует задания двух параметров: среднего μ и дисперсии σ^2. Степень предпочтения малых значений большим может контролироваться простым изменением величины σ^2.

Этого можно добиться, прибавляя к каждому элементу вектора случаный шум с Гауссовским распределением с нулевым средним ($\mu=0$). Это называется **Гауссовской сверткой** (***Gaussian convolution***). Большая часть шума будет близка к 0, поэтому вектор сильно не изменится. Но некоторые значения могут измениться довольно существенно. Как и в случае с алгоритмом 8, шум к каждому элементу добавляется с вероятностью p.

Алгоритм 11 Гауссовская свертка

$\vec{v} \leftarrow$ вектор $\langle v_1, v_2, \dots v_l\rangle$ для конволюции
$p \leftarrow$ вероятность прибавить шум к элементу вектора {Часто $p=1$}
$\sigma^2 \leftarrow$ дисперсия нормального распределения для конволюции {Нормальное = Гауссовское}
$min \leftarrow$ минимальное допустимое значение элемента вектора
$max \leftarrow$ максимальное допустимое значение элемента вектора

for $i=1$ до l **do**
  if $p \geq$ случайное число равномерное распределенное в $[0.0, 1.0]$ **then**
    repeat
      $n \leftarrow$ случайное число распределенное по нормальному закону $N(0,\sigma^2)$
    until $min \leq v_i + n \leq max$
  end if
  $v_i = v_i + n$
end for
return $\vec{v}$

(1+1) – название для стандартного алгоритма локального поиска (Алгоритм 4), использующего такую модифицированную вероятностную версию функции Tweak. **(1+λ)** – название для аналогичным образом модифицированного алгоритма локального поиска с наискорейшим подъемом (Алгоритм 5), а **(1+λ)** – название для модифицированного алгоритма локального поиска с наискорейшим подъемом с замещением (Алгоритм 6). Такие названия кажутся загадочными, но далее им будет дана более содержательная интерпретация.

Если присмотреться то Гауссовская свертка дает не один дополнительный параметр для настройки (σ^2), чтобы регулировать соотношение исследования и эксплуатации, а *два* таких параметра. Рассмотрим алгоритм локального поиска с наискорейшим подъемом с замещением (Алгоритм 6), в котором параметр n определяет сколько потомков будет создано из родительского решения операцией Tweak. В «глобальной» версии алгоритма, $(1,\lambda)$, величина n оказывает существенное влияние на σ^2: если значение σ^2 велико (значительный шум), то алгоритм будет производить активный поиск

[1] Карл Фридрих Гаусс, 1777-1855, вундеркинд, физик и, вероятно, наиболее выдающийся математик всех времен.

		Шум в Tweak	
		Большой	Малый
Точек	Мало		Исследование
	Много	Эксплуатация	

Таблица .1: Упрощенное описание взаимодействия двух факторов и их влияние на баланс между исследованием и эксплуатацией. В качестве факторов рассматриваются: уровень шума в операции Tweak; количество точек, на основании которых генерируется новое потенциальное решение.

в малоприспособленных областях[2], однако при больших n, будет происходить весьма агрессивный отсев плохих потенциальных решений из таких областей. Дело в том, что если значение n мало, то плохое потенциальное решение имеет определенные шансы быть лучше в группе из n объектов, но если n велико, то такая вероятность значительно уменьшается. Поэтому хотя рост σ^2 и направлено в сторону исследования пространства поиска, увеличение n ведет к усилению эксплуатации. Параметр n является примером того, что в дальнейшем будет названо **давлением селекции** (***selection pressure***). В таблице .1 дано обобщение описанного взаимодействия параметров.

Многие генераторы псевдослучайных чисел обеспечивают средства для генерации случайных величин, подчиняющихся нормальному (Гауссвоскому) распределению. Если у вашего нет такой функции, то можно воспользоваться **полярным методом Бокса-Мюллера-Марсаглия**[3] (***Box-Muller-Marsaglia Polar Method***).

Алгоритм 12 Элемент из Гауссовского распределения (Полярный метод Бокса-Мюллера-Марсаглия)

$\mu \leftarrow$ центр нормального распределения {«нормальное» = «Гауссовское»}
$\sigma^2 \leftarrow$ дисперсия нормального распределения

repeat
  $x \leftarrow$ случайное число равномерное распределенное в $[0.0, 1.0]$
  $y \leftarrow$ случайное число равномерное распределенное в $[0.0, 1.0]$ {x и y – независимые}
  $w \leftarrow x^2 + y^2$
until $0 < w < 1$ {В противном случае последует деление на ноль или извлечение корня из отрицательного числа!}
$g \leftarrow \mu + x\sigma\sqrt{-2\frac{\ln w}{w}}$ {Используется σ, т.е. $\sqrt{\sigma^2}$. Также заметьте, что ln это $\log_e$}
$h \leftarrow \mu + y\sigma\sqrt{-2\frac{\ln w}{w}}$ {Аналогичное замечание}
return g и h {Метод генерирует сразу два числа. Если что, используйте только одно.}

Некоторые генераторы псевдослучайных чисел (такие как java.util.Random) используют только **стандартное нормальное распределение** $N(0,1)$. Можно очень легко преобразовать полученные с его помощью числа к Гауссовскому распределению с произвольными средним μ и дисперсией σ^2, либо стандартным отклонением σ:

$$N(\mu, \sigma^2) = \mu + \sqrt{\sigma^2}N(0,1) = \mu + \sigma N(0,1).$$

[2] В оригинале было более интересное выражение: *crazy locations*. – Ю.Ц.

[3] Этот метод был впервые описан в статье George Edward Pelham Box and Mervin Muller, 1958, A note on the generation of random normal deviates, The Annals of Mathematical Statistics , 29(2), 610–611. Однако полярная форма метода, приведенная здесь, обычно приписывается George Marsaglia. Существует и более быстрый, но не более простой, метод с замечательным названием: **метод Зиккурата** (***Ziggurat Method***).