Федеральное агентство по образованию
Воронежский государственный университет

И.Л. Каширина

ВВЕДЕНИЕ

В ЭВОЛЮЦИОННОЕ МОДЕЛИРОВАНИЕ

Учебное пособие

ВОРОНЕЖ
2007

Утверждено Научно-методическим советом факультета прикладной математики, информатики и механики 25 января 2007 г., протокол № 5

В учебном пособии рассмотрены основные стратегии, принципы и концепции нового направления «Эволюционное моделирование». Описаны фундаментальные основы генетических алгоритмов. Проанализированы архитектуры генетического поиска и модели генетических операторов. Приведены конкретные примеры решения основные задач оптимизации на основе генетических алгоритмов.
Пособие подготовлено на кафедре Математических методов исследования операций факультета Прикладной математики, информатики и механики Воронежского государственного университета. Рекомендуется для магистров первого года обучения факультета ПММ.

Для специальности: 010500 (510200) Прикладная математика и информатика

1. Основные идеи и механизмы эволюционного моделирования

Основной тезис подхода, названого ***эволюционным моделированием*** □□ заменить процесс моделирования сложного объекта моделированием его эволюции. Он направлен на применение механизмов естественной эволюции при синтезе сложных систем обработки информации. В своей теории происхождения видов Ч. Дарвин открыл и обосновал основной закон развития органического мира, охарактеризовав его взаимодействием трех следующих факторов:

1) наследственной изменчивости;
2) борьбы за существование;
3) естественного отбора.

Дарвиновская теория получила подтверждение и развитие в генетике и других науках. Одним из крупнейших эволюционистов, нашим соотечественником И.И.Шмальгаузеном были выдвинуты следующие необходимые и достаточные условия, определяющие неизбежность эволюции:

1) наследственная изменчивость, то есть мутации как предпосылка эволюции, ее материал;
2) борьба за существование как контролирующий и направляющий фактор;
3) естественный отбор как преобразующий фактор.

Понятие «эволюционное моделирование» сформировалось в работах Л. Фогеля, А. Оуэне, М. Уолша. В 1966 году вышла их совместная книга «Искусственный интеллект и эволюционное моделирование».

Эволюционные методы

Эволюционные вычисления □□ термин, обычно используемый для общего описания алгоритмов поиска, оптимизации или обучения, основанных на формализованных принципах естественного эволюционного процесса. ***Эволюционные методы*** предназначены для поиска предпочтительных решений и основаны на статистическом подходе к исследованию ситуаций и *итерационным приближении* к искомому состоянию систем. В отличие от точных методов математического программирования *эволюционные методы* позволяют находить решения, близкие к оптимальным, за приемлемое время, а в отличии от известных эвристических методов оптимизации характеризуются существенно меньшей зависимостью от особенностей приложения (т.е. более универсальны) и во многих случаях обеспечивают лучшую степень приближения к оптимальному решению. Основное преимущество эволюционных методов оптимизации заключается в возможности

решения многомодальных (имеющих несколько локальных экстремумов) задач с большой размерностью за счет сочетания элементов случайности и детерминированности точно так же, как это происходит в природной среде.

Детерминированность этих методов заключается в моделировании природных процессов отбора, размножения и наследования, происходящих по строго определенным правилам, основным из которых является является закон эволюции: «выживает сильнейший». Другим важным фактором эффективности эволюционных вычислений является моделирование процессов размножения и наследования. Рассматриваемые варианты решений могут по определенному правилу порождать новые решения, которые будут наследовать лучшие черты своих «предков».

В качестве случайного элемента в методах эволюционных вычислений используется моделирование процесса мутации. С ее помощью характеристики того или иного решения могут быть случайно изменены, что приведет к новому направлению в процессе эволюции решений и может ускорить процесс выработки лучшего решения.

Достоинства и недостатки эволюционных вычислений

Преимущества и недостатки эволюционных вычислений можно вкратце сформулировать следующим образом.

Достоинства эволюционных вычислений:

- широкая область применения;
- возможность проблемно-ориентированного кодирования решений, подбора начальной популяции, комбинирования эволюционных вычислений с неэволюционными алгоритмами, продолжения процесса эволюции до тех пор, пока имеются необходимые ресурсы;
- пригодность для поиска в сложном пространстве решений большой размерности;
- отсутствие ограничений на вид целевой функции;
- ясность схемы и базовых принципов эволюционных вычислений;
- интегрируемость эволюционных вычислений с другими неклассическими парадигмами искусственного интеллекта, такими как искусственные нейросети и нечеткая логика.

Недостатки эволюционных вычислений:

- эвристический характер эволюционных вычислений не гарантирует оптимальности полученного решения (правда, на практике, зачастую, важно за заданное время получить одно или несколько субоптимальных альтернативных решений, тем более что исходные данные в задаче могут динамически меняться, быть неточными или неполными);
- относительно высокая вычислительная трудоемкость, которая однако преодолевается за счет распараллеливания на уровне организации эво-

люционных вычислений и на уровне их непосредственной реализации в вычислительной системе;

- относительно невысокая эффективность на заключительных фазах моделирования эволюции (операторы поиска в эволюционных алгоритмах не ориентированы на быстрое попадание в локальный оптимум);
- нерешенность вопросов самоадаптации.

История эволюционных вычислений началась с разработки ряда различных независимых моделей эволюционного процесса. Среди этих моделей можно выделить несколько основных парадигм:

- генетические алгоритмы;
- генетическое программирование;
- эволюционные стратегии;
- эволюционное программирование.

2. Основные понятия и базовая схема генетического алгоритма

Важнейшим частным случаем *эволюционных методов* являются ***генетические методы***, основанные на поиске лучших решений с помощью наследования и усиления полезных свойств множества объектов определенного приложения в процессе имитации их эволюции.

Генетические алгоритмы (ГА) □□ это стохастические, эвристические оптимизационные методы, впервые предложенные Холландом (1975). Идея генетических алгоритмов заимствована у живой природы и состоит в организации эволюционного процесса, конечной целью которого является получение решения в сложной задаче оптимизации. Разработчик генетических алгоритмов выступает в данном случае как "создатель", который должен правильно установить законы эволюции, чтобы достичь желаемой цели как можно быстрее. Впервые эти нестандартные идеи были применены к решению оптимизационных задач в середине 70-х г. Примерно через десять лет появились первые теоретические обоснования этого подхода. На сегодняшний день генетические алгоритмы доказали свою конкурентоспособность при решении многих трудных задач и особенно в практических приложениях, где математические модели имеют сложную структуру и применение стандартных методов типа ветвей и границ, динамического или линейного программирования крайне затруднено.

Цель в оптимизации с помощью ГА состоит в том, чтобы найти лучшее возможное решение или решения задачи по одному или нескольким критериям. Чтобы реализовать генетический алгоритм, нужно сначала выбрать подходящую структуру для представления этих решений. В постановке задачи поиска объект этой структуры данных представляет точку в про-

странстве поиска всех возможных решений. Свойства объектов представлены значениями параметров, объединяемыми в запись, называемую в *эволюционных методах* ***хромосомой***. В генетических методах оперируют хромосомами, относящимися к множеству объектов □□ ***популяции***. Имитация генетических принципов □□ ***вероятностный выбор*** родителей, среди членов популяции, скрещивание их хромосом, отбор потомков для включения в новые поколения объектов на основе оценки целевой функции □□ ведет к эволюционному улучшению значений ***целевой функции*** *(функции приспособленности)* от поколения к поколению.

Чаще всего хромосома □□ это битовая строка. Однако ГА не ограничены бинарным представлением данных. Некоторые реализации используют целочисленное или вещественное кодирование. Несмотря на то, что для многих реальных задач, видимо, больше подходят строки переменной длины, в настоящее время структуры фиксированной длины наиболее распространены и изучены. Вначале и мы ограничимся только структурами, которые являются одиночными строками по *n* бит.

Каждая ***хромосома*** (строка) представляет собой конкатенацию ряда подкомпонентов, называемых генами. Гены располагаются в различных позициях или локусах хромосомы и принимают значения, называемые аллелями. В представлениях с бинарными строками ***ген*** □□ бит, ***локус*** □□ его позиция в строке и ***аллель*** □□ его значение (0 или 1). Биологический термин "***генотип***" относится к полной генетической модели особи и соответствует структуре в ГА. Термин "***фенотип***" относится к внешним наблюдаемым признакам и соответствует вектору в пространстве параметров. Чрезвычайно простой, но иллюстративный пример - задача максимизации следующей функции двух переменных:

$$f(x_1, x_2) = x_1 x_2, \quad \text{где} \quad 0 <= x_1 <= 1 \quad \text{и} \quad 0 <= x_2 <= 1.$$

Обычно методика кодирования реальных переменных x_1 и x_2 состоит в их преобразовании в двоичные целочисленные строки достаточной длины □□ достаточной для того, чтобы обеспечить желаемую точность. Предположим, что 10-разрядное кодирование достаточно и для x_1, и x_2. Установить соответствие между генотипом и фенотипом закодированных особей можно, разделив соответствующее двоичному представлению целое число на значение $2^{10}-1$. Например, код [0000000000] соответствует вещественному значению 0/1023 или 0, тогда как код [1111111111] соответствует 1023/1023 или 1. Оптимизируемая структура данных □□ 20-битная строка, представляющая конкатенацию кодировок x_1 и x_2. Переменная x_1 размещается в крайних левых 10-разрядах, тогда как x_2 размещается в правой части генотипа особи (20-битовой строке). *Генотип* □□ точка в 20-мерном бинарном пространстве, исследуемом ГА. *Фенотип* □□ точка в двумерном пространстве параметров.

Кодирование решений

После того как выбраны параметры, их число и разрядность, необходимо решить, как непосредственно записывать данные. Можно использовать обычное кодирование, когда, например, $1011_2=11_{10}$, либо ***коды Грея***, когда $1011_G = 1110_2 = 14_{10}$. Несмотря на то, что коды Грея влекут неизбежное кодирование/декодирование данных, они позволяют избежать некоторых проблем, которые появляются в результате обычного кодирования. Можно лишь сказать, что преимущество кода Грея в том, что если два числа являются последовательными при кодировании, то и их двоичные коды различаются только на один разряд, а в двоичных кодах это не так. Стоит отметить, что кодировать и декодировать в коды Грея можно так: сначала копируется самый старший разряд, затем:

- из двоичного кода в код Грея: G[i] = XOR(B[i+1], B[i]) ;
- из кода Грея в двоичный код: B[i] = XOR(B[i+1], G[i]).

Здесь, G[i] □□ i-й разряд кода Грея, а B[i] - i-й разряд бинарного кода. Например, последовательность чисел от 0 до 7 в двоичном коде: {000, 001, 010, 011, 100, 101, 110, 111}, а в кодах Грея: {000, 001, 011, 010, 110, 111, 101, 100}.

Общая схема генетического алгоритма

Общая схема такого алгоритма может быть записана следующим образом.

1. *Формирование начальной популяции.*
2. *Оценка особей популяции.*
3. *Отбор (селекция).*
4. *Скрещивание.*
5. *Мутация.*
6. *Формирование новой популяции.*
7. *Если популяция не сошлась, то 2. Иначе –останов.*

Остановимся подробнее всех этапах этого алгоритма.

Формирование начальной популяции

Стандартный генетический алгоритм начинает свою работу с формирования ***начальной популяции*** I_0 — конечного набора допустимых решений задачи. Эти решения могут быть выбраны случайным образом или получены с помощью простых приближенных алгоритмов. Выбор начальной популяции не имеет значения для сходимости процесса в асимптотике, однако формирование "хорошей" начальной популяции (например, из множества локальных оптимумов) может заметно сократить время достижения глобального оптимума. Если отсутствуют предположения о местоположении глобального оптимума, то индивиды из начальной популяции желательно распределить равномерно по всему пространству поиска решения.

Оценка особей популяции

Чтобы оптимизировать какую-либо структуру с использованием ГА, нужно задать меру качества для каждого индивида в пространстве поиска. Для этой цели используется ***функция приспособленности***. В задачах максимизации целевая функция часто сама выступает в качестве функции приспособленности (например, как в рассмотренном ранее двумерном примере); для задач минимизации целевую функцию следует инвертировать. Если решаемая задача имеет ограничения, выполнение которых невозможно контролировать алгоритмически, то функция приспособленности, как правило, включает также штрафы за невыполнение ограничений (они уменьшают ее значение).

Отбор (селекция)

На каждом шаге эволюции с помощью вероятностного оператора ***селекции*** (отбора) выбираются два решения-родителя для их последующего скрещивания. Среди операторов селекции наиболее распространенными являются два вероятностных оператора ***пропорциональной*** и ***турнирной*** селекции. В некоторых случаях также применяется ***отбор усечением***.

Пропорциональный отбор (Proportional selection)

При пропорциональной селекции вероятность на k-м шаге выбрать решение i в качестве одного из родителей задается формулой: $P\{i-выбрано\} = \dfrac{f(i)}{\sum_{j \in I_k} f(j)}$, в предположении, что $f(i) > 0$ для всех $i \in I_k$ (здесь I_k –популяция на k-м шаге).

Простейший пропорциональный отбор □□ рулетка □□ отбирает особей с помощью n "запусков" рулетки. Колесо рулетки содержит по одному сектору для каждого i-го члена популяции. Размер i-го сектора пропорционален соответствующей величине $P(i)$. При таком отборе члены популяции с более высокой приспособленностью с большей вероятностью будут чаще выбираться, чем особи с низкой приспособленностью.

Турнирный отбор (Tournament selection)

Турнирный отбор может быть описан следующим образом: из популяции, содержащей m строк (***особей***), выбирается случайным образом t строк и лучшая строка записывается в промежуточный массив (между выбранными строками проводится турнир). Эта операция повторяется m раз. Строки в полученном промежуточном массиве затем используются для скрещивания (также случайным образом). Размер группы строк, отбираемых для турнира, часто равен 2. В этом случае говорят о *двоичном/парном турнире*. Вообще же t называется *численностью турнира.*

Отбор усечением (Truncation selection)

Данная стратегия использует отсортированную по убыванию популяцию. Число особей для скрещивания выбирается в соответствии с *порогом* $T \in [0;1]$. Порог определяет, какая доля особей, начиная с самой первой (самой приспособленной), будет принимать участие в отборе. В принципе, порог можно задать и равным 1, тогда все особи текущей популяции будут допущены к отбору. Среди особей, допущенных к скрещиванию случайным образом m/2 раз выбираются родительские пары, потомки которых образуют новую популяцию.

Скрещивание

Как только два решения-родителя выбраны, к ним применяется вероятностный ***оператор скрещивания*** (***crossover***), который строит на их основе новые (1 или 2) решения-потомка. Отобранные особи подвергаются кроссоверу (иногда называемому рекомбинацией) с заданной вероятностью P_c. Если каждая пара родителей порождает двух потомков, для воспроизводства популяции необходимо скрестить $m/2$ пары. Для каждой пары с вероятностью P_c применяется кроссовер. Соответственно, с вероятностью $1-P_c$ кроссовер не происходит □□ и тогда неизмененные особи переходят на следующую стадию (мутации).

Существует большое количество разновидностей оператора скрещивания. Простейший ***одноточечный кроссовер*** работает следующим образом. Сначала случайным образом выбирается одна из возможных точек разрыва. (Точка разрыва □□ участок между соседними битами в строке.) Обе родительские структуры разрываются на два сегмента по этой точке. Затем соответствующие сегменты различных родителей склеиваются и получаются два генотипа потомков.

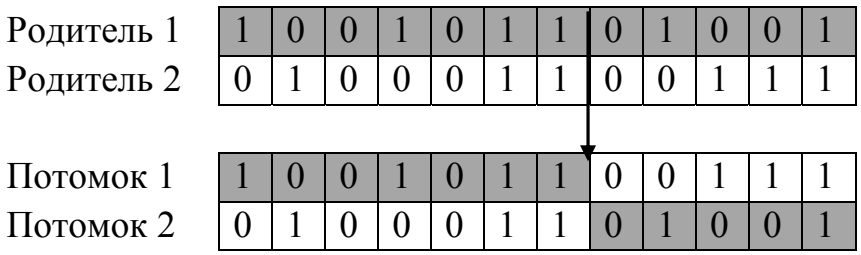

Рис. 1. Пример работы одноточечного кроссовера

В настоящее время исследователи ГА предлагают много других операторов скрещивания. ***Двухточечный кроссовер*** и ***равномерный кроссовер*** □□ вполне достойные альтернативы одноточечному оператору. В двухточечном кроссовере выбираются две точки разрыва, и родительские хромосомы обмениваются сегментом, который находится между двумя этими точками. В равномерном кроссовере каждый бит первого потомка случайным образом наследуется от одного из родителей; второму потомку достается бит другого родителя.

Мутация

После того как закончится стадия кроссовера, потомки могут подвергаться случайным модификациям, называемым ***мутациями***. В простейшем случае в каждой хромосоме, которая подвергается ***мутации***, каждый бит с вероятностью P_m изменяется на противоположный (это так называемая ***одноточечная мутация***).

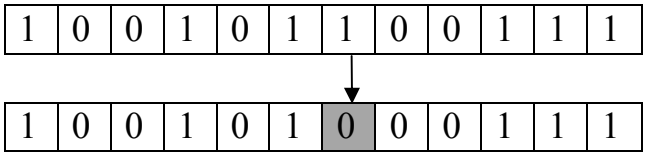

Рис. 2. Пример действия мутации

Более сложной разновидностью мутации являются операторы *инверсии* и *транслокации*. ***Инверсия*** –это перестановка генов в обратном порядке внутри наугад выбранного участка хромосомы.

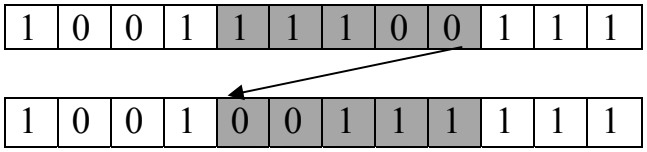

Рис. 3. Пример действия инверсии

Транслокация - это перенос какого-либо участка хромосомы, в другой сегмент этой же хромосомы.

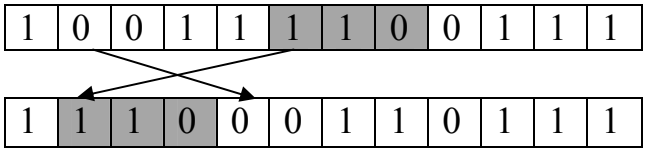

Рис. 4. Пример действия транслокации.

Стоит отметить, что все перечисленные генетические операторы (одноточечный и многоточечный кроссовер, одноточечная мутация, инверсия, транслокация) имеют схожие биологические аналоги.

В некоторых работах предлагается использовать стратегию ***инцеста*** как механизма самоадаптации оператора мутации. Она заключается в том, что вероятность мутации каждого гена P_m определяется для каждого потомка на основании генетической близости его родителей. Например, это может быть отношение числа совпадающих генов родителей к общему числу генов хромосомы. Это приводит к интересному эффекту □□ при высоком разнообразии генофонда популяции последствия мутации будут минимальными, что позволяет оператору скрещивания работать без стороннего вмешательства. В случае же понижения разнообразия, что возникает в основном при застревании алгоритма в локальном оптимуме, последствия мутации становятся более ощутимыми, а при полном схождении популяции алгоритм про-

сто становится стахостическим, что увеличивает вероятность выхода популяции из локального оптимума.

Иногда (с целью повышения средней приспособленности популяции) допустимо осуществлять ***направленные мутации***, т. е. после каждого изменения хромосомы проверять, повысилась ли в результате этой мутации ее приспособленность и, если нет, возвращать хромосому к исходному состоянию.

Формирование нового поколения

После скрещивания и мутации особей необходимо решить проблему о том, какие из новых особей войдут в следующее поколение, а какие □□ нет, и что делать с их предками. Есть два наиболее распространенных способа.

1. Новые особи (потомки) занимают места своих родителей. После чего наступает следующий этап, в котором потомки оцениваются, отбираются, дают потомство и уступают место своим "детям".
2. Следующая популяция включает в себя как родителей, так и их потомков.

Во втором случае необходимо дополнительно определить, какие из особей родителей и потомков попадут в новое поколение. В простейшем случае, в него после каждого скрещивания включаются две лучших особи из четверки родителей и их потомков. Более эффективным является механизм ***вытеснения,*** который реализуется таким образом, что стремится удалять «похожие» хромосомы из популяции и оставлять отличающиеся.

Принцип "элитизма"

Суть этого принципа заключается в том, что в новое поколение всегда включаются лучшие родительские особи. Их число может быть от 1 и больше. Использование "элитизма" позволяет не потерять хорошее промежуточное решение, но в то же время из-за этого алгоритм может "застрять" в локальном экстремуме. В большинстве случаев "элитизм" не вредит поиску решения, и главное □□ это предоставить алгоритму возможность анализировать *разные* хромосомы из пространства поиска.

Останов алгоритма

Работа ГА представляет собой итерационный процесс, который продолжается до тех пор, пока не пройдет заданное число поколений или не выполнится какой-либо иной критерий останова. В оптимизационных задачах традиционными критериями останова алгоритма являются, например, длительное отсутствие прогресса в смысле улучшения значения средней (или лучшей) приспособленности популяции, малая разница между лучшим и худшим значением приспособленности для текущей популяции и т.п.

2. Модели ГА

1. *Canonical GA (J. Holland)*

Данная модель алгоритма является классической. Она была предложена Джоном Холландом в его знаменитой работе "Адаптация в природных и исусственных средах" (1975). Часто можно встретить описание *простого ГА* (Simple GA, D. Goldberg), он отличается от канонического тем, что использует либо рулеточный, либо турнирный отбор. Модель канонического ГА имеет следующие характеристики.

- Фиксированный размер популяции.
- Фиксированная разрядность генов.
- Пропорциональный отбор.
- Одноточечный кроссовер и одноточечная мутация.
- Следующее поколение формируется из потомков текущего поколения без "элитизма".

Алгоритм работы ГА (***репродуктивный план Холланда***) может иметь в данном случае следующий вид.

1. *Инициализация начальной популяции.* Положить номер эпохи $t=0$. Инициализировать случайным образом m генотипов особей и сформировать из них случайную популяцию. Вычислить приспособленность особей популяции $F(0)=(f_1(0), .., f_m(0))$, а затем □□ среднюю приспособленность по популяции $f_{cp}(0) = \sum_{i=1}^{m} f_i(0)/m$.
2. *Выбор родителей для скрещивания.* Увеличить номер эпохи на единицу*:* $t = t + 1$. Определить случайным образом номер первого родителя $l \in \{1 \dots m\}$, назначив вероятность выпадения любого номера h пропорциональной величине $f_h(t)/f_{cp}(t)$. Повторным испытанием определить номер второго родителя k.
3. *Формирование генотипа потомков.* С заданной вероятностью p_c провести над генотипами выбранных родителей одноточечный кроссовер. Далее к каждому из полученных потомков с вероятностью p_M применить оператор мутации.
4. *Обновление популяции.* Поместить потомков в популяцию, предварительно удалив из нее родителей. Вычислить приспособленности потомков и обновить значение средней приспособленности популяции $f_{cp}(t)$. Если формирование популяции не завершено, перейти к шагу 2.

Замечание. В некоторых модификациях этого алгоритма потомки замещают в популяции не своих родителей, а две случайно выбранные особи.

2. Genitor (D. Whitley)

В данной модели используется специфичная стратегия отбора. Вначале, как и полагается, популяция инициализируется и её особи оцениваются. Затем выбираются случайным образом две особи, скрещиваются, причем получается только один потомок, который оценивается и занимает место наименее приспособленной особи. После этого снова случайным образом выбираются 2 особи, и их потомок занимает место особи с самой низкой приспособленностью. Таким образом на каждом шаге в популяции обновляется только одна особь. Подводя итоги, можно выделить следующие характерные особенности.

- Фиксированный размер популяции.
- Фиксированная разрядность генов.
- Особи для скрещивания выбираются случайным образом.
- Ограничений на тип кроссовера и мутации нет.
- В результате скрещивания особей получается один потомок, который занимает место наименее приспособленной особи.

3. Hybrid algorithm (L. "Dave" Davis)

Использование гибридного алгоритма позволяет объединить преимущества ГА с преимуществами классических методов. Дело в том, что ГА являются робастными алгоритмами, т.е. они позволяют находить хорошее решение, но нахождение оптимального решения зачастую оказывается намного более трудной задачей в силу стохастичности принципов работы алгоритма. Поэтому возникла идея использовать ГА на начальном этапе для эффективного сужения пространства поиска вокруг глобального экстремума, а затем, взяв лучшую особь, применить один из "классических" методов оптимизации. Характеристики алгоритма имеют следующий вид.

- Фиксированный размер популяции.
- Фиксированная разрядность генов.
- Любые комбинации стратегий отбора и формирования следующего поколения
- Ограничений на тип кроссовера и мутации нет.
- ГА применяется на начальном этапе, а затем в работу включается классический метод оптимизации.

4. Island Model GA

Представим себе следующую ситуацию. В некотором океане есть группа близкорасположенных островов, на которых живут популяции особей одного вида. Эти популяции развиваются независимо и только изредка происходит обмен представителями между популяциями. Островная модель ГА использует описанный принцип для поиска решения. Вариант, безуслов-

но, интересный и является одной из разновидностей параллельных ГА. Данная модель генетического алгоритма обладает следующими свойствами.

- Наличие нескольких популяций, как правило, одинакового фиксированного размера.
- Фиксированная разрядность генов.
- Любые комбинации стратегий отбора и формирования следующего поколения в каждой популяции. Можно сделать так, что в разных популяциях будут использоваться разные комбинации стратегий, хотя даже один вариант дает разнообразные решения на различных "островах".
- Ограничений на тип кроссовера и мутации нет.
- Случайный обмен особями между "островами". Если миграция будет слишком активной, то особенности островной модели будут сглажены и она будет не очень сильно отличаться от моделей ГА без параллелизма.

5. CHC (Eshelman)

CHC расшифровывается как **C**ross-population selection, **H**eterogenous recombination and **C**ataclysmic mutation. Данный алгоритм довольно быстро сходится из-за того, что в нем нет мутаций, используются популяции небольшого размера и отбор особей в следующее поколение ведется и между родительскими особями, и между их потомками. После нахождения некоторого решения алгоритм перезапускается, причем лучшая особь копируется в новую популяцию, а оставшиеся особи подвергаются сильной мутации (мутирует примерно треть битов в хромосоме) и поиск повторяется. Еще одной специфичной чертой является стратегия скрещивания: все особи разбиваются на пары, причем скрещиваются только те пары, в которых хромосомы особей существенно различны (хэммингово расстояние больше некоторого порога, плюс возможны ограничения на максимальную длину цепочки одинаковых бит). При скрещивании используется так называемый HUX-оператор (Half Uniform Crossover) - это разновидность однородного кроссовера, но в нем к каждому потомку попадает ровно половина битов хромосомы от каждого родителя. Таким образом, модель обладает следующими свойствами.

- Фиксированный размер популяции.
- Фиксированная разрядность генов.
- Перезапуск алгоритма после нахождения решения.
- Небольшая популяция.
- Особи для скрещивания разбиваются на пары и скрещиваются при условии существенных отличий.
- Отбор в следующее поколение проводится между родительскими осо-

бями и потомками.

- Используется половинный однородный кроссовер (HUX).
- Макромутация при перезапуске.

3. Теорема схем (шим, шаблонов)

Схемой (шимой) называется строка вида $(s_1, s_2, ..., s_i, ..., s_n)$, $s_i \in \{0,1,*\}$. Символом "*" в некотором разряде обозначается то, что там может быть как 1, так и 0. Например, для двух бинарных строк "111000111000" и "110011001100" схема будет выглядеть следующим образом: "11*0****1*00". Т.е. с помощью схем можно как бы выделять общие участки двоичных строк и маскировать различия. Имея в составе схемы m символов "*" можно закодировать (обобщить) 2^m двоичных строк. Так, например, схема "01*0*1" описывает набор строк {"010001", "010011", "011001", "011011"}.

Опеределяющей длиной схемы называется расстояние между двумя крайними символами "0" и/или "1". Для схемы "01*0*1" определяющая длина равна 5, а для схемы "**0**1*" определяющая длина равна 3. ***Порядок схемы*** □□ это ещё одна характеристика схемы, она равна числу фиксированных позиций в строке, т.е. общему числу "0" и "1". Для схем "01*0*1" и "**0**1*" порядки равны 4 и 2 соответственно.

Хотя внешне кажется, что ГА обрабатывает строки, на самом деле при этом неявно происходит обработка схем, которые представляют шаблоны подобия между строками (Гольдберг, 1989; Холланд, 1992). ГА практически не может заниматься полным перебором всех представлений в пространстве поиска. Однако он может производить выборку значительного числа гиперплоскостей в областях поиска с высокой приспособленностью. Каждая такая гиперплоскость соответствует множеству похожих строк с высокой приспособленностью.

Строящие блоки (Goldberg, 1989) □□ это схемы, обладающие:

- высокой приспособленностью,
- низким порядком,
- короткой определяющей длиной.

Приспособленность схемы определяется как среднее приспособленностей строк, которые ее содержат. После процедуры отбора остаются только строки с более высокой приспособленностью. Следовательно, строки, которые являются примерами схем с высокой приспособленностью, выбираются чаще. Кроссовер реже разрушает схемы с более короткой определенной длиной, а мутация реже разрушает схемы с низким порядком. Поэтому такие схемы имеют больше шансов переходить из поколения в поколение. Холланд (1992) показал, что, в то время как ГА явным образом обрабатывает n строк на каждом поколении, в то же время неявно обрабатываются порядка

n^3 таких коротких схем низкого порядка и с высокой приспособленностью. Он называл это явление неявным параллелизмом. Для решения реальных задач присутствие неявного параллелизма означает, что большая популяция имеет больше возможностей локализовать решение экспоненциально быстрее популяции с меньшим числом особей.

Теорема схем (The schema theorem)

Теорема схем является одной из основных теорем теории ГА. Она получена для канонического генетического алгоритма.

Предположим, что у нас есть популяция двоичных строк длины *n*. Вероятность проведения кроссовера равна P_c. Тип кроссовера □□ одноточечный. Пусть определяющая длина схемы *S* равна $\delta(S)$, а её приспособленность $f(S)$. Доля строк, соответствующих схеме *S* в текущем поколении t, равна $m(H,t)$. Нас интересует, какая доля строк, соответствующих схеме S будет присутствовать в популяции в следующем поколении □□ $m(S,t+1)$. Вычислим, с какой вероятностью кроссовер разрушит уже имеющуюся схему. Очевидно, что если точка разрыва не попадает внутрь уже имеющейся схемы, то схема не будет разрушена. Т.е. если $P_c*\delta(S)/(n-1)$ □□ вероятность того, что точка разрыва попадет внутрь схемы, то $1-P_c*\delta(H)/(L-1)$ □□ вероятность того, что кроссовер не разрушит схему. Согласно стратегии отбора канонического ГА шансы особи принять участие в скрещивании вычисляются в соответствии с отношением f/f_{cp}, где f □□ значение приспособленности данной особи, а f_{cp} - средняя приспособленность. Таким образом, вероятность того, что строка соответствующая схеме H будет участвовать в скрещивании равна $m(S,t)*f(S,t)/f_{cp}(t)$. Принимая во внимание вероятность разрушения схемы кроссовером, можно сформулировать первоначальную теорему схем (Холланд, 1975):

$$m(S,t+1) \geq m(S,t)\frac{f(S,t)}{f_{cp}(t)}\left[1-p_c\frac{\partial(S)}{n-1}\right].$$

Одним из недостатков данной теоремы является то, что в ней отсутствует влияние мутации на создание и разрушение схем. Если считать, что вероятность мутации равна P_m, а порядок схемы H равен $o(S)$, то вероятность того, что мутация не разрушит схему равна $(1-P_m)^{o(S)}$. Т.е. если мутирующий разряд не попадает ни на одну фиксированную позицию внутри схемы, то она не изменяется. С учетом этого исправленная теорема схем выглядит следующим образом (Холланд, 1992):

$$m(S,t+1) \geq m(S,t)\frac{f(S,t)}{f_{cp}(t)}\left[1-p_c\frac{\partial(S)}{n-1}\right](1-p_m)^{o(S)}$$

В то время как теорема схем предсказывает рост примеров хороших схем, сама теорема весьма упрощенно описывает поведение ГА. Прежде все-

го, $f(S)$ и $f_{ср}$ не остаются постоянными от поколения к поколению. Приспособленности членов популяции знаменательно изменяются уже после нескольких первых поколений. Во-вторых, теорема схем объясняет потери схем, но не появление новых. Новые схемы часто создаются кроссовером и мутацией. Кроме того, по мере эволюции, члены популяции становятся все более и более похожими друг на друга так, что разрушенные схемы будут сразу же восстановлены.

Однако, несмотря на простоту, теорема схем описывает несколько важных аспектов поведения ГА. Мутации с большей вероятностью разрушают схемы высокого порядка, в то время как кроссоверы с большей вероятностью разрушают схемы с большей определяющей длиной. Когда происходит отбор, популяция сходится пропорционально отношению приспособленности лучшей особи к средней приспособленности в популяции; это отношение □□ мера давления отбора. Увеличение или P_c, или P_m или уменьшение давления отбора ведет к увеличению разнообразия выборки или исследованию пространства поиска, но не позволяет использовать все хорошие схемы, которыми располагает ГА. Уменьшение или P_c, или P_m или увеличение давления выбора ведет к улучшению использования найденных схем, но тормозит исследование пространства в поисках новых хороших схем. ГА должен поддержать тонкое равновесие между тем и другим, что обычно известно как проблема "баланса исследования и использования".

4. Решение задачи коммивояжера с помощью генетических алгоритмов

Введение в проблему коммивояжера

Задача коммивояжера в теории дискретной оптимизации считается классической тестовой задачей. Впервые она была сформулирована еще в 1759 году. Суть задачи состоит в том, чтобы найти кратчайший замкнутый путь обхода нескольких городов, заданных своими координатами (или с помощью матрицы расстояний между ними). Города могут посещаться только единожды. Известно, что уже для 50 городов поиск оптимального пути представляет собой сложную задачу, побудившую развитие различных новых методов (в том числе нейросетевых и генетических алгоритмов).

Эта задача NP-полная (задача с экспоненциальной оценкой числа итераций, необходимых для отыскания точного решения) и мультимодальная (имеет локальные экстремумы). ГА используется для нахождения околооптимального пути за линейное время.

Функция приспособленности

Значение функции приспособленности должно соответствовать расстоянию, которое проходит коммивояжер согласно пути, представляемого хромосомой.

Поскольку это значение должно быть минимально, то конечная формула функции приспособленности *j*-й хромосомы часто выглядит следующим образом: $f_j = d_{\max} \cdot 1{,}1 - d_j$, где $d_{\max}$ □□ длина максимального маршрута в текущей популяции d_j □□ длина маршрута, представляемого *j*-й хромосомой.

Значение этой функции чем больше, тем лучше.

Кодирование решений

В настоящее время существует четыре основных представления маршрута коммивояжера в виде хромосомы: *соседское, порядковое, путевое и матричное*. Поскольку классические операторы скрещивания и мутации для них, как правило, неприменимы, каждое из этих представлений имеет собственные "генетические" операторы, все они очень сильно различаются. Некоторые из рассмотренных ниже кроссоверов могут создавать потомство, не имеющее решения (*невалидные* решения). Невалидные решения могут быть полезны для создания и внесения разнообразия в популяцию, однако этим они могут замедлить или даже предотвратить сходимость ГА. Одним из решений этой проблемы может стать идея *восстановления* невалидных решений.

1. Соседское представление

Соседское представление представляет маршрут как список из *n* городов. Город *j* находится на позиции *i* только в том случае, если маршрут проходит из города *i* в город *j*. Например, маршрут со следующим порядком обхода городов: 1→2→4→3→8→5→9→6→7→1 представляется как хромосома *P=[2 4 8 3 9 7 1 5 6*].

Каждый маршрут имеет только одно соседское представление, но не каждый вектор в соседском представлении соответствует какому-либо маршруту. Например, вектор [2 4 8 1 9 3 5 7 6] обозначает последовательность 1→2→4→1..., т.е. часть маршрута □□ это замкнутый цикл, что недопустимо.

Соседское представление не поддерживает классическую операцию кроссовера (он практически всегда порождает недопустимые маршруты). Для него разработаны три собственных оператора.

Кроссовер альтернативных ребер

Кроссовер alternating edges строит потомков, выбирая поочередно первое ребро от первого родителя, потом второе ребро от второго, потом

опять следующее от первого и т.д. Если новое ребро представляет замкнутый цикл, берут ребро из того же родителя (нарушая чередование). Если и оно образует цикл, берут случайное ребро, которое еще не выбиралось и не образует замкнутого цикла. Второй потомок строится аналогично, но первое ребро берут от второго родителя. Для примера рассмотрим построение одного из потомков двух хромосом:

P1 = [2 3 8 7 9 1 4 5 6] ,
P2 = [7 5 1 6 9 2 8 4 3].

Шаг 1. П1 = [2 _ _ _ _ _ _ _ _]. (В маршрут входит ребро 1→2).

Шаг 2. П1 = [2 5 _ _ _ _ _ _ _]. (Маршрут содержит фрагмент 1→2→5).

Шаг 3. П1 = [2 5 8 _ _ _ _ _ _]. (Маршрут содержит фрагменты 1→2→5, 3→8).

Шаг 4. П1 = [2 5 8 6 _ _ _ _ _]. (Маршрут содержит фрагменты 1→2→5, 3→8, 4→6).

Шаг 4. П1 = [2 5 8 6 9 _ _ _ _]. (Маршрут содержит фрагменты 1→2→5→9, 3→8, 4→6).

Шаг 5. П1 = [2 5 8 6 9 1 _ _ _]. (Маршрут содержит фрагменты 4→6→1→2→5→9, 3→8).

Шаг 6. П1 = [2 5 8 6 9 1 4 _ _]. (Маршрут содержит фрагменты 7→4→6→1→2→5→9, 3→8).

Шаг 7. П1 = [2 5 8 6 9 1 4 7 _]. (Маршрут содержит фрагмент 3→8→7→4→6→1→2→5→9). На этом этапе потомок получил ребро 8→7, не присутствующее ни у кого из родителей, т. к. добавление в маршрут 5-го или 4-го родительских городов привело бы к образованию замкнутого подцикла.

Шаг 8. П1 = [2 5 8 6 9 1 4 7 3]. (Маршрут имеет вид: 3→8→7→4→6→1→2→5→9 →3).

Кроссовер фрагментов

Кроссовер subtour chunks создает потомков, выбирая (случайно) фрагмент от одного из родителей, затем случайной длины фрагмент от другого из родителей, и т.д. И опять, если на каком-то уровне образуется замкнутый цикл, он решается аналогичным образом (выбирается другой случайный фрагмент, который еще не выбирался).

Эвристический кроссовер

Heuristic crossovers строит потомков, выбирая случайный город как стартовую точку для маршрута - потомка. Потом он сравнивает два соответствующих ребра от каждого из родителей и выбирает белее короткое. Затем конечный город выбирается как начальный для выбора следующего более короткого ребра из этого города. Если на каком-то шаге получается замкну-

тый маршрут, он продолжается любым случайным городом, который еще не посещался.

Преимущества соседского представления в том, что оно позволяет схематически анализировать создаваемые маршруты. Это представление имеет в основании "строящие блоки" – связи между городами. Например, схема (* * * 3 * 7 * * *) описывает множество всех маршрутов с ребрами (4→3) и (6→7). Основной же недостаток данного представления - в том, что множество всех его операций очень бедно. В частности, для него не существует простых алгоритмов мутации. Кроссовер alternating edges часто разрушает хорошие маршруты, которые были у обоих родителей до применения этой операции. Кроссовер subtour chunks имеет лучшие характеристики, чем первый, благодаря тому, что его разрушительные свойства меньше. Но все равно его эксплуатационные качества все же достаточно низки. Кроссовер heuristic crossover, конечно же, наилучший оператор для данного представления. Причина в том, что предыдущие операции слепы, они не берут в расчет настоящую длину ребер. С другой стороны, heuristic crossover выбирает лучшее ребро из двух возможных. Может получиться, что дальнейший путь будет невозможен и придется выбирать ребро, длина которого неоправданно большая.

2. Порядковое представление

Порядковое представление определяет маршрут как список из *n* городов; *i*-й элемент списка □□ номер от *1* до *n-i-1*. Идея порядкового представления состоит в следующем. Есть упорядоченный список городов, который служит для связи маршрутов с их порядковым представлением. Предположим, что такой упорядоченный список прост: C = [1 2 3 4 5 6 7 8 9]. Тогда маршрут 1→2→4→3→8→5→9→6→7→1 будет представлен как список : L=[1 1 2 1 4 1 3 1 1].

Он может быть интерпретирован следующим образом.

- Так как первый номер списка L равен 1, берем первый город из списка C как первый город маршрута (город номер 1) и исключаем его из списка. Текущий фрагмент маршрута □□ это 1.
- Следующий номер в списке L также 1, поэтому берем первый номер из оставшегося списка C=[2 3 4 5 6 7 8 9]. Так как мы исключили из списка C 1-й город, следующий город - 2. Исключаем и этот город из списка. Маршрут на данном шаге приобретает вид: 1 → 2.
- Следующий номер в списке L- 2. Берем из списка C = [3 4 5 6 7 8 9] 2-ой по порядку оставшийся город. Это - 4. Исключаем его из списка. Маршрут: 1 → 2 → 4.
- Следующий номер в списке L - 1. Берем из списка C =[3 5 6 7 8 9] 1-й город – с номером 3. Имеем маршрут : 1 →2 → 4 → 3.

- Следующий в списке L - 4, берем 4-й город из оставшегося списка C =[5 6 7 8 9] (это 8). Маршрут: 1 →2 → 4 → 3 → 8.
- Следующий номер в списке L - 1. Берем из списка C = [5 6 7 9] 1-й город – с номером 5. Маршрут: 1 →2 → 4 → 3 → 8→ 5.
- Следующий номер в списке L - 3. Берем третий город из списка C=[6 7 9] (это 9). Удаляем его из C. Маршрут: 1 →2 → 4 → 3 → 8→ 5→9.
- Следующий номер в списке L - 1, поэтому берем первый город из текущего списка C = [6 7] (город номер 6), и удаляем его из C. Частичный маршрут имеет вид: 1 →2 → 4 → 3 → 8→ 5→9→6.
- Последним номером в списке L всегда будет 1, поэтому берем последний оставшийся город из текущего списка C = [7], и удаляем его из C. Окончательно маршрут имеет вид: 1 →2 → 4 → 3 → 8→ 5→9→6→7→1.

Основное преимущество порядкового представления в том, что классический кроссовер в данном случае работает! Любые два маршрута в порядковом представлении, разрезанные в любой позиции и склеенные вместе, породят два потомка, каждый из которых будет правильным маршрутом. Например, два родителя P1 = [1 1 2 1| 4 1 3 1 1] и P2 = [5 1 5 5 | 5 3 3 2 1], которые обозначают соответственно маршруты 1 → 2 → 4 → 3 → 8 → 5 → 9 → 6 → 7 →1 и 5 → 1 → 7 → 8 → 9 → 4 → 6 → 3 → 2→5, с точкой разреза, обозначенной " | " породят следующих потомков: П1 = [1 1 2 1 5 3 3 2 1] и П2 = [5 1 5 5 4 1 3 1 1].
Эти потомки обозначают маршруты 1 →2 → 4 → 3 → 9 → 7 → 8 → 6 → 5 →1 и 5 → 1 → 7 → 8 → 6 → 2 → 9 → 3 → 4→5.

Легко заметить, что части маршрута слева от линии разреза не изменились, тогда как части маршрута справа от линии разреза имеют достаточно много отличий от окончаний родительских хромосом.

В качестве ***мутации*** используется изменение с вероятностью p_m *i*-го элемента порядкового представления на случайный номер от *1* до *n-i-1*

3. Путевое представление

Путевое представление – это наиболее естественное представление маршрута. Например, тур 5 → 1 → 7 → 8 → 9 → 4 → 6 → 2 → 3 →5 будет представлен как строка [5 1 7 8 9 4 6 2 3].

Частично отображаемый кроссовер

Частично отображаемый кроссовер берет сначала часть пути одного родителя и сохраняет последовательность и позиции как можно большего числа городов другого родителя. Точка кроссовера выбирается случайно.

$$V_1 = 12 \mid 543$$
$$V_2 = 35 \mid 421$$

Рис.8. Точка сечения частично отображаемого кроссовера

Затем производится отображение замен первых частей хромосом $\{1 \Leftrightarrow 3, 2 \Leftrightarrow 5\}$, которое применяется поточечно к родителям для получения потомства. Проверяется каждый элемент первого родителя: если имеется для него замена, то она производится и затем копируется ($\Updownarrow$) в первого потомка , в противном случае он просто копируется без замены ($\Downarrow$) (рис.9). Подобная процедура может быть использована и для второго потомка.

$$V_1 = 1\ 2\ 5\ 4\ 3$$
$$\Updownarrow\ \Updownarrow\ \Updownarrow\ \Downarrow\ \Updownarrow$$
$$V_6 = 3\ 5\ 2\ 4\ 1$$

Рис.9. Частично отображаемый кроссовер

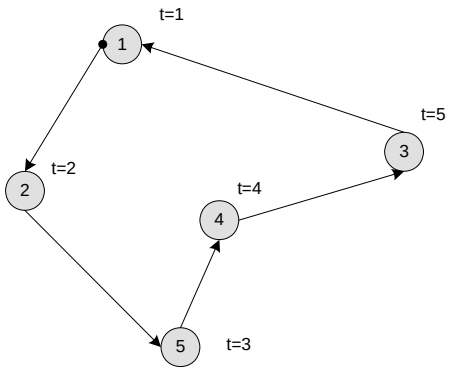

Рис.10. Родитель V_1

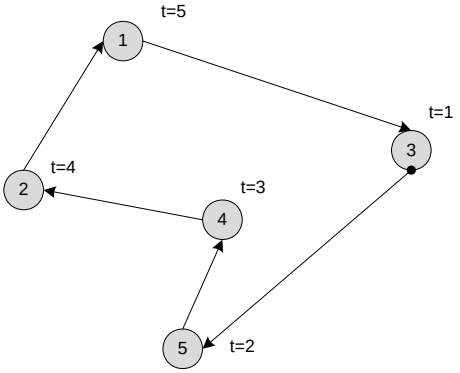

Рис.11. Родитель V_2

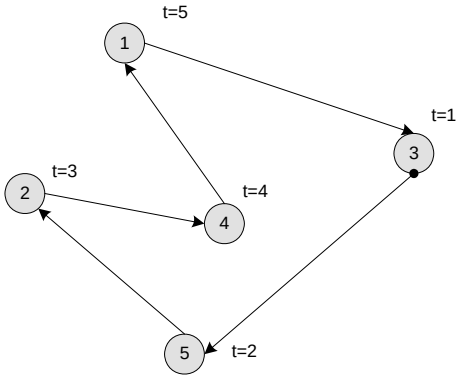

Рис.12. Потомок V_6

Упорядоченный кроссовер

Упорядоченный кроссовер берет часть пути одного родителя и сохраняет родственный порядок городов из другого родителя. Первые две точки сечения кроссовера выбираются случайно (рис. 13). Каждый элемент центральной секции первого родителя копируется в потомка (рис.14).

$$V_1 = 1\ |\ 2\ 5\ |\ 43$$
$$V_2 = 3\ |\ 54\ |\ 21$$

Рис.13. Точки сечения кроссовера

$$V_1 = 1\ 2\ 5\ 4\ 3$$
$$\Downarrow\ \Downarrow$$
$$V_7 = \quad 2\ 5$$

Рис.14. Создание нового потомка

Затем элементы второго родителя собираются в список (рис.15), начиная со

второй точки кроссовера. Наконец, города, уже представленные в потомках, удаляются (рис.16), и оставшиеся элементы копируются вместо пустых пробелов потомка, начиная со второй точки сечения кроссовера (рис. 17).

[2 1 3 5 4]

Рис.15. Список городов во втором родителе

[2 1 3 5 4] => [1 3 4]

Рис.16. Удаление дублируемых городов

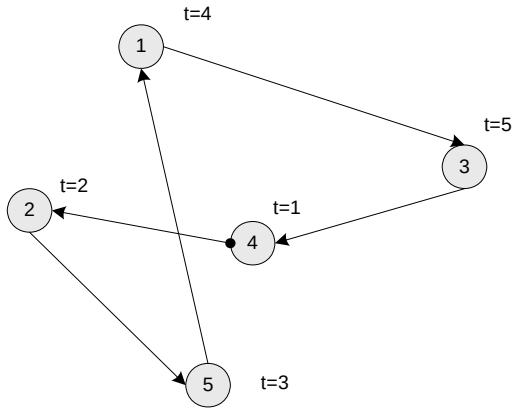

Рис.17. Потомок $V_7 = 42513$

Циклический кроссовер

В циклическом кроссовере каждый город берется от одного из родителей. Первый город потомка берется от первого родителя. Второй город потомка берется у второго родителя из последней позиции (рис.19). В данном случае мы не можем взять его от второго родителя, так как этот город уже находится в хромосоме, поэтому мы его берем от первого родителя, и т.д. до тех пор, пока новая хромосома не будет создана.

$V_1 = 1\ 2\ 5\ 4\ 3$

$V_6 = 3\ 5\ 4\ 2\ 1$

$\Downarrow$

$V_8 = 1$

Рис.18. Копирование города из 1-го родителя

$V_1 = 1\ 2\ 5\ 4\ 3$

$V_6 = 3\ 5\ 4\ 2\ 1$

$\Downarrow \qquad \Downarrow$

$V_8 = 1 \qquad 3$

Рис.19. Копирование следующего города

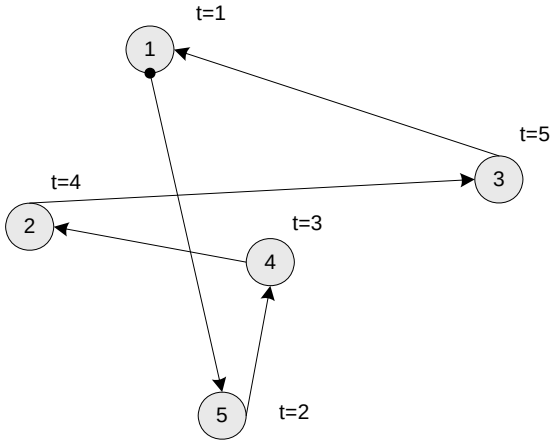

Рис. 20. Потомок $V_8 = 15423$

Кроссовер рекомбинации ребер

Кроссовер рекомбинации ребер делает потомство только с помощью ребер, представленных в обоих родителях. Сначала из двух родителей

(рис.21) строится список ребер (табл.4). Города в потомках выбираются по одному, при этом выбирается город с наименьшим количеством ребер. Первым элементом в хромосоме является город с маленьким количеством связей (рис.22). После того как город выбран, он удаляется из таблицы, и города, присоединенные к нему, рассматриваются как кандидаты при следующем выборе.

$V_1 = 1\,2\,5\,4\,3$

$V_2 = 3\,5\,4\,2\,1$

Рис.21. Исходные родители

Список ребер

Город	Соединен с		
1	2	3	
2	1	5	4
3	4	1	5
4	5	3	2
5	2	4	3

$V_9 = 1\ \_\ \_\ \_\ \_$

Рис.22. Хромосома на первом шаге алгоритма

Список ребер после первого шага

Город	Соединен с		
2	5	4	
3	4	5	
4	5	3	2
5	2	4	3

$V_9 = 1\ 3\ \_\ \_\ \_$

Рис.23. Хромосома на втором шаге

Список ребер после второго шага

Город	Соединен с	
2	5	4
4	5	2
5	2	4

$V_9 = 1\ 3\ 5\ \_\ \_$

Рис.24. Хромосома на третьем шаге

Список ребер после третьего шага

Город	Соединен с
2	4
4	2

$V_9 = 1\ 3\ 5\ 2\ \_$

Рис.25. Хромосома на четвертом шаге

Список ребер после четвертого шага

Город	Соединен с
4	

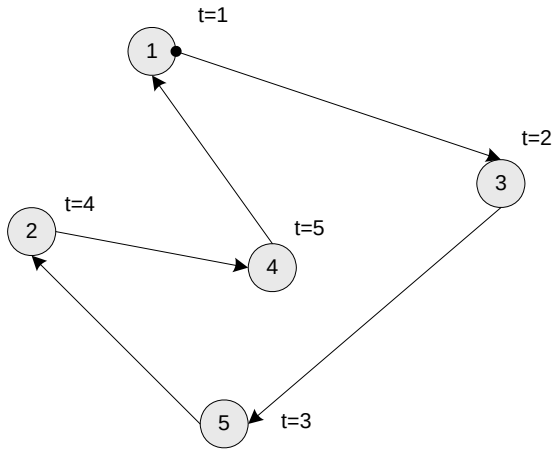

Рис.26. Полученный потомок $V_9 = 13524$

Этот тип кроссовера создает хромосомы, более чем в 95% случаев соответствующие каким-либо маршрутам.

Улучшенный кроссовер рекомбинации ребер

Улучшенная версия кроссовера рекомбинации ребер состоит в сохранении записи ребер, представленных в обоих родителях и способствующих их выбору.

$V_1 = 1\,2\,5\,4\,3$

$V_2 = 3\,5\,4\,2\,1$

Рис.27. Исходные родители

Список ребер

Город	Соединен с		
1	2	3	
2	1	5	4
3	4	1	5
4	5	3	2
5	2	4	3

$V_{10} = 1 \_ \_ \_ \_$

Рис.28. Хромосома на первом шаге алгоритма

Список ребер после первого шага

Город	Соединен с		
2	5	4	
3	4	5	
4	5	3	2
5	2	4	3

$V_{10} = 1\,3 \_ \_ \_$

Рис.29. Хромосома на втором шаге

Список ребер после 2-го шага

Город	Соединен с	
2	5	4
4	5	2
5	2	4

$V_{10} = 1\,3\,5 \_ \_$

Рис.30. Хромосома на третьем шаге

Список ребер после 3-го шага

Город	Соединен с
2	4
4	2

$V_{10} = 1\,3\,5\,4 \_$

Рис.31. Хромосома на четвертом шаге

Список ребер после 4-го шага

Город	Соединен с
4	2

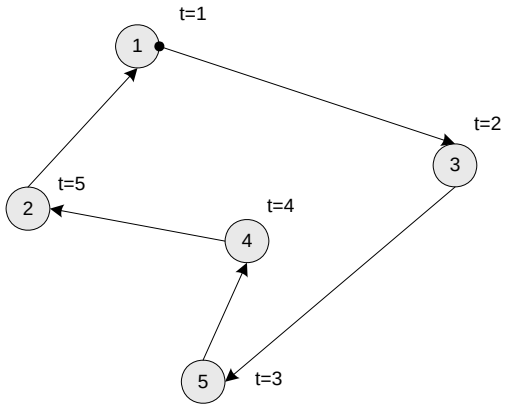

Рис.32. Полученный потомок $V_{10} = 13542$

Измененный кроссовер

Работа этого кроссовера может быть описана следующим образом. Выбирается точка сечения (рис. 33), затем первая часть одного родителя копируется в первого потомка (рис. 34). Во вторую часть потомка копируются гены второго родителя (рис. 35). Если такие гены уже встречаются в потомке, то они пропускаются, а оставшуюся часть потомка дополняют гены первого родителя. Аналогичный алгоритм и для второго потомка.

$$V_1 = 12 \mid 543$$
$$V_2 = 35 \mid 421$$

Рис.33. Точка сечения кроссовера

$V_{11} = 1\,2\,\_\,\_\,\_$
Рис.34. Потомок V_{11}

$V_{11} = 1\,2\,4\,\_\,\_$
Рис.35. Потомок V_{11}

$V_{11} = 1\,2\,4\,5\,3$
Рис.36. Потомок V_{11}

Мутация

Мутация работает по следующему принципу. Выбираются случайным образом два города в хромосоме и меняются местами.

$$V_1 = 1\,2\,5\,4\,3$$
$$V_1' = 1\,3\,5\,4\,2$$

Рис.37. Мутация хромосомы V_1 (2 и 5 элемент меняются местами)

“Жадная мутация”

“Жадная мутация” (*greedy reconnection*) состоит в упорядочивании последовательности городов. Ее применение помогает хорошо искать локальные оптимумы. Сначала случайным образом выбираются две точки сечения в хромосоме (рис.38) так, чтобы между ними было по крайне мере 2 города. Затем последовательность городов между точками сечения упорядочивается: они переставляются в зависимости от близости друг к другу. В нашем случае среди городов 2, 5 и 4 определяется город, ближайший к городу 1. Пусть, например, это город 5. Он ставится следом за городом 1. Затем среди городов 2 и 4 определяется ближайший к городу 5. Пусть, например, это го-

род 4. Ставим его за городом 5. На последнее место ставим оставшийся город 2. В результате имеем модифицированную хромосому (рис. 41).

$V_1 = 1|254|3$

$V_1' = 1\ \_\ \_\ \_\ 3$

Рис.38. Шаг 1

$V_1' = 15\_\_3$

Рис.39. Шаг 2

$V_1' = 154\_3$

Рис.40. Шаг 3

$V_1' = 15423$

Рис.41. Потомок, полученный при мутации V_1

4. Матричное представление

Для кодирования хромосомы также может служить бинарная матрица V, где $V_{ct} = 1$, если город c посещен в момент времени t (находится в маршруте в позиции *t*), в противном случае $V_{ct} = 0$.

Например, пути ABEDC и CEDBA могут быть представлены в виде следующих матриц (строки-города, столбцы-время):

Таблица пути ABEDC

V_1=

	1	**2**	**3**	**4**	**5**
A	1	0	0	0	0
B	0	1	0	0	0
C	0	0	0	0	1
D	0	0	0	1	0
E	0	0	1	0	0

Таблица пути CEDBA

V_2=

	1	**2**	**3**	**4**	**5**
A	0	0	0	0	1
B	0	0	0	1	0
C	1	0	0	0	0
D	0	0	1	0	0
E	0	1	0	0	0

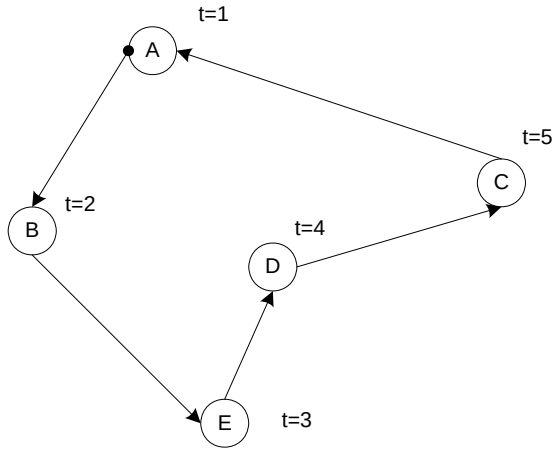

Рис.42. Путь ABEDC

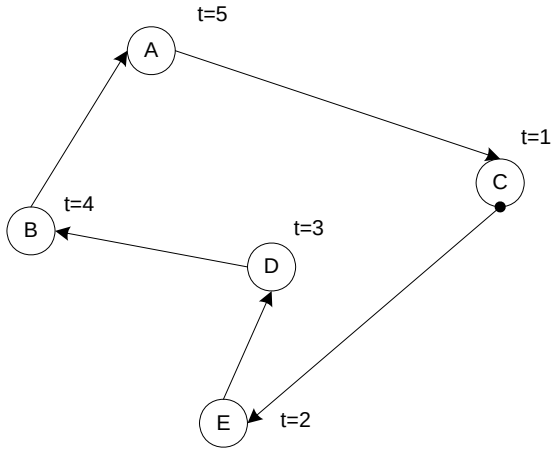

Рис.43. Путь CEDBA

Матрицы могут быть представлены в виде следующих хромосом:

$$V_1 = 100000100000001000100100$$

$$V_2 = 000010001010000001000100$$

Необходимо отметить, что города в одинаковой последовательности, но с разными стартовыми точками или с противоположным направлением кодируются разными матрицами, т.е. разными хромосомами. Например:

$V_3(BEDCA) = 000011000000010001000 1000 \neq V_1 \quad (ABEDC \equiv BEDCA)$.

ГА может создать некоторые хромосомы, не имеющие путевого представления (*невалидные* решения). Это может произойти при создании начальной популяции и при действии стандартных генетических операторов . Например, при одноточечном кроссовере может возникнуть следующая ситуация:

$$V_1 = 100000100000 \mid 0010001000100,$$
$$V_2 = 000010001010 \mid 0000010001000,$$
$$V_4 = 100000100000 \;\; 0000010001000.$$

Рис.44. Пример скрещивания, в результате которого получилось неверное решение V_4

Очевидно, что такое решение неправильно, т.к. город С ни разу не посещен.

Таблица неверного решения

$V_4 =$

	1	**2**	**3**	**4**	**5**
A	1	0	0	0	0
B	0	1	0	0	0
C	0	0	0	0	0
D	0	0	1	0	0
E	0	1	0	0	0

Также неправильное решение появится в результате скрещивания V_1 и V_3.

В этом случае необходимо проводить *восстановление* решений (добавлять единицы в недостающих местах или убирать лишние). Например, восстановленное решение V4 может иметь вид:

$$V_4 = 10000\ 01000\ 00010\ 00100\ 00001.$$

Фокс (Fox) и Макмахон (McMahon) рассматривали маршрут как двоичную матрицу, в которой элемент X_{ij} содержит 1, если город *i* стоит в маршруте раньше, чем город *j*.

Существует также альтернативное матричное представление, при котором в матрицу в элемент X_{ij} записывается 1 только в том случае, если *i*-й город непосредственно предшествует *j*-му(является соседом, стоящим непосредственно перед j-м городом).

Для этих представлений также разработаны способы скрещивания и мутации, но они достаточно сложны, поэтому в алгоритмах чаще используются рассмотренные ранее способы символьного кодирования хромосом.

5. Непрерывные генетические алгоритмы

Математический аппарат непрерывных ГА

При работе с оптимизационными задачами в непрерывных пространствах вполне естественно представлять гены напрямую вещественными числами. В этом случае хромосома есть вектор вещественных чисел. 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 непрерывных 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 непрерывных алгоритмов для представления решений удобно, поскольку близко к постановке большинства прикладных задач. Кроме того, отсутствие операций кодирования/декодирования, которые необходимы в классическом ГА, 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 : пропорциональный, турнирный, отбор усечением. Однако операторы скрещивания и мутации не годятся: в классических реализациях они работают с битовыми строками. Нужны собственные реализации, учитывающие специфику непрерывных алгоритмов. Оператор скрещивания непрерывного ГАпорождает одного или нескольких потомков от двух хромосом. Собственно говоря, требуется из двух векторов вещественных чисел получить новые векторы по каким-либо законам. Большинство непрерывных 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.

Плоский кроссовер (flat crossover)

Создается потомок $H = (h_1, \ldots, h_k, \ldots, h_n)$, где h_k, $k=1,\ldots,$ n – случайное число из интервала $[c_k^{min}, c_k^{max}]$, $c_k^{min} = \min(c_k^1, c_k^2)$, $c_k^{max} = \max(c_k^1, c_k^2)$.

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 $h_k^1 = w*c_k^1 + (1-w)*c_k^2$, $h_k^2 = w*c_k^2+(1-w)*c_k^1$, $k=1,\ldots,n$, 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 $h_k^1 = (c_k^1)^w(c_k^2)^{1-w}$, $h_k^2 = (c_k^2)^w(c_k^1)^{1-w}$, w – случайное число из интервала [0;1]. (Применяется только если векторы C_1 и C_2 неотрицательны)

Смешанный кроссовер (alpha crossover)

Генерируется потомок $H = (h_1,\ldots,h_k,\ldots,h_n)$, где h_k – случайное число из интервала $[c_k^{min}-I_k*alpha, c_k^{max}+I_k*alpha]$, $c_k^{min}=\min(c_k^1,c_k^2)$, $c_k^{max}=\max(c_k^1,c_k^2)$, $I_k=c_k^{max}-c_k^{min}$. При *alpha* =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 приспособленных потомка.

Дискретный кроссовер (discrete crossover)

Каждый ген h_k выбирается случайно по равномерному закону из конечного множества $\{c_k^1,c_k^2\}$.

Расширенный линейный 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].

В качестве оператора ***мутации*** наибольшее распространение получила случайная мутация. При *случайной мутации* ген, подлежащий изменению, принимает случайное значение из интервала своего изменения.

Рассмотренные кроссоверы исторически были предложены первыми, однако во многих задачах их эффективность оказывается невысокой. Исключение составляет *смешанный 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. 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 $h_i^1 = 0.5\,[(1-\beta_i)x_i^1 + (1+\beta_i)x_i^2]$, $h_i^2 = 0.5\,[(1+\beta_i)x_i^1 + (1-\beta_i)x_i^2]$, $\beta_i \ge 0$ – 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, $n \in [2,5]$ – 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.

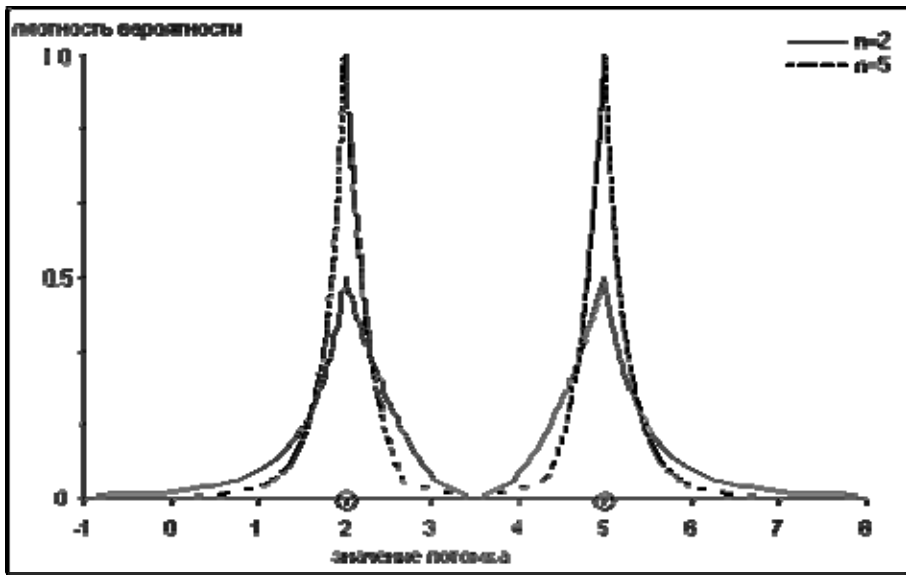

Рис. 43. Геометрическая интерпретация работы SBX кроссовера

Эксперименты автора SBX кроссовера показали, что он во многих случаях эффективнее *смешанного кроссовера*, хотя не существует ни одного кроссовера, эффективного во всех случаях. Исследования показывают, что использование нескольких 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. В любом случае, если стоит задача оптимизации в непрерывных пространствах с использованием эволюционных стратегий, то следует сделать выбор в пользу непрерывного генетического алгоритма.

6. Генетическое программирование

Идею ***генетического программирования*** (ГП) впервые предложил Коза в 1992 году, опираясь на концепцию генетических алгоритмов. Генетическое программирование − это расширение генетической модели обучения в область программного обеспечения. Его объектом в отличие от генетических алгоритмов являются не символьные строки фиксированной длины, кодирующие возможные решения проблемы, а собственно программы, исполняя которые и получают различные варианты решения задачи. В генетическом программировании программы представляются в виде дерева грамматического разбора, а не в виде строк кода, поэтому в ГП все операции производятся не над строками, а над ***деревьями***. При этом используются такие же операторы, как и в ГА: селекция, скрещивание и мутация.

В ГП хромосомами являются ***программы***. Программы представлены как деревья с ***функциональными*** (промежуточными) и ***терминальными*** (конечными) элементами. Терминальными элементами являются константы, действия и функции без аргументов, функциональными - функции, использующие аргументы.

Для примера можно рассмотреть функцию *2+x*4/7*, представленную на рисунке 44. Терминальные элементы $T = \{2,x,4,7\}$, функциональные $F = \{+,*,/\}$.

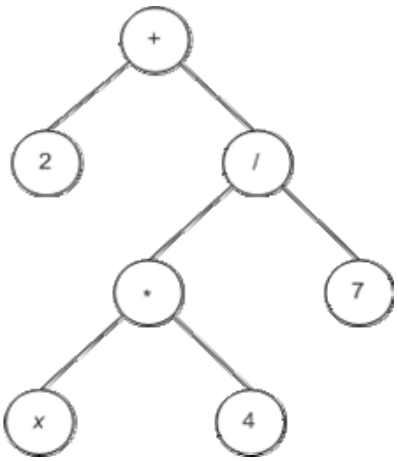

Рис. 44. Древовидное представление функции *2+x*4/7*

Для того чтобы применить ГП к какой-либо проблеме, необходимо определить:

a) множество терминальных элементов;
b) множество функциональных элементов;
c) меру приспособленности;
d) параметры, контролирующие эволюцию;
e) критерий останова эволюции.

Генетические программы не пишутся программистом, а создаются в результате следующего итерационного пошагового процесса.

1. Генерируется начальная популяция программ, представляющих собой случайные композиции функций (арифметических, логических и др. операций) и терминалов (переменных и констант), взятых из множеств функциональных и терминальных элементов, относящихся к решаемой проблеме. Если мы собираемся вывести программу, которая управляет стрельбой из по движущейся цели, то в набор терминалов войдут следующие переменные - расстояние до цели, ее размер и скорость, сила и направление ветра, тип орудия и т.д. Набор функций в этом случае будет включать различные математические операции, как простые (умножение, деление, вычитание, сложение), так и более сложные.
2. Каждая программа выполняется и ей присваивается значение приспособленности, соответствующее тому, насколько хорошо она решает задачу.
3. Создается новая популяция компьютерных программ за счет:
   1) копирования в нее наилучших программ старой популяции;
   2) создания новых программ с помощью мутаций (случайного изменения функций и терминаторов или даже целых поддеревьев);
   3) создания новых программ с помощью скрещивания существующих.

   Операция скрещивания реализуется за счет обмена поддеревьев двух хромосом, выбранных случайно (в соответствии с их приспособленностью).

4. Все программы снова выполняются и цикл повторяется до тех пор, пока не будет получен необходимый результат.

Особенности операторов ГП

Алгоритм работы ГП такой же, как и ГА: селекция, скрещивание и мутация. Однако поскольку ГП оперирует над деревьями, а не над строками, то операторы скрещивания и мутации имеют отличия.

Скрещивание

Оператор скрещивания работает следующем образом: выбираются случайные части родительских деревьев, и эти части меняются местами.

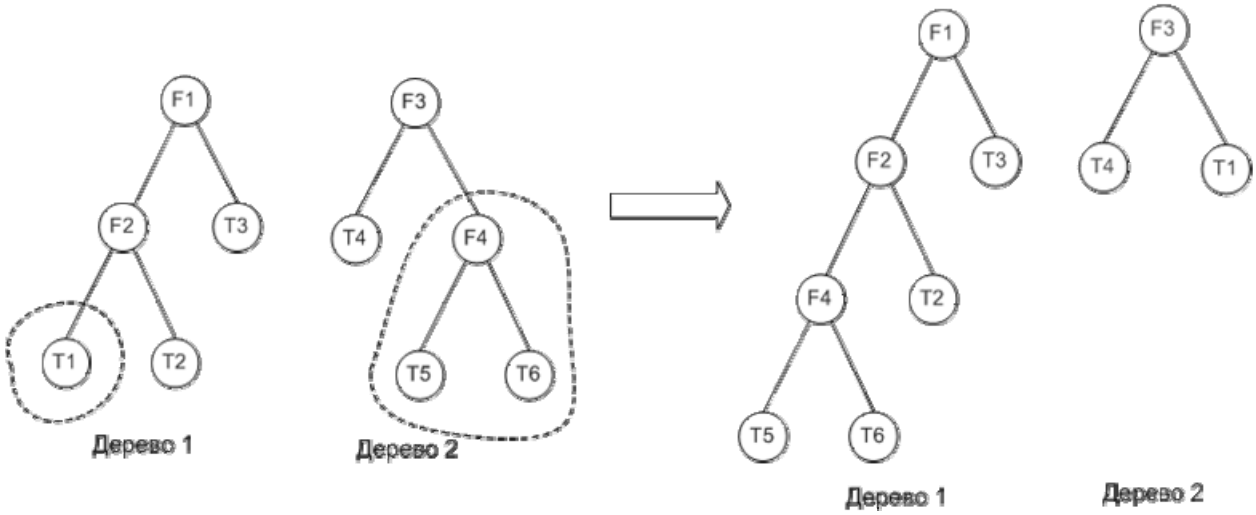

Рис. 45. Скрещивание двух деревьев

В качестве особенности необходимо отметить, что в ГП размер хромосомы меняется. Чтобы предотвратить чрезмерное разрастание дерева, вводят максимальное количество функциональных элементов в дереве или максимальную глубину дерева. Однако при операции скрещивания возможна ситуация, когда при скрещивании двух деревьев получится одно из деревьев, превосходящее заданный лимит. В этом случае вместо конфликтного дерева копируется родительское дерево.

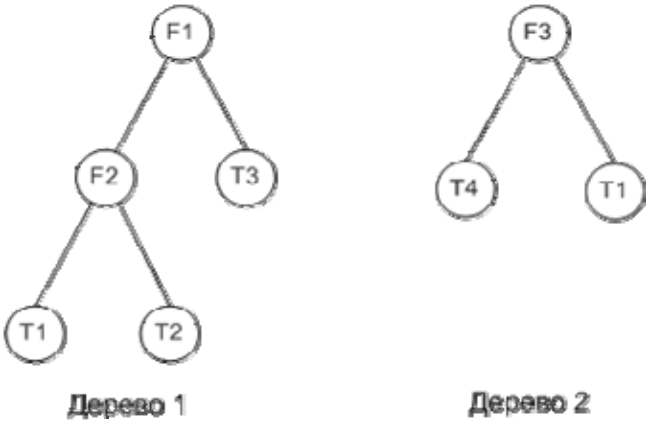

Рис. 46. Разрешение конфликтной ситуации предыдущего оператора скрещивания при максимальной глубине дерева, равной трем

Также следует отметить, что часто при применении оператора скрещивания появляются **интроны** − ничего не делающие участки кода (например, вычисляющие выражения вида: x:=x*1). На первый взгляд, время, которое ГП тратит на увеличение и развитие интронов, проходит впустую. С другой стороны, интроны помогают сохранять от разрушения хорошие блоки для будущих поколений, что повышает шансы на получение еще более эффективных особей.

Мутация

Оператор мутации случайно удаляет часть дерева и заменяет ее новым деревом.

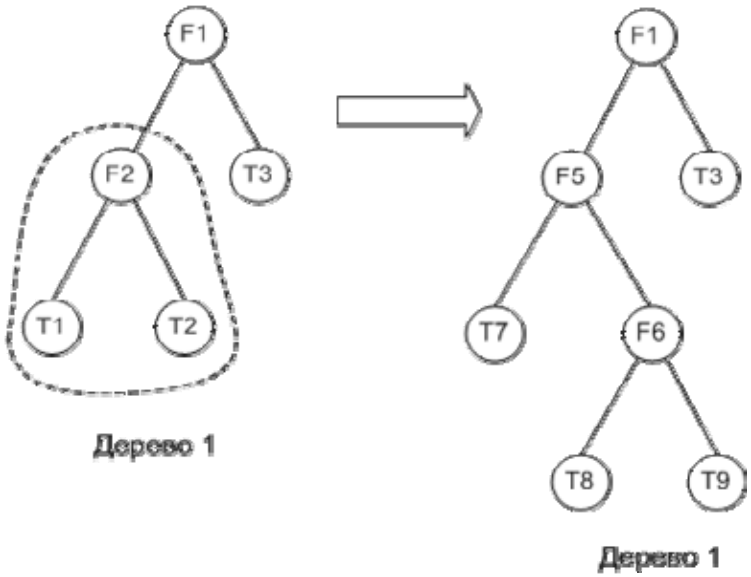

Рис. 47. Оператор мутации

Применение ГП

Классифицирующие системы (classifier systems) − очень интересное направление, изучающее вопросы создания самообучающихся машин. Один из его создателей Дж. Холланд предложил для этих целей использовать когнитивную систему, способную классифицировать состояние окружающей среды и соответственно реагировать на это состояние. Что необходимо для построения такой системы? Очевидно, (1) −среда; (2) − рецепторы, которые будут сообщать системе о том, что происходит; (3) −эффекторы, которые позволят нашей системе манипулировать средой; и (4) − собственно система, "черный ящик", кот рый имеет (2) и (3) и "живет" внутри (1). Такие системы часто называют «аниматами» (animal + robot = animat). Основная идея классифицирующих систем − начав с нулевого знания, используя случайно сгенерированную классифицирующую популяцию, позволить системе создать свою программу с помощью индукции. Она приводит входной поток к некому шаблону, который позволяет анимату классифицировать свое текущее состояние/контекст и реагировать соответствующим образом.

ПРИЛОЖЕНИЕ

Математическая модель и генетический алгоритм для решения задачи составления школьного расписания

Как известно, задача составления школьного расписания является плохоформализуемой и труднорешаемой. Школьное расписание является довольно сложной системой, все элементы которой взаимосвязаны между собой напрямую или косвенно. Помимо большого количества условий, которые приходится строго выполнять, необходимо оптимизировать множество параметров, влияющих на качество расписания. Поскольку интересы участников учебного процесса многообразны, задача составления расписания по сути еще и многокритериальная. Размерность задачи, не позволяющая надеяться на отыскание ее точного решения, и сложность объекта, для которого строится математическая модель, обусловливают необходимость серьезного математического исследования задачи с целью увеличения функциональных возможностей алгоритмов составления расписаний.

Рассмотрим одну из возможных постановок задачи составления школьного расписания и метод ее решения, основанный на использовании генетического алгоритма. Представленный алгоритм, хотя и является приближенным, позволяет выполнить все обязательные условия, учитывая при этом многокритериальность поставленной задачи.

Постановка задачи

Формально задачу составления учебного расписания можно поставить следующим образом.

Заданы множество преподавателей P, множество учебных групп K, множество учебных дисциплин (предметов) U, учебный план, фиксирующий количество часов каждого предмета в каждом классе, множество кабинетов (аудиторий) A. Требуется найти расписание, обеспечивающее проведение всех занятий без накладок относительно аудиторий, без пустых уроков (“окон”) для учащихся и удовлетворяющее обязательным требованиям преподавателей. Кроме этих необходимых условий, на расписание накладываются дополнительные требования такие, как, например, выполнение норм СанПин, касающихся правил распределения нагрузки в течение учебной недели и расстановки уроков в течение учебного дня.

Элементом расписания является учебная дисциплина (предмет), которой ставится в соответствие учебная группа и преподаватель, а так же указывается, требуется ли для ее проведения специальная аудитория (например, компьютерный класс или химическая лаборатория) или нет. Причем считается, что если для какого-то класса предусмотрено 4 урока математики, то это 4 разные элемента расписания. Все предметы перенумеровываются и за-

носятся в таблицу. Здесь n – общее число предметов, q - количество учебных групп, p – общее количество уроков, которые можно провести в течение недели (например: пятидневка, по 6 уроков в день, $p = 30$).

Заполненная таблица представляет собой некоторое расписание. Заштрихованные ячейки означают, что они не участвуют в построении расписания. То есть, например, в группе 1 в понедельник предусмотрено 4 урока, в группе 2 − пять и т.д. Заранее зафиксированная структура расписания гарантирует отсутствие "окон" у школьников и позволяет изначально выполнять некоторые требования СанПин (например, что максимальное количество уроков должно быть в среду и т.д.) .

Представление учебного расписания

День недели	Номер урока	Учебные группы			
		Гр.1	Гр.2	…	Гр. q
Понедельник	1	1	g_1+1	…	$g_{q-1}+1$
	2	2	…	…	…
	3	3	…	…	…
	4	4	…	…	…
	5		…	…	…
	6			…	
…	…	…	…	…	
…	p	g_1	g_2	…	n

Для каждого предмета k ($k = \overline{1,n}$) вводится в рассмотрение множество *I(k)* – номера уроков, на которых он может проводиться (в соответствии с учебным планом или обязательными требованиями преподавателя). Соответственно, множества $K_j(i)$ - это номера предметов j–й уч. группы, которые могут быть проведены на i-ом уроке.

Рассмотрим матрицу совместимости предметов S, которая вводится следующим образом:

$$S_{kr} = \begin{cases} 0, \textit{если предмет k совместим с предметом } r \\ 1, \textit{иначе}, r = \overline{1,n}, k = \overline{1,n}. \end{cases}$$

Предметы являются несовместимыми, если они используют общий ресурс (преподаватель или кабинет). Такие предметы не могут находиться в расписании на одной строке.

Обязательные требования к расписанию могут быть описаны следующей математической моделью:

$$\sum_{i=1}^{p}\sum_{j=1}^{q} \sum_{g_{j-1}<k\le g_j} \sum_{l>g_j} S_{kl}\, z_{ki} z_{li} \to \min$$

$$\sum_{i\in I(k)} z_{ki} = 1, \quad z_{ki} = 0,\ i \notin I(k), k = \overline{1,n}$$

$$\sum_{k \in K_j(i)} z_{k\,i} = 1, \quad i = \overline{1, p}\,, j = \overline{1, q}$$

Здесь $z_{k\,i} = \begin{cases} 1, \textit{к – й предмет преподается на i – м уроке}. \\ 0, \textit{иначе;}\ k = \overline{1, n}\ i = \overline{1, p}\,. \end{cases}$

Целевая функция минимизирует количество несовместимых предметов преподающихся на каждом уроке. Первая группа ограничений отвечает за то, что каждый предмет k проводится только один раз, причем на допустимом уроке. Вторая группа ограничений следит за тем, чтобы в каждой группе j на каждом уроке i проводится только один предмет. Заметим, что в результате получилась модель, похожая на постановку квадратичной задачи о назначениях [1], причем оптимальное значение целевой функции известно заранее и равно 0. Все дополнительные требования к расписанию, не учтенные в модели, выполняются алгоритмически.

Генетический алгоритм

Для построения генетического алгоритма необходимо выбрать способ представления данных в виде хромосом. В данном случае хромосома будет матрицей. Ее элемент $y_{i\,j} = k$, если у *j*-й группы на *i*-м уроке проводится предмет k ($y_{i\,j} = 0$, если у *j*-й группы нет *i*-го урока). Связь с переменными $z_{k\,i}$ в модели: $y_{i\,j} = k$, *если* $z_{k\,i} = 1$, $g_{j-1} < k \le g_j$, $i \in \overline{1, p}, j \in \overline{1, q}, k \in K_j(i)$. В качестве стратегии отбора в данной задаче используется турнирный отбор (он оказался для нее более эффективным, чем пропорциональный). Поэтому функцию приспособленности можно не максимизировать, а уменьшать от популяции к популяции. В ее роли выступает целевая функция задачи с добавками в виде штрафов за нарушение дополнительных ограничений. (Например, водится матрица : l_{ij} =1, если по нормам СанПин предметы *i,j* не могут преподаваться в один день; l_{ij} =0, если могут, и рассчитывается количество нежелательных ситуаций в оцениваемом расписании. И т.п.) Полученные значения добавляется в функцию приспособленности (возможно, с коэффициентами, характеризующими их значимость). При скрещивании происходит обмен фрагментами внутри одного выбранного столбца матрицы *Y*. В качестве оператора скрещивания используется циклический кроссовер. Он гарантирует допустимость получаемых после скрещивания хромосом, т. к. каждый ген потомка будет находиться на том месте, на котором он находился в одном из родителей. Для мутации внутри одного столбца выбирается любая пара генов таких, что их обмен не нарушит допустимости хромосомы, затем эти гены меняются местами. Проведенные эксперименты показывают эффективность генетического алгоритма построения расписания.

ЛИТЕРАТУРА

Основная

1. Гладков Л.А. Генетические алгоритмы / Л.А. Гладков, В.В. Курейчик, В.М.Курейчик. –М : Физматлит, 2006 г. -402 с.
2. Емельянов В.В. Теория и практика эволюционного моделирования/ В.В. Емельянов, В.В. Курейчик, В.М. Курейчик. –М : Физматлит, 2003 г.- 431 с.
3. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. – М : Горячая Линия - Телеком, 2006 г. - 452 с.
4. Батищев Д.И. Генетические алгоритмы решения экстремальных задач: учеб. пособие / Д.И. Батищев. – Воронеж: ВГТУ, 1995. - 69 с.

Дополнительная

1. Каширина И.Л. Генетический алгоритм решения квадратичной задачи о назначениях специального вида// Вестник ВГУ. Серия физика, математика, 2003, № 1, с. 128-131.
2. Гигиенические требования к условиям обучения школьников в различных учебных заведениях. Приложение № 6. (СанПин, 542-96).
3. 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.

Учебное издание

Каширина Ирина Леонидовна

ВВЕДЕНИЕ В ЭВОЛЮЦИОННОЕ МОДЕЛИРОВАНИЕ

Учебное пособие

Редактор Е.С. Котлярова