Непрерывные генетические алгоритмы - математический аппарат

Немного истории

Генетические алгоритмы (ГА) – это мощный инструмент для решения сложных задач. Они нашли применение в оптимизации, искусственном интеллекте, инженерии и других областях. В основе ГА лежат принципы, заимствованные из биологии и генетики. Напомним: основная идея ГА состоит в создании популяции особей (индивидов), каждая из которых представляется в виде хромосомы. Любая хромосома есть возможное решение рассматриваемой оптимизационной задачи. Для поиска лучших решений необходимо только значение целевой функции, или функции приспособленности. Значение функции приспособленности особи показывает, насколько хорошо подходит особь, описанная данной хромосомой, для решения задачи. Хромосома состоит из конечного числа генов, представляя генотип объекта, т.е. совокупность его наследственных признаков. Процесс эволюционного поиска ведется только на уровне генотипа. К популяции применяются основные биологические операторы: скрещивания, мутации, инверсии и др. В процессе эволюции действует известный принцип "выживает сильнейший". Популяция постоянно обновляется при помощи генерации новых особей и уничтожения старых, и каждая новая популяция становится лучше и зависит только от предыдущей.

Фиксированная длина хромосомы и кодирование строк двоичным алфавитом преобладали в теории ГА с момента начала ее развития, когда были получены теоретические результаты о целесообразности использования именно двоичного алфавита (подробнее о математическом аппарате ГА с двоичными хромосомами читайте в первом материале этого цикла). К тому же, реализация такого ГА на ЭВМ была сравнительно легкой. Все же, небольшая группа исследователей шла по пути применения в ГА отличных от двоичных алфавитов для решения частных прикладных задач. Одной из таких задач является нахождение решений, представленных в форме вещественных чисел, что называется не иначе как "поисковая оптимизация в непрерывных пространствах". Возникла следующая идея: решение в хромосоме представлять напрямую в виде набора вещественных чисел. Естественно, что потребовались специальные реализации биологических операторов. Такой тип генетического алгоритма получил название *непрерывного ГА* (real-coded GA), или *генетического алгоритма с вещественным кодированием*.

Первоначально непрерывные гены стали использоваться в специфических приложениях (например, хемометрика, оптимальный подбор параметров операторов стандартных ГА и др.). Позднее они начинают применяться для решения других задач оптимизации в непрерывных пространствах (работы исследователей Wright, Davis, Michalewicz, Eshelman, Herrera в 1991-1995 гг). Поскольку до 1991 теоретических обоснований работы непрерывных ГА не существовало, использование этого нового подвида было спорным; ученые, знакомые с фундаментальной теорией эволюционных вычислений, в которой было доказано превосходство двоичного алфавита перед другими, критически воспринимали успехи real-coded алгоритмов. После того, как спустя некоторое время теоретическое обоснование появилось, непрерывные ГА полностью вытеснили двоичные хромосомы при поиске в непрерывных пространствах.

Далее в тексте по аналогии с англоязычной терминологией для ГА с двоичным кодированием будет использоваться аббревиатура BGA (Binary coded), для ГА с непрерывными генами – RGA (Real coded).

Преимущества и недостатки двоичного кодирования

Прежде чем излагать особенности математического аппарата непрерывных ГА, остановимся на анализе достоинств и недостатков двоичной схем кодирования.

Как известно, высокая эффективность отыскания глобального минимума или максимума генетическим алгоритмом с двоичным кодированием теоретически обоснована в фундаментальной теореме генетических алгоритмов ("теореме о шаблоне"), доказанной Холландом. Ее подробное освещение и доказательство можно найти в соответствующих источниках. Ее суть в том, что двоичный алфавит позволяет обрабатывать максимальное количество информации по сравнению с другими схемами кодирования.

Однако двоичное представление хромосом влечет за собой определенные трудности при поиске в непрерывных пространствах большой размерности, и когда требуется высокая точность найденного решения. В BGA для преобразования генотипа в фенотип используется специальный прием, основанный на том, что весь интервал допустимых значений признака объекта $[a_i,b_i]$ разбивается на участки с требуемой точностью. Заданная точность p определяется выражением

$$p = \frac{b_i - a_i}{2^N - 1},$$

где N – количество разрядов для кодирования битовой строки.

Эта формула показывает, что p сильно зависит от N, т.е. точность представления определяется количеством разрядов, используемых для кодирования одной хромосомы. Поэтому при увеличении N пространство поиска расширяется и становится огромным. Известный книжный пример: пусть для 100 переменных, изменяющихся в интервале [-500; 500], требуется найти экстремум с точностью до шестого знака после запятой. В этом случае при использовании ГА с двоичным кодированием длина строки составит 3000 элементов, а пространство поиска – около 10^{1000} хромосом.

Эффективность BGA в этом случае будет невысокой. На первых итерациях алгоритм потратит много усилий на оценку младших разрядов числа, закодированных во фрагменте двоичной хромосомы. Но оптимальное значение на первых итерациях будет зависеть от старших разрядов числа. Следовательно, пока в процессе эволюции алгоритм не выйдет на значение старшего разряда в окрестности оптимума, операции с младшими разрядами окажутся бесполезными. С другой стороны, когда это произойдет, станут не нужны операции со старшими разрядами – необходимо улучшать точность решения поиском в младших разрядах. Такое "идеальное" поведение не обеспечивает семейство алгоритмов BGA, т.к. эти алгоритмы оперируют битовой строкой целиком, и на первых же эпохах младшие разряды чисел "застывают", принимая случайное значение. В классических ГА разработаны специальные приемы по выходу из этой ситуации. Например, последовательный запуск ансамбля генетических алгоритмов с постепенным сужением пространства поиска.

Есть и другая проблема: при увеличении длины битовой строки необходимо увеличивать и численность популяции.

Математический аппарат непрерывных ГА

Как уже отмечалось, при работе с оптимизационными задачами в непрерывных пространствах вполне естественно представлять гены напрямую вещественными числами. В этом случае хромосома есть вектор вещественных чисел. Их точность будет определяться исключительно разрядной сеткой той ЭВМ, на которой реализуется real-coded алгоритм. Длина хромосомы будет совпадать с длиной вектора-решения оптимизационной задачи, иначе говоря, *каждый ген будет отвечать за одну переменную*. Генотип объекта становится идентичным его фенотипу.

Вышесказанное определяет список основных преимуществ real-coded алгоритмов:

1. Использование непрерывных генов делает возможным поиск в больших пространствах (даже в неизвестных), что трудно делать в случае двоичных генов, когда увеличение пространства поиска сокращает точность решения при неизменной длине хромосомы.
2. Одной из важных черт непрерывных ГА является их способность к локальной настройке решений.
3. Использование RGA для представления решений удобно, поскольку близко к постановке большинства прикладных задач. Кроме того, отсутствие операций кодирования/декодирования, которые необходимы в BGA, повышает скорость работы алгоритма.

Как известно, появление новых особей в популяции канонического ГА обеспечивают несколько биологических операторов: отбор, скрещивание и мутация. В качестве операторов отбора особей в родительскую пару здесь подходят любые известные из BGA: рулетка, турнирный, случайный. Однако операторы скрещивания и мутации не годятся: в классических реализациях они работают с битовыми строками. Нужны собственные реализации, учитывающие специфику real-coded алгоритмов.

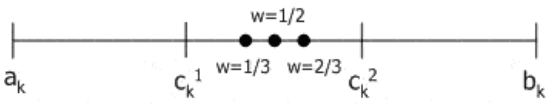
Оператор скрещивания непрерывного ГА, или кроссовер, порождает одного или нескольких потомков от двух хромосом. Собственно говоря, требуется из двух векторов вещественных чисел получить новые векторы по каким-либо законам. Большинство real-coded алгоритмов генерируют новые векторы в окрестности родительских пар. Для начала рассмотрим простые и популярные кроссоверы.

Пусть $C_1=(c_1^1,c_2^1,\ldots,c_n^1)$ и $C_2=(c_1^2,c_2^2,\ldots,c_n^2)$ – две хромосомы, выбранные оператором селекции для скрещивания. После формулы для некоторых кроссоверов приводится рисунок – геометрическая интерпретация его работы. Предполагается, что $c_k^1<=c_k^2$ и $f(C_1)>=f(C_2)$.

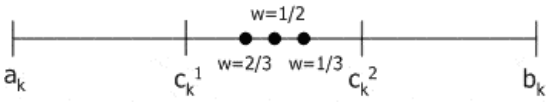
Плоский кроссовер (flat crossover): создается потомок $H=(h_1,\ldots,h_k,\ldots,h_n)$, h_k, k=1,…, n – случайное число из интервала $[c_k^1,c_k^2]$.

Простейший кроссовер (simple crossover): случайным образом выбирается число k из интервала {1,2,…,n-1} и генерируются два потомка $H1=(c_1^1,c_2^1,\ldots,c_k^1,c_{k+1}^2,\ldots,c_n^2)$ и $H_2=(c_1^2,c_2^2,\ldots,c_k^2,c_{k+1}^1,\ldots,c_n^2)$.

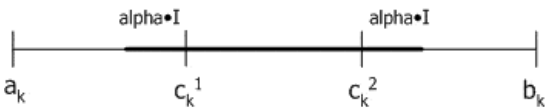
Арифметический кроссовер (arithmetical crossover): создаются два потомка $H_1=(h_1^1,\ldots,h_n^1)$, $H_2=(h_1^2,\ldots,h_n^2)$, где $h_k^1=w*c_k^1+(1-w)*c_k^2$, $h_k^2=w*c_k^2+(1-w)*c_k^1$, k=1,…,n, w либо константа (равномерный арифметический кроссовер) из интервала [0;1], либо изменяется с увеличением эпох (неравномерный арифметический кроссовер).



Геометрический кроссовер (geometrical crossover): создаются два потомка $H_1=(h_1^1,\ldots,h_n^1)$, $H_2=(h_1^2,\ldots,h_n^2)$, где $h_k^1=(c_k^1)^{w}*(c_k^2)^{1-w}$, $(c_k^2)^{w}*(c_k^1)^{1-w}$, w – случайное число из интервала [0;1].

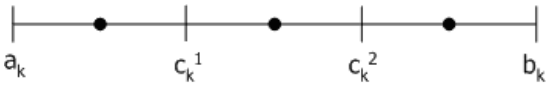


Смешанный кроссовер (blend, BLX-alpha crossover): генерируется один потомок $H=(h_1,\ldots,h_k,\ldots,h_n)$, где h_k – случайное число из интервала $[c_{min}-I*alpha,c_{max}+I*alpha]$, $c_{min}=\min(c_k^1,c_k^2)$, $c_{max}=\max(c_k^1,c_k^2)$, $I=c_{max}-c_{min}$. BLX-0.0 кроссовер превращается в плоский.



Линейный кроссовер (linear crossover): создаются три потомка $H_q=(h_1^q,\ldots,h_k^q,\ldots,h_n^q)$, q=1,2,3, где $h_k^1=0.5*c_k^1+0.5*c_k^2$, $h_k^2=1.5*c_k^1-0.5*c_k^2$, $h_k^3=-0.5*c_k^1+1.5*c_k^2$. На этапе селекции в этом кроссовере

отбираются два наиболее сильных потомка.



Дискретный кроссовер (discrete crossover): каждый ген h_k выбирается случайно по равномерному закону из конечного множества $\{c_k^1, c_k^2\}$.

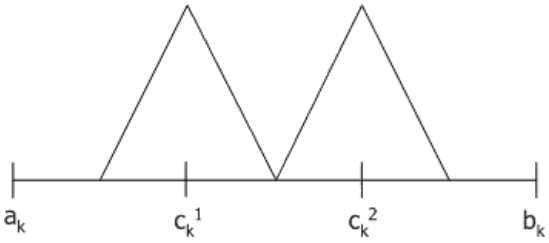
Расширенный линейчатый кроссовер (extended line crossover): ген $h_k=c_k^1+w*(c_k^2-c_k^1)$, w – случайное число из интервала [-0.25;1.25].

Эвристический кроссовер (Wright's heuristic crossover). Пусть C_1 – один из двух родителей с лучшей приспособленностью. Тогда $h_k=w*(c_k^1-c_k^2)+c_k^1$, w – случайное число из интервала [0;1].

Нечеткий кроссовер (fuzzy recombination, FR-d crossover): создаются два потомка $H_1=(h_1^1,\ldots,h_n^1)$, $H_2=(h_1^1,\ldots,h_n^2)$. Вероятность того, что в i-том гене появится число v_i , задается распределением $p(v_i)$ $\{F(c_k^1),F(c_k^2)\}$, где $F(c_k^1),F(c_k^2)$ – распределения вероятностей треугольной формы (треугольные нечеткие функции принадлежности) со следующими свойствами ($c_k^1<=c_k^2$ и $I=|c_k^1-c_k^2|$):

Распределение вероятностей	Минимум	Центр	Максимум
$F(c_k^1)$	c_k^1-d*I	c_k^1	c_k^1+d*I
$F(c_k^2)$	c_k^2-d*I	c_k^2	c_k^2+d*I

Параметр d определяет степень перекрытия треугольных функций принадлежности, по умолчанию d=0.5.



В качестве оператора мутации наибольшее распространение получили: случайная и неравномерная мутация (random and non-uniform mutation).

При случайной мутации ген, подлежащий изменению, принимает случайное значение из интервала своего изменения. В неравномерной мутации значение гена после оператора мутации рассчитывается по формуле:

$$c_k^* = \begin{cases} c_k + \Delta(t, b_k - c_k), & w = 0 \\ c_k - \Delta(t, b_k - c_k), & w = 1 \end{cases},$$

$$\Delta(t, y) = y\left[1 - r^{\left(1-\frac{t}{e\_max}\right)^b}\right].$$

Сложно сказать, что более эффективно в каждом конкретном случае, но многочисленные исследования доказывают, что непрерывные ГА не менее эффективно, а часто гораздо эффективнее справляются с задачами оптимизации в многомерных пространствах, при этом более просты в реализации из-за отсутствия процедур кодирования и декодирования хромосом.

Рассмотренные кроссоверы исторически были предложены первыми, однако во многих задачах их эффективность оказывается невысокой. Исключение составляет BLX-кроссовер с параметром alpha=0.5 – он превосходит по эффективности большинство простых кроссоверов. Позднее были разработаны улучшенные операторы скрещивания, аналитическая формула которых и эффективность обоснованы теоретически. Рассмотрим подробнее один из таких кроссоверов – SBX

SBX кроссовер

SBX (англ.: Simulated Binary Crossover) – кроссовер, имитирующий двоичный. Был разработан в 1995 году исследовательской группой под руководством K. Deb'a. Как следует из его названия, этот кроссовер моделирует принципы работы двоичного оператора скрещивания.

SBX кроссовер был получен следующим способом. У двоичного кроссовера было обнаружено важное свойство – среднее значение функции приспособленности оставалось неизменным у родителей и их потомков, полученных путем скрещивания. Затем автором было введено понятие силы поиска кроссовера (search power). Это количественная величина, характеризующая распределение вероятностей появления любого потомка от двух произвольных родителей. Первоначально была рассчитана сила поиска для одноточечного двоичного кроссовера, а затем был разработан вещественный SBX кроссовер с такой же силой поиска. В нем сила поиска характеризуется распределением вероятностей случайной величины β:

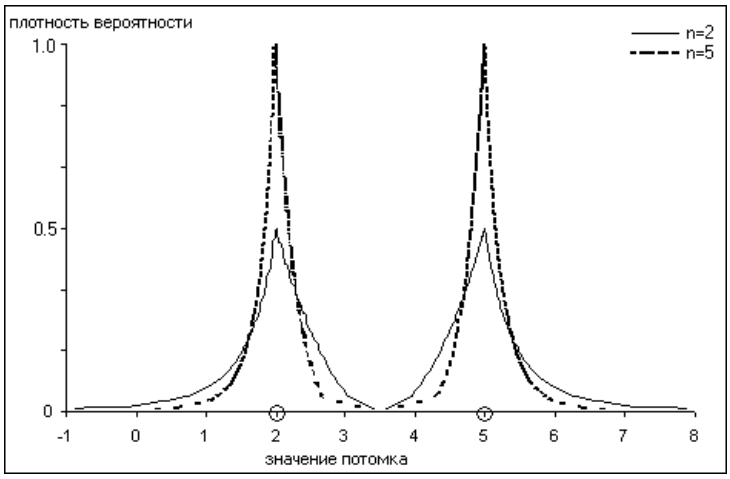
$$P(\beta) = \begin{cases} 0.5(n+1)\beta^n, & \beta \le 1 \\ 0.5(n+1)\beta^{-(n+2)}, & \beta > 1. \end{cases}$$

Для генерации потомков используется следующий алгоритм, использующий выражение для $P(\beta)$. Создаются два потомка $H_k=(h_1^k, \ldots, h_j^k, \ldots, h_n^k)$, k=1,2, где $h_j^1 = 0.5 \cdot [(1-\beta_k)c_j^1 + (1+\beta_k)c_j^2]$, – число, полученное по формуле:

$$\beta(u) = \begin{cases} (2u)^{\frac{1}{n+1}}, & u(0,1) \le 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{n+1}}, & u(0,1) > 0.5. \end{cases}$$

В формуле $u(0,1)$ – случайное число, распределенное по равномерному закону, n [2,5] – параметр кроссовера.

На рисунке приведена геометрическая интерпретация работы SBX кроссовера при скрещивании двух хромосом, соответствующих вещественным числам 2 и 5. Видно, как параметр n влияет на конечный результат: увеличение n влечет за собой увеличение вероятности появления потомка в окрестности родителя и наоборот.



Эксперименты автора SBX кроссовера показали, что он во многих случаях эффективнее BLX, хотя, очевидно, что не существует ни одного кроссовера, эффективного во всех случаях. Исследования показывают, что использование нескольких различных операторов кроссовера позволяет уменьшить вероятность преждевременной сходимости, т.е. улучшить эффективность алгоритма оптимизации в целом. Для этого могут использоваться специальные стратегии, изменяющие вероятность применения отдельного эволюционного оператора в зависимости от его «успешности», или использование гибридных кроссоверов, которых в настоящее время насчитывается несколько десятков. В любом случае, если перед Вами стоит задача оптимизации в непрерывных пространствах, и Вы планируете применить эволюционные техники, то следует сделать выбор в пользу непрерывного генетического алгоритма.

Николай Пакплин

Литература

1. Herrera F., Lozano M., Verdegay J.L. Tackling real-coded Genetic algorithms: operators and tools for the behaviour analysis // Artificial Intelligence Review, Vol. 12, No. 4, 1998. – P. 265-319.
2. Herrera F., Lozano M., Sanchez A.M. Hybrid Crossover Operators for Real-Coded Genetic Algorithms: An Experimental Study // Soft Comput. 9(4): 280-298 (2005).
3. Wright A. Genetic algorithms for real parameter optimization // Foundations of Genetic Algorithms, V. 1. – 1991. – P. 205-218.
4. Deb, K. and Kumar, A. (1995). Realcoded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems. Complex Systems, 9(6), 431–454.